МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 13

Под общей редакцией
чл.-корр. РАН Л.Н. Королева

МАКС ПРЕСС

МОСКВА – 2012

УДК 519.6+517.958
ББК 22.19
П75

Печатается по решению
Редакционно-издательского совета факультета
вычислительной математики и кибернетики МГУ имени М.В. Ломоносова

П75 **Программные системы и инструменты:** Тематический сборник/ Под ред. Королева Л.Н. – М: Издательский отдел факультета ВМиК МГУ (лицензия ИД №05899 от 24.09.2001 г.); МАКС Пресс, 2012. – № 13. – 268 с.
ISBN 978-5-89407-496-2
ISBN 978-5-317-04314-8

В данный сборник включены научные работы и сообщения, связанные с общими вопросами программирования и информатики, которые могут оказаться полезными во многих практических приложениях.

В этих публикациях нашли отражение исследования и разработки в области создания программных систем, выполненных учеными, аспирантами и студентами факультета. Большая часть результатов доложена на Ломоносовских Чтениях 2012 года.

Статьи сборника будут интересны студентам, аспирантам и специалистам в области разработки прикладных программных систем с использованием новых информационных технологий.

УДК 519.6+517.958
ББК 22.19

ISBN 978-5-89407-496-2
ISBN 978-5-317-04314-8

СОДЕРЖАНИЕ

От редколлегии:

В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные описаниям инструментальных программных систем, разработанных авторами публикаций, относящихся к методам программирования для суперЭВМ, анализу текстов на естественных языках, математической лингвистике, инструментальным средствам, обеспечивающим работу систем управления в реальном времени (СРВ), обработке изображений и некоторым вопросам теоретического программирования.

В нем также публикуются статьи и заметки общего характера, касающиеся разделов информатики, тесно связанных с программированием.

Раздел I
Общие вопросы программирования и информатики

Рябов Г. Г., Серов В. А.

Стандартная кубическая решетка R_c^n и биективное (генетическое) кодирование в ней

Аннотация

Стандартная n-мерная кубическая решетка R_c^n и ее топологические и метрические свойства, начиная с 80-ых годов прошлого века [1], продолжают оставаться актуальными и по сей день, особенно в свете значительного интереса к решению комбинаторных задач, связанных с разбиениями пространств. Несмотря на впечатляющие успехи в производительности суперкомпьютеров, проблема распараллеливаемости вычислений при решении комбинаторных задач стимулирует поиск новых представлений многомерных объектов, которые бы способствовали такому параллелизму.

В статье развиваются методы представления структур в стандартной кубической решетке R_c^n [1,2] в виде биективного кодирования на конечном алфавите [6]. В конечном итоге эти методы направлены на эффективные компьютерные реализации при хранении и вычислении топологических, метрических и комбинаторных характеристик таких структур для больших размерностей n.

Расширяется метрика Хаусдорфа-Хемминга, введенная в [7] для k-граней на n-кубе, до метрики Громова-Хаусдорфа между «кубическими» метрическими пространствами. Рассматриваются симплициальные разбиения в n-кубе, их биективное кодирование и эргодические свойства. Комбинаторное наполнение при разбиениях на R_c^n, и связанные с ним численные характеристики рассматриваются по отношению к возможностям суперкомпьютеров.

Ключевые слова: n-куб, решетка R_c^n, биективное кодирование, метрика Громова-Хаусдорфа, симплициальные разбиения, комбинаторное наполнение.

1. Исходные положения

Рассматривается биективное кодирование для конструктивного мира [3] кубических структур в стандартной кубической решетке R_c^n (на заданном в $\mathbf{R}^n$ реперe $B=(0,\mathbf{e}_1,\mathbf{e}_2,\ldots\mathbf{e}_n)$ и вершинами в целых точках $\mathbf{Z}^n$) и

состоящей из n-кубов, примыкающих друг к другу n-1-мерными гипергранями [2]. Такое кодирование каждой k-мерной грани (k-грани в n-кубе) ставит во взаимно однозначное соответствие n-разрядное троичное слово $D=<d_1,d_2,\ldots d_n>$, где d_i из алфавита $A=\{0,1,2\}$; Поскольку каждую k-грань можно представить как декартово произведение(П) единичных отрезков, каждый $\mathbf{I}(\mathbf{e}_i)$ из которых привязан к $\mathbf{e}_i \in B_1 \subset B$, и трансляции(Т) вдоль остальных $\mathbf{e}_j \in B_2 \subset B$ ($B_2=B \backslash B_1$), то свойство биективности для грани $f_{nk}(B_1,B_2)$ можно записать:

$$f_{nk}(B_1,B_2)=\prod_{k}\mathbf{I}(\mathbf{e}_i) + \overset{[1:1]}{\underset{n-k}{T}}(\mathbf{e}_j);\leftarrow\!\!\rightarrow<d_1,\ldots d_n>,$$

где $d_i=2$ для $\mathbf{e}_i \in B_1$, и $d_j=0,1$для $\mathbf{e}_j \in B_2$ При этом $d_j=0$, когда нет трансляции по $\mathbf{e}_j$ и $d_j=1$, когда есть.

При таком представлении множество всех n-разрядных троичных слов $A^*n=\{<d_1,\ldots d_n>\};d_i \in A=\{0,1,2\}$ можно считать множеством всех *генетических кодов* для всех k-граней n-куба над действиями декартова произведения (П) и трансляции (Т). Для краткости такие слова названы *кубантами* [6]. На множестве кубантов вместе с пополнением алфавита символом Ø пустого множества до $A'=\{Ø;0;1;2\}$,т.е. на A'^*_n (для всех n-разрядных четверичных кодов) введена поразрядная операция умножения(×), которая задается следующими правилами:

$$0\times0=0;\ 0\times1=1\times0=Ø;\ 0\times2=2\times0=0;\ 1\times1=1;\ 1\times2=2\times1=1;\ 2\times2=2;\ Ø\times0=Ø;$$
$$Ø\times1=Ø;\ Øx2=Ø;\ (1)$$

По существу это теоретико-множественное пересечение для трех множеств: 0;1-точки-концы единичного отрезка, 2-множество всех точек **I**. С введением этой операции A'^*_n становится *моноидом* с единицей–кубантом <22…2>(n-грань в n-кубе, т.е сам n-куб).

2. Метрики Хаусдорфа-Хемминга и Громова-Хаусдорфа

В [7-9] приведен ряд поразрядных операций над кубантами, их свойств и примеров построений комплексов и многообразий с использованием вычислений на кубантах. Рассмотрим два свойства кубантов, существенные для дальнейшего изложения:

1. Свойство умножения (1) – число символов Ø в произведении двух кубантов D_1 и D_2 равно длине минимального пути по ребрам n-куба между биективными гранями:

$$\#(Ø)(D_1\times D_2)=Lmin(D_1;D_2);\ (2)$$

2. Алгоритм вычисления кубанта D_1^*/D_2 для максимально удаленной грани (D_1) от грани с кубантом D_2, состоящий в рассмотрении всех пар $d_{1i} \in D_1$ и $d_{2i} \in D_2$ и во всех случаях, когда ($d_{1i}=2;d_{2i}=0$) замена $d_{1i}=1$ или ($d_{1i}=2;d_{2i}=1$) замена $d_{1i}=0$ (при остальных

комбинациях пар без изменений в разрядах D_1). С введенными изменениями D_1 обозначается как D_1*/D_2. Аналогично вычисление D_2*/D_1. Учитывая (2), можно записать:

$$Max_{D1\to D2}\{Lmin(D_1*/D_2;D_2)\}=\#(\emptyset)((D_1*/D_2)\times D_2); \quad (3)$$

$$Max_{D2\to D1}\{Lmin(D_2*/D_1;D_1)\}=\#(\emptyset)((D_2*/D_1)\times D_1); \quad (4)$$

Исходя из определения метрики Хаусдорфа и учитывая компактность k-граней на основании (3) и (4), можно положить, что для граней с кубантами D_1 и D_2:

$$\rho_{HH}(D_1;D_2)=Max\{\#(\emptyset)((D_1*/D_2\times D_2); \#(\emptyset)((D_2*/D_1)\times D_1)\}; \quad (5)$$

Таким образом все k-грани (k=0,…n) n-куба образуют как точки *конечное метрическое пространство Хаусдорфа-Хемминга* (метрика Хемминга- частный случай, когда кубанты двоичные слова, биективные вершинам n-куба)[7].

Отсюда можно рассматривать множество соответствующих конечных метрических пространств $\{M(\mathbf{I}),M(\mathbf{I}^2),\dots M(\mathbf{I}^n),\dots\}$, для которых в соответствии с теорией Громова методы вычисления расстояний между метрическими пространствами (ρ_{GH}-метрика Громова-Хаусдорфа) естественно переносится на этот случай. Поскольку в основе лежат методы изометрического вложения сопоставляемых метрических пространств M1 и M2 в пространство M3, где и вычисляется расстояние, для нашего случая в качестве M3 всегда для $M(\mathbf{I}^k)$ и $M(\mathbf{I}^n)$ может выступать $M(\mathbf{I}^n)$ при $n\geq k$. В этом случае изометрическое вложение для $M(\mathbf{I}^k)$ в $M(\mathbf{I}^n)$ это k-грань в n-кубе, для которой кубант равен D_1=<22…200…0> и #(2)D_1=k; #(0)D_1=n-k; Для $M(\mathbf{I}^n)$ кубант D_2=<22…2>. Формально вычисляя $\rho_{HH}(D_1,D_2)$ согласно (5), получаем:

$$\rho_{HH}(D_1;D_2)=\rho_{GH}(M(\mathbf{I}^k);M(\mathbf{I}^n))=|n-k|;$$

Отюда можно говорить для этого случая о сквозной *метрике Громова-Хаусдорфа-Хемминга* (GHH-метрика), вычисляемой на кубантах.

3. Симплициальные разбиения в кубических структурах

Симплициальные k-комплексы и их вложения в $\mathbf{R}^n$ продолжают оставаться в центре внимания ведущих математиков. В данном разделе рассматриваются представления симплициальных разбиений в кубических структурах R_c^n. За основу возьмем каноническую триангуляцию n-куба или разбиение n-куба на n-симплексы [Б], где каждый симплекс задается на основании обхода по n+1 вершине куба, начиная с вершины (00…0), и затем последовательно переходя в вершину с добавлением «1» в один из оставшихся с «0» разрядов (пусть с номером i), т.е. прохождение по ребру, параллельному $\mathbf{e}_i$. Процесс заканчивается всегда в вершине (11…1). Общее число таких различных

путей между (00…0) и (11…1) равно n! (как и всех перестановок множества индексов векторов из B) и каждому из них однозначно соответствует n-симплекс.

Множества вершин V и ребер E симплекса при заданном порядке обхода ($\mathbf{e_{i1},e_{i2},\ldots e_{in}}$) вычисляется следующим образом: $V=\{v_0=(00\ldots0); v_i=v_{i-1}+\mathbf{e_{is}};\ s=1\div n\}$; $E=\{v_0v_1;v_0v_2;v_0v_3;v_1v_3;v_2v_3;\ldots v_{n-1}v_n\}$; V и E образуют 1-мерный остов симплекса.

Для канонического разбиения конкретной k-грани в n-кубе (кубант D) порядок обхода задается на множестве $B_1=\{\mathbf{e_{is}}:d_{is}\in D;d_{is}=2;\}=\{\mathbf{e_{j1},\ldots e_{jk}}\}$ подстановкой P: $P(\mathbf{e_{j1},\ldots e_{jk}})=(\mathbf{e_{m1},\ldots e_{mk}})$;

Далее следуют действия для образования множеств V и E, аналогичные случаю n-куба. Таким образом обозначая действие P ($P\in S_k$, где S_k-симметрическая группа подстановок) на D по образованию V и E, как Θ, а симплекс с построенным 1-остовом - через Δ, можно записать:

$$[1:1]$$
$$\Theta(D,P)=(V,E)\leftarrow\rightarrow\Delta_0(D,P);$$
$$\Delta(D,P)=\Delta_0(D,P)+T(\mathbf{e_{it}}),\ \text{где } \mathbf{e_{it}}\in B_2;B_2=B\backslash B_1;$$

По аналогии с кубантом пару (кубант, подстановка) и обозначаемую в виде W=<D/P>= <d1,d2,…dn/m1,…mk> будем называть *симпантом*. Алфавит для симпантов расширен до всех цифр 0,1,…9 (не считая вспомогательных символов). Графическая интерпретация симплексов в $\mathbf{I^9}$ с симпантами <002121102/9,3,5> и <222222222/7,6,4,9,3,5,1,2,8> показана на рис.1.

Множество всех симплексов при симплициальных разбиениях всех k-граней n-куба (или биективных им симпантов) можно рассматривать как множество точек конечного метрического НН-пространства по аналогии со случаем кубантов. Однако алгоритм вычисления НН-расстояний в этом случае будет иным. Прежде всего отметим, что 1-остов кубических к-граней состоит только из единичных отрезков, которые обуславливают применение метрики $l1$. В случае симплициальных разбиений 1-остов состоит из единичных отрезков и всевозможных диагоналей в k-кубах (k=1,…n) с евклидовыми длинами $\sqrt{k}$, т.е.: $1,\sqrt{2},\sqrt{3},2,\sqrt{5},\sqrt{6},\sqrt{7},2\sqrt{2},3,\sqrt{10},\ldots$ Таким образом для симплексов (соответственно симпантов) можно рассмотреть метрику Хаусдорфа в n-кубе на множестве кратчайших путей по расширенному множеству ребер, евклидовые длины, которых приведены выше.

Пусть в n-кубе заданы два симплекса Δ1 и Δ2, которым соответствуют симпанты W1=<D1/P1> и W2=<D2/P2>, где D1 и D2-кубанты разбиваемых на симплексы k1 и k2- граней, а P1 и P2 - подстановки, задающие порядок обхода по вершинам граней в соответствии с вышеизложенным построением симплексов.

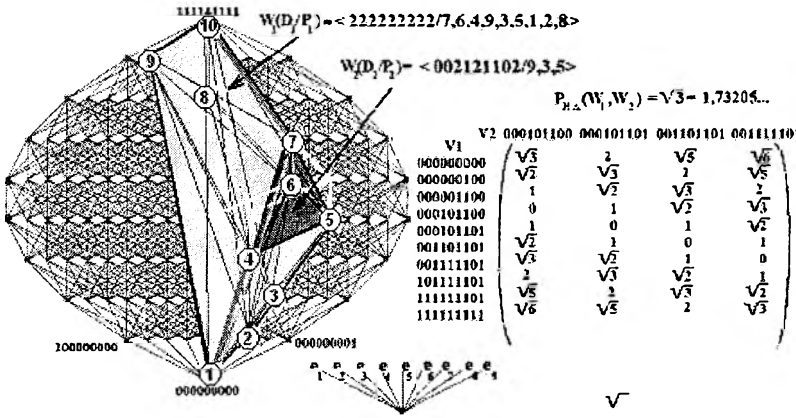


Рис. 1. На плоской проекции одномерного остова (вершины и ребра) I^9 показаны одномерные остовы 3-симплекса, биективного симпанту<002121102/9,3,5> (темносерый цвет) и 9-симплекса, биективного симпанту <222222222/7,6,4,9,3,5,1,2,8>(светлосерый цвет). На ребрах 1-4 и 7-10 реализуется $\rho_{H\Delta}(W1,W2)$

Соответствующие множества вершин (0-остовы) пусть будут V_1 и V_2 с числом вершин k_1+1 и k_2+1. Построим матрицу размерности $(k_1+1)\times(k_2+1)$, каждая строка которой соответствует одной из вершин V_1 и каждая строка одной из вершин V_2 так, что элемент на их пересечении равен длине одного ребра (т.е. кратчайшего отрезка) между этими вершинами из множества единичных и диагональных ребер во всех k-гранях n-куба. Для каждой строки выбирается минимальный элемент (т.е. для вершины из V_1) и затем берется максимум по этим минимальным элементам M_1. Аналогично вычисляется для столбцов M_2. $\max\{M1,M2\}=\rho_{H\Delta}(W1,W2)$. Численный пример приведен рядом с рис.1.

4. Комбинаторное наполнение

Итак каждой кубической k-грани в n-кубе соответствует k! симпантов, биективных k! симплексам и общее их число в n-кубе:

$$F_\Delta(\mathbf{I}^n)=\sum_{k=2}^{n} k!C_n^k\,2^{n-k}; \qquad (6)$$

Поскольку $k!C_n^k=n!/(n-k)!$ и подставляя в (6), получаем:

$$F\Delta(\mathbf{I}^n)=n!\Sigma 2^{n-k}/(n-k)!\approx n!e^2;$$

Отсюда: $\lim_{n\to\infty} F_\Delta(\mathbf{I}^n)/n!=e^2$; (7)

Это *эргодическое свойство* можно сформулировать так.

Предел числа всех симплексов при канонических симплициальных разбиениях всех k-граней n-куба, деленной на n! при $n \to \infty$ *конечен и равен* e^2.

Здесь трудно удержаться от некоторой метафоры. Представим n-куб при больших n, как большой арбуз, который по заданному способу разрезается на большие доли (сродни n-симплексам). А все семечки внутри арбуза будем считать как симплексы всех граней. Так что после разрезания на большие доли и высыпания всех внутренних семечек у нас будут две кучки - больших долей и всех семечек. Каким бы большим арбуз не был ($n \to \infty$) число семечек будет всегда только в e^2 (т.е. примерно в семь с небольшим) раз больше чем число больших долей.

Отметим, что для кубических граней всех размерностей в n-кубе:

$$F_{\square}(\mathbf{I}^n)=3^n;\ F_{\square}(\mathbf{I}^n)/3^n \equiv 1; (8)$$

В этой связи эргодическое свойство (7) можно трактовать как конечность *нормированного комбинаторного наполнения* (деленного на n!, т.е. на число n-симплексов в n-кубе) для симплициальных разбиений (канонической триангуляции всех k-граней) при $n \to \infty$ и равного e^2.

Прологарифмировав (7) и (8) и учитывая $\Sigma \ln x \approx \int \ln x dx = x\ln x - x$; для больших n, получаем:

$$\ln F_{\square}(\mathbf{I}^n)=3\ln n;\quad \ln F_{\Delta}(\mathbf{I}^n) \approx n \ln n - n + 3;\ \text{(для больших n)}.$$

Можно назвать эту величину *степенью комбинаторного наполнения* для заданного способа разбиения. Далее распространим этот подход для случая введенных на R_c^n кубических окрестностей Q_r^n радиуса r ($r \in \mathbf{N}$) относительно (0,0,...0), состоящих из всех n-кубов с координатами вершин по модулю $\le r$.

Так для Q_1^n показано, что число k-граней в таком комплексе:

$$F_{nk}(Q_r^n)=C^k_n\,(2r)^k(2r+1)^{n-k};\ [9]$$

Для мнемоники этого соотношения: 2r букв алфавита идет для отображения отрезков-сомножителей декартовых произведений и 2r+1 буква для трансляций (точки-концы отрезков) при формировании слов (кубантов), биективных k-граням в Q_r^n. Отсюда:

$$F_{\square}(Q_r^n)=(4r+1)^n;$$

$$F_{\Delta}(Q_r^n) \approx n!(2r)^n e^{2r+1};\ \text{для больших n};$$

Обозначив степень комбинаторного наполнения кубическими k-гранями для Q_r^n через $\Phi_1(n,r)=\ln F_{\square}(Q_r^n)$, и k-симплексами через $\Phi_2(n,r)=\ln F_{\Delta}(Q_r^n)$ получаем:

$$\Phi_1(n,r)=n\ln(4r+1);$$

$$\Phi_2(n,r) \approx n\ln n - n + n\ln(2r)+3;\ \text{для больших n};$$

Графически поверхности Φ_1 и Φ_2 представлены на рис.2б.

5. Дискуссия

В случае попытки конструктивного решения задач на многомерных разбиениях (сетках) даже с применением самых мощных суперкомпьютерных систем возникает естественный вопрос о мере комбинаторного наполнения при выборе r и n для $Q_r^n \subset R_c^n$ и выбора кубического или симплициального разбиений. Грубо говоря, возникает вопрос, как развивать конструктивную решетку (с генетическим-биективным кодированием всех кубических или симплициальных k-граней), увеличивая радиус r окрестности, или увеличивая размерность n решетки? Конечно, ответ на этот вопрос прежде всего зависит от существа решаемой задачи (например для методов «частицы в ячейке»), но в случае возможности вариации n и r и на основании вышеизложенного на первый взгляд напрашивается несколько *парадоксальный вывод*. При конструктивном подходе «легче» (т.е. без больших скачков в степени комбинаторного наполнения) продвигаться в сторону увеличения размерности n, чем увеличения радиуса r. Формально можно принять логарифм числа узлов или симплексов для сеточных методов больших задач (10^9-10^{12}) в размерности n=3, как некоторую оценку возможностей современных суперкомпьютеров. Она будет численно близка степени комбинаторного наполнения для разбиений со значениями (n,r) для чисел из таблицы, помеченных вертикальной жирной линией.(рис.2а) Отсюда видно, что размеры достижимых (по комбинаторному наполнению) (n,r)-областей для современных суперкомпьютеров еще достаточно скромны. С другой стороны *поразрядный характер операции умножения* над генетическими кодовыми словами, которая лежит в основе всех метрических вычислений и вычислений по связности, при стринговой организации памяти компьютера обеспечивает максимальное распараллеливание вычислений для структур в решетке R_c^n при n >>3.

кубическое разбиение

r \ n	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	5.493	6.592	7.690	8.789	9.888	10.986	12.085	13.183	14.282	15.381	16.479	17.578	18.676	19.775	20.874	21.972
1	8.047	9.657	11.266	12.876	14.485	16.094	17.704	19.313	20.923	22.532	24.142	25.751	27.360	28.970	30.579	32.189
2	10.986	13.183	15.381	17.578	19.775	21.972	24.169	26.367	28.564	30.761	32.958	35.156	37.353	39.550	41.747	43.944
3	12.825	15.390	17.955	20.520	23.085	25.649	28.214	30.779	33.344	35.909	38.474	41.039	43.604	46.169	48.734	51.299
4	14.196	16.999	19.832	22.665	25.499	28.332	31.165	33.999	36.832	39.665	42.498	45.331	48.165	50.998	53.831	56.664
5	15.223	18.267	21.312	24.356	27.401	30.445	33.490	36.534	39.579	42.623	45.668	48.712	51.757	54.801	57.846	60.890
6	16.034	19.313	22.532	25.751	28.970	32.189	35.408	38.627	41.846	45.064	48.283	51.502	54.721	57.940	61.159	64.378
7	16.836	20.204	23.571	26.938	30.306	33.673	37.040	40.406	43.775	47.142	50.509	53.877	57.244	60.611	63.979	67.346
8	17.483	20.979	24.476	27.972	31.469	34.966	38.462	41.958	45.455	48.951	52.448	55.944	59.441	62.937	66.434	69.930
9	18.056	21.666	25.276	28.887	32.498	36.109	39.720	43.331	46.942	50.553	54.164	57.775	61.386	64.997	68.607	72.218
10	18.568	22.281	25.995	29.709	33.422	37.136	40.849	44.563	48.276	51.990	55.704	59.417	63.131	66.844	70.558	74.271
11	19.033	22.840	26.647	30.453	34.260	38.067	41.873	45.680	49.487	53.293	57.100	60.907	64.713	68.520	72.327	76.133
12	19.459	23.351	27.243	31.135	35.026	38.918	42.810	46.702	50.594	54.486	58.377	62.269	66.161	70.053	73.945	77.836
13	19.851	23.822	27.792	31.762	35.733	39.703	43.673	47.644	51.614	55.584	59.554	63.525	67.496	71.465	75.436	79.406
14	20.215	24.258	28.301	32.344	36.387	40.431	44.474	48.517	52.560	56.603	60.646	64.689	68.732	72.775	76.818	80.861
15	20.554	24.665	28.776	32.887	36.998	41.109	45.220	49.330	53.441	57.552	61.663	65.774	69.885	73.996	78.107	82.217

симплициальное разбиение

0	6.947	7.751	9.621	11.636	13.775	16.026	18.377	20.819	23.344	25.947	28.621	31.361	34.166	37.027	39.944	42.915
1	9.913	11.935	14.473	17.181	20.013	22.957	26.001	29.137	32.365	35.681	39.078	42.452	45.948	49.503	53.114	56.778
2	12.979	16.068	19.325	22.726	26.252	29.889	33.626	37.454	41.366	45.355	49.415	53.542	57.732	61.980	66.284	70.641
3	16.006	19.201	22.164	25.970	29.901	33.943	38.086	42.320	46.637	51.031	55.497	60.030	64.625	69.278	73.988	78.750
4	19.044	23.227	24.177	29.271	32.490	36.820	41.251	45.772	50.377	55.059	59.812	64.632	69.515	74.457	79.454	84.503
5	22.560	26.596	29.739	35.055	34.496	39.052	43.705	48.456	53.278	58.103	63.180	68.203	73.356	78.473	83.650	88.886
6	25.472	30.660	34.016	31.515	36.139	40.875	45.711	50.635	55.648	60.736	65.894	71.120	76.408	81.755	87.158	92.613
7	29.242	33.586	38.095	32.748	37.527	42.416	47.406	52.486	57.662	62.894	68.207	73.585	79.029	84.530	90.085	95.695
8	33.010	34.386	38.029	33.816	38.728	43.752	48.875	54.090	59.388	64.763	70.210	75.723	81.299	86.933	92.624	98.366
9	36.499	35.093	39.854	34.759	39.788	44.930	50.171	55.593	60.919	66.412	71.976	77.607	83.301	89.053	94.861	100.722
10	21.026	35.725	30.591	36.601	40.737	45.983	51.330	56.768	62.289	67.897	73.557	79.293	85.092	90.950	96.863	102.829
11	21.522	36.297	31.299	36.364	41.594	46.936	52.378	57.911	63.528	69.221	74.986	80.818	86.712	92.665	98.674	104.735
12	21.937	36.819	31.956	37.060	42.376	47.806	53.335	58.955	64.659	70.440	76.292	82.210	88.192	94.232	100.327	106.476
13	22.338	37.299	32.426	37.760	43.098	48.607	54.216	59.916	65.700	71.560	77.492	83.491	89.552	95.672	101.848	108.077
14	22.738	37.744	32.947	38.293	43.766	49.348	55.031	60.805	66.663	72.598	78.604	84.677	90.812	97.006	103.255	109.559
15	23.053	38.165	33.430	38.845	44.386	50.038	55.790	61.633	67.560	73.564	79.639	85.781	91.985	98.240	104.547	110.899

Рис. 2. а). Таблицы со значениями степени комбинаторного наполнения для кубического и симплициального разбиений; помечены примерные границы возможностей обработки объектов (размерности «n» и радиуса «r») для современных суперкомпьютеров

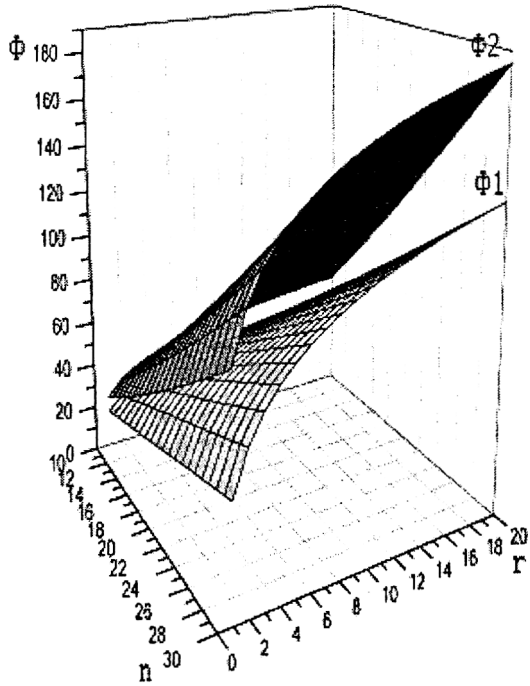


Рис. 2. б). Поведение Ф1 и Ф2

6. Об алфавите для Q_r^n при больших n и r

При больших r может возникнуть вопрос о буквенном (символьном) наполнении алфавита для биективного кодирования, поскольку букв в таком алфавите должно быть 6r+1. А именно 2r букв для отрезков, 2r+1-для точек(концов отрезков) и 2r для расстояний между точками и отрезками. Поскольку вышеизложенное в основном касалось одного n-куба, естественным было под троичным кодированием подразумевать алфавит {0,1,2}, а под четверичным {Ø,0,1,2}. Однако с таким же успехом могли быть использованы другие буквы, для которых таблица поразрядного умножения была бы перекодирована простой заменой букв одного алфавита на буквы другого алфавита, вероятно потеряв наглядность при объяснении самого метода кодирования (что для компьютера безразлично).

Для случая Q_r^n необходим более универсальный способ кодирования. В качестве советчика по кодированию призовем здесь методы кодирования ДНК (биологической природы), когда аминокислоты кодируются 3-связками (кодонами) из четырех букв, однозначно соответствующих нуклеотидам. [6]

Пусть наш алфавит состоит из составных букв вида i+целое со знаком для отрезков, p+целое со знаком для точек, d+целое(всегда положительное) для расстояний и просто целое- для номеров базисных векторов. Пусть все целые представлены в десятичной системе счисления с числом q десятичных разрядов таким, что $q=\max\{\lceil \lg n \rceil;\lceil \lg r \rceil\}$, где lg-десятичный логарифм. Дополнительные символы-разделители: < > начало и конец одного кубанта или симпанта, /разделитель в записи симпанта между буквами кубанта и числами, определяющими порядок симплициального обхода, символ «,» для разделения букв в слове.

Так для примера вид некоторого симпанта для 4-симплекса в Q_7^6:

$$W= <i-7,i+5,p+6,p-3,i-4,i+4/5,1,2,6>;$$

В частности для симплексов рис.1 симпанты в предложенной кодировке будут иметь следующий вид:

W1=<002121102/9,3,5>→<p0,p0,i+1,p+1,i+1,p+1,p+1,p0,i+1/9,3,5>;

W2=<222222222/7,6.4,9,3,5,1,2,8>→<i+1,i+1,…i+1/7,6,4,9,3,5,1,2,8>;

Такая кодировка остается инвариантной для этих симплексов при сколь угодно большом r в Q_r^n.

6. Заключение

Вышеизложенное есть краткое описание части математического обеспечения инструментального комплекса под общим названием «Топологический процессор», программное обеспечение которого развивается на суперкомпьютере МГУ «Чебышев» с 2008 года. Авторы выражают глубокую благодарность В.А.Садовничему, В.П.Маслову, Л.Н.Королеву, А.В.Тихонравову, А.Н.Томилину за поддержку работ по тематике биективного кодирования для супервычислений.

Литература

1. С.П.Новиков. Топология. Москва-Ижевск. РХД. 2002.
2. Долбилин Н.П., Штанько М.А., Штрогин М.И. Кубические многообразия в решетках. // Изв. РАН. Сер.матем. 1994. 58. вып.2. 93-107.
3. Бухштабер В.М., Панов Т.Е. Торические действия в топологии и комбинаторике. МЦНМО. 2004.
4. Yu.I. Manin. Classical computing, quantum computing and Shor's factoring algorithm.//arXiv:quant-ph/9903008v1. 2 Mar 1999.
5. M.Gromov, L.Guth. Generalizations of the Kolmogorov-Barzdin embedding estimates.//arXiv:1103.3423v1 [math.GT] 17 Mar 2011.
6. F.H.Crick, L.Barnett, S.Brenner, R.J.Watts-Tobin. General nature of the genetic code for proteins.//Nature.1961.(192), 1227-1232.
7. Г.Г.Рябов. О четверичном кодировании кубических структур.//Вычислительные методы и программирование. 2009. т.10, 340-347.
8. Хаусдорфова метрика на гранях n-мерного куба.//Фундамент. и прикл. Матем., 16:1, (2010), 151-155.
9. Г.Г.Рябов, В.А.Серов. О метрико-топологических вычислениях в конструктивном мире кубических структур.//Вычислительные методы и программирование. 2010 т.11, 326-335.

Баранов М. И.

Алгоритмы локального поиска для задач удовлетворения ограничений

Введение

Решение оптимизационных и комбинаторных задач всегда было актуальной проблемой. Достаточно рассмотреть такие задачи планирования, как задача распределения работ на многопроцессорной системе или задача календарного планирования. Проблемы, возникающие при решении подобных задач, связаны с тем, что они являются NP-трудными, то есть для них не придумано алгоритмов, которые позволяли бы решать эти задачи за приемлемое (полиномиальное) время.

Для решения таких задач в рамках дисциплины искусственный интеллект было предложено несколько подходов, называемых метаэвристиками. Метаэвристические методы поиска решений можно определить как шаблоны алгоритмов, которые могут быть использованы как направляющие стратегии для поиска решения в пространстве состояний [1]. К метаэвристикам относят такие классы алгоритмов, как генетические алгоритмы, муравьиные алгоритмы, алгоритмы локального поиска и другие.

Семейство алгоритмов локального поиска существенно отличается от алгоритмов полного перебора тем, что они не производят систематический перебор всего пространства поиска и за счёт этого, обычно, имеют значительно меньшее время работы. В алгоритмах локального поиска процесс отыскания решения ведется только на основе текущего состояния, а ранее пройденные состояния не учитываются и не запоминаются [2]. Важное преимущество этого класса алгоритмов состоит в том, что занимаемая во время работы область памяти имеет полиномиальный порядок роста (чаще всего линейный) в зависимости от размерности задачи. Однако эти алгоритмы обладают существенным недостатком: в общем случае нет гарантий того, что работа алгоритма завершится успешным нахождением решения. Очень часто такие алгоритмы, попадая в локальный минимум, никак не могут его преодолеть, помогает только перезапуск алгоритма с другими начальными значениями параметров.

Несмотря на то, что эвристика локального поиска успешно используется на практике для решения отдельных задач, вопрос применимости локального поиска к целым классам задач остаётся открытым. В данной работе рассматривается применение эвристики локального поиска к задачам удовлетворения ограничений.

Задачи удовлетворения ограничений

Согласно [3], задачи удовлетворения ограничений – это задачи, определяемые набором объектов, чьё состояние должно удовлетворять ряду ограничений. Такие задачи являются предметом интенсивных исследований в областях искусственного интеллекта и исследования операций, так как большое количество задач, часто неродственных между собой, можно сформулировать в терминах задач удовлетворения ограничений. В качестве примеров можно привести задачу выполнимости булевого выражения – самую первую известную NP-полную задачу, задачу раскраски карты, задачу решения судоку и многие другие. Обычно задачи удовлетворения ограничений обладают высокой вычислительной сложностью и требуют совмещения алгоритмов комбинаторного поиска и различных эвристик.

Задачу удовлетворения ограничений можно определить формально как тройку $<X, S, C>$, где X – это набор переменных $(x_1, ..., x_n)$, S – область определения переменных, C – набор ограничений на переменные. Каждое ограничение задаётся n-арным отношением R над S. Состояние s – это набор значений переменных из области определения $s: X \rightarrow S$. Состояние s удовлетворяет ограничению R, если $(s(x_1), ..., s(x_n)) \in R$. Решением задачи является любое состояние, которое удовлетворяет всем ограничениям из C. Если состояние s не является решением задачи, оно не удовлетворяет некоторым ограничениям из C. Ограничения, которым не удовлетворяет s, будем называть коллизиями состояния s.

Стратегия локального поиска

Рассмотрим основные особенности алгоритмов локального поиска. Пусть P – задача с пространством состояний S. Необходимо найти в пространстве S состояние, являющееся решением задачи P. Для применения стратегии локального поиска должна быть задана функция N, которая для каждого возможного состояния $s \in S$ находит *соседние состояния* $N(s) \in S$. Элемент $s' \in N(s)$ называется *соседом* s. Также должна быть задана *оценочная функция* $f(s)$, которая определяет качество состояния s. Семантику и реализацию этих функций следует тщательно продумать, так как от них напрямую зависит скорость получения решения и сама возможность отыскания решения задачи.

Алгоритм локального поиска, стартуя из какого-то *начального состояния* s_0 (которое может быть получено с помощью отдельного алгоритма или сгенерировано случайным образом), итерационно в цикле исследует пространство поиска, проходя от текущего состояния к одному из его соседей, пока не найдёт состояния, являющегося решением задачи. Каждое передвижение от состояния к состоянию

называется *переходом*. Выбор состояния для очередного перехода осуществляется, исходя из оценочной функции *f(s)*.

Необходимо помнить, что алгоритм локального поиска может найти решение не из любого начального состояния. Например, при попадании в *локальный минимум* алгоритм не может найти решение. Учитывая эту особенность, нужно предусмотреть возврат алгоритма к генерации нового стартового состояния при попадании в локальный минимум или в ситуации, когда алгоритм слишком долго блуждает по пространству состояний без улучшения текущего состояния. Кроме того, нужно обеспечить элемент случайности в алгоритме генерации начального состояния. Иначе алгоритм локального поиска будет стартовать всегда из одного и того же состояния и, скорее всего, будет проходить один и тот же неплодотворный путь поиска.

Формулировка алгоритма локального поиска:

1. Генерация начального состояния $s := s_0$
2. Построение множества соседних состояний $N(s)$
3. Выбор очередного состояния s', учитывающий оценочную функцию $f(s)$ и стратегию выбора соседа
4. Если s' является решением, окончание работы
5. Осуществление перехода $s := s'$
6. Принятие решения о целесообразности дальнейшего поиска из текущего состояния. Если дальнейший поиск целесообразен, возврат на шаг 2, иначе возврат на шаг 1.

Задачи удовлетворения ограничениям являются потенциально перспективными для алгоритмов локального поиска в силу того, что они формулируются в терминах состояний. К тому же наличие множества ограничений позволяет определить оценочную функцию естественным образом – как количество нарушаемых ограничений. Далее рассмотрим построение алгоритмов локального поиска для двух задач.

Задача об n ферзях

Рассмотрим применение эвристики локального поиска к задаче об n ферзях. Задача об n ферзях заключается в том, чтобы расставить n ферзей на шахматной доске размером $n \times n$ так, чтобы ни один из них не находился под боем другого. Ферзи находятся под боем друг друга, если они стоят на одной горизонтали, вертикали или диагонали.

Сформулируем задачу об n ферзях в терминах задач удовлетворения ограничений. Состоянием будет являться расстановка n ферзей на шахматной доске, набором переменных – позиции n ферзей на доске. Тогда набор ограничений можно сформулировать следующим образом: на каждой вертикали, горизонтали и диагонали доски расположено не более одного ферзя.

Введём оценочную функцию $f(s)$ как количество коллизий на доске. В этом случае решением задачи будет являться расстановка ферзей s такая, что $f(s) = 0$.

Между ферзями могут быть два типа коллизий:

1. Коллизии по горизонталям и вертикалям: на одной горизонтали или вертикали стоит более одного ферзя.
2. Коллизии по диагоналям: на одной диагонали стоит более одного ферзя.

Из условия задачи следует, что в состоянии, которое является решением задачи, на каждой горизонтали доски может находиться только один ферзь. В процессе поиска решения будем рассматривать только состояния, удовлетворяющие этому условию. При этом состояние однозначно задаётся номерами вертикалей, на которых расположены ферзи. Основной структурой данных будет вектор Q_n: элемент q_i хранит номер вертикали, в которой стоит ферзь в i-ой горизонтали.

В качестве перехода будем использовать следующую перестановку ферзей. Пусть для перестановки выбраны ферзи на i-ой и j-ой горизонталях. Перемещаем ферзя в i-ой горизонтали на q_j-ую вертикаль, а ферзя в j-ой горизонтали – на q_i-ую вертикаль. Заметим, что для удачной перестановки хотя бы один из ферзей должен участвовать в коллизиях, причём выбранные ферзи не должны конфликтовать друг с другом. Применение операции перехода изображено на рисунке 1.

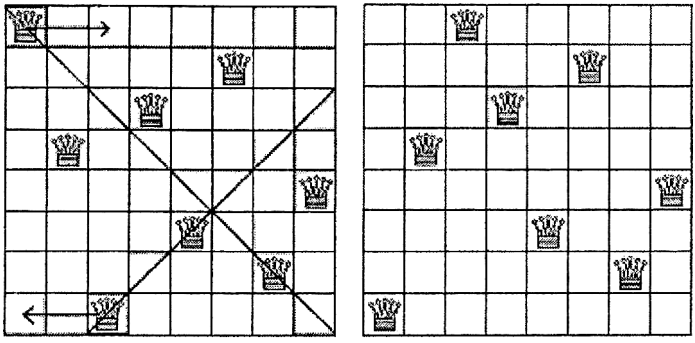


Рис. 1. Состояние доски до и после осуществления перехода

Наложением ограничений на начальное состояние s_0 легко добиться того, чтобы в расстановке отсутствовали коллизии первого типа. Для этого достаточно обеспечить выполнение условия: числа q_1, q_2, ..., q_n должны образовывать перестановку чисел $1..n$. Также заметим, что введённая выше операция перехода не может привести к появлению

коллизий первого типа, поскольку не меняет количество ферзей на горизонталях и вертикалях. Таким образом, поиск решения будет осуществляться в подмножестве состояний, удовлетворяющих ограничениям на горизонтали и вертикали.

Для отслеживания коллизий второго типа будем использовать вектора $P_{(2n-1)}$ и $N_{(2n-1)}$. Элемент p_i хранит количество ферзей на i-ой диагонали положительного уклона, элемент n_i хранит количество ферзей на i-ой диагонали отрицательного уклона. Нужно отметить, что значения в этих векторах должны обновляться при каждом осуществлённом переходе. По заданной позиции (i, j) на доске $n \times n$ можно легко определить номер соответствующих ей диагоналей. Номер диагонали положительного уклона $I_p = i - j + n$, номер диагонали отрицательного уклона $I_n = i + j - 1$.

Число коллизий на диагонали, на которой расположено p ферзей:

$$K(p) = \begin{cases} 0, & p = 0 \\ p - 1, & p \neq 0 \end{cases}$$

Общее число коллизий второго типа в расстановке равно сумме коллизий по всем диагоналям.

Отметим, что все используемые для решения задачи структуры данных (Q_n, $P_{(2n-1)}$ и $N_{(2n-1)}$) будут занимать область памяти линейного размера по отношению к размерности задачи.

Рассмотрим, как реализованный алгоритм локального поиска для задачи об n ферзях работает на практике. Для испытания выбирались различные значения n. Для каждого n было выполнено 10 запусков программы. В таблицах 1 и 2 приведены средние значения следующих величин: n – размерность задачи; m – количество попыток (количество генераций начальных расстановок); k – количество коллизий в начальной расстановке; s – количество переходов, которые потребовалось провести для достижения решения; *time* – время поиска одного решения.

n	32	512	2 000	128 000	512 000
m	14	2	1	1	1
k	5	21	63	985	3996
s	1321	2191	2359	63457	243837
time	0:00.00035	0:00.00030	0:00.00074	0:00.04581	0:00.18273

Табл. 1. Результаты работы программы для решения задачи об n ферзях при размерности задачи от 32 до 512000

n	1000000	4000000	16000000	64000000	100000000
m	1	1	1	1	1
k	7959	30367	121404	484143	764710
s	480485	1893174	7483937	30948274	47746173
time	0:00.89516	0:03.84917	0:18.36939	1:21.78338	2:15.06694

Табл. 2. Результаты работы программы для решения задачи об n ферзях при размерности задачи от 1000000 до 100000000

Начиная с n = 2000, среднее количество генераций начальных расстановок для получения решения равно 1, то есть решение находилось, исходя из каждой начальной расстановки. Это говорит о том, что при увеличении размерности задачи алгоритму локального поиска становится проще устранять коллизии. Это можно объяснить тем, что при увеличении шахматной доски алгоритм получает всё большую свободу действий по перестановке ферзей.

Начиная с n = 1000000, количество шагов поиска и время работы алгоритма хорошо интерполируются линейными функциями от размерности задачи n.

Задача составления школьного расписания

Задача составления школьного расписания состоит в том, чтобы назначить уроки таким образом, чтобы ни один учитель и ни один класс не участвовали более чем в одном уроке в каждый момент времени. Желательно, чтобы у классов не было незанятых уроков в середине дня, а учителя могли выбирать время, когда им удобнее вести занятия.

Формализуем данную задачу. Пусть есть m классов $c_1, \ldots, c_m$, n учителей $t_1, \ldots, t_n$ и p периодов $1, \ldots, p$ – уроки в течение недели. Пусть задана целочисленная матрица $R_{m\times n}$ неотрицательных чисел, называемая *матрицей требований*. Элемент r_{ij} задаёт количество уроков, которое учитель t_j должен провести у класса c_i. По существу, матрица R является учебным планом. Занятость учителей задаётся *матрицей предпочтения учителей* $T_{n\times p}$: элемент t_{jk} = 1, если учитель t_j может вести занятие в период k, и t_{jk} = 0 в противном случае.

Математическая формулировка задачи выглядит так:

Найти x_{ijk} = 0 или 1 (i = 1..m; j = 1..n; k = 1..p) такие, что

$$\sum_{k=1}^{p} x_{ijk} = r_{ij} \qquad (i = 1..m; j = 1..n) \qquad (1)$$

$$\sum_{j=1}^{n} x_{ijk} \le 1 \qquad (i = 1..m;\ k = 1..p) \qquad (2)$$

$$\sum_{i=1}^{m} x_{ijk} \le t_{jk} \qquad (j = 1..n;\ k = 1..p) \qquad (3)$$

Майкл Гэри и Дэвид Джонсон показали, что задача в такой формулировке является NP-полной [4].

В реальных расписаниях нежелательны пустые уроки в середине дня; нагрузка класса в разные дни недели не должна сильно различаться. Информацию о необходимой нагрузке классов будем хранить в *матрице занятости классов* $D_{m \times p}$: элемент $d_{ik} = 1$, если у класса c_i должен быть назначен урок в период времени k, и $d_{ik} = 0$ в противном случае. Расписание должно удовлетворять неравенствам

$$\sum_{j=1}^{n} x_{ijk} \ge d_{ik} \qquad (i = 1..m;\ k = 1..p), \qquad (4)$$

Очевидно, сформулированная задача является задачей удовлетворения ограничений. Состоянием является расписание занятий на неделю, возможно, даже недопустимое. Состояние, удовлетворяющее всем ограничениям из (1) - (4), является решением задачи. Коллизиями для состояния будут являться уравнения или равенства, которым не удовлетворяет данное состояние.

Введём оценочную функцию *f(s)* как количество коллизий в расписании *s*. Задача составления школьного расписания в этом случае будет состоять в том, чтобы минимизировать оценочную функцию, решением задачи будет являться расписание *s* такое, что *f(s) = 0*.

Для решения задачи составления школьного расписания будем использовать два типа переходов:

1. *Атомарный переход* $\{c, t, p_1, p_2\}$ заключается в переносе у класса *c* занятия учителя *t* с периода p_1 на период p_2. Необходимым условием применимости такого перехода являстся назначенное занятие учителя *t* у класса *c* в период p_1.
2. *Двойной переход* $\{c, t_1, t_2, p_1, p_2\}$ заключается в обмене двух занятий у класса *c* местами: занятие учителя t_1 переносится с периода p_1 на период p_2, а занятие учителя t_2 – с p_2 на p_1. Условием применимости такого перехода является назначенные занятия у класса *c*: занятие учителя t_1 в период p_1, занятие учителя t_2 в период p_2.

Очевидно, что двойной переход $\{c, t_1, t_2, p_1, p_2\}$ эквивалентен последовательному применению двух атомарных переходов $\{c, t_1, p_1, p_2\}$ и $\{c, t_2, p_2, p_1\}$. Разница между двойным переходом и двумя атомарными заключается в том, что после первого атомарного перехода

количество коллизий в расписании зачастую увеличивается, и только второй переход может исправить ситуацию. Поэтому с точки зрения алгоритма локального поиска нельзя разбивать двойные переходы на два атомарных.

Задача составления школьного расписания накладывает жёсткие ограничения на свои решения. Разобьём ограничения на группы:

1. Коллизии требований учебного плана R. Каждой паре (класс, учитель) должно быть назначено определённое количество уроков в неделю.
2. Коллизии занятости классов. В каждый момент у класса должно быть не более одного урока, а в некоторые моменты какой-либо урок должен быть назначен обязательно (неравенства (2) и (4)).
3. Коллизии занятости учителей. В каждый момент времени у учителя должно быть не более одного урока (неравенства (3)).

Составить расписание, удовлетворяющее ограничениям первых двух групп, достаточно просто. Нужно для каждого класса составить список необходимых занятий, а затем расставить их в произвольном порядке за два прохода по расписанию. За первый проход расставить занятия только на периоды, обязательные к заполнению уроками, за второй – на оставшиеся периоды произвольным образом. Очевидно, что сформированное таким образом расписание, скорее всего, не будет удовлетворять ограничениям из третьей группы.

Далее будем предполагать, что мы работаем с множеством расписаний, которые могут содержать коллизии только по ограничениям из третьей группы. Двойные переходы, очевидно, не выводят из этого множества. На применение атомарного перехода $\{c_i, t, p_1, p_2\}$ нужно наложить дополнительное условие: урок p_1 не является обязательным для класса c_i, т.е. $d_{ip_1} \neq 1$.

Тогда расписание можно представить в виде матрицы $M_{n \times p}$. Элемент матрицы расписания m_{jk} представляет собой список занятий, назначенных учителю j в момент времени k. Элементом этого списка может быть номер класса или особое значение *. Если элементом списка m_{jk} является номер класса c, это означает, что по расписанию в период времени k учителю j назначено занятие у класса c. Если элементом списка m_{jk} является особое значение *, это означает, что по расписанию в период времени k учитель j занят внеучебной работой. Строка матрицы j задаёт расписание на неделю для учителя t_j. Понятно, что если в матрице расписания присутствуют элементы-списки с более чем одним элементом, то такое расписание является недопустимым, то есть содержит коллизии.

Количество коллизий в списке m_{jk}:

$$K(m_{jk}) = \begin{cases} 0, & m_{jk} = nil \\ Length\ (m_{jk}) - 1, & m_{jk} \neq nil \end{cases}$$

Количество коллизий в расписании равно сумме коллизий по всем элементам матрицы расписания.

Рассмотрим примеры таких расписаний. Для большей наглядности номера классов преобразованы в их названия. Исходный фрагмент расписания, представленный в таблице 3, содержит две коллизии:

1. Учитель 1 на пятом уроке занят внеучебной работой, но в это же время у него назначено занятие у класса 5В.
2. Учителю 3 на третий урок назначены два занятия: одно у класса 5В и одно у класса 5Г.

	Понедельник					Вторник	...
Учитель	1 урок	2 урок	3 урок	4 урок	5 урок	...	...
1	( )	( )	(5А)	(5Б)	(* 5В)	...	...
2	(*)	(5А)	(5Б)	(5В)	(*)	...	...
3	( )	(5Б)	(5В5Г)	(5А)	(*)	...	...

Табл. 3. Пример фрагмента расписания с коллизиями

Попробуем устранить коллизии при помощи атомарных переходов. Для этого перенесём занятие класса 5В с пятого урока на второй и занятие класса 5Г с третьего урока на первый. Полученный фрагмент расписания, представленный в таблице 4, не содержит коллизий.

	Понедельник					Вторник	...
Учитель	1 урок	2 урок	3 урок	4 урок	5 урок	...	...
1	( )	(5В)	(5А)	(5Б)	(*)	...	...
2	(*)	(5А)	(5Б)	(5В)	(*)	...	...
3	(5Г)	(5Б)	(5В)	(5А)	(*)	...	...

Табл. 4. Пример фрагмента расписания без коллизий

Рассмотрим, как реализованный алгоритм локального поиска для задачи составления расписания работает на практике. Эксперименты проводились с двумя модельными школами. Для каждого из примеров алгоритм запускался 20 раз. В качестве исследуемых параметров рассматривались время работы алгоритма,

количество попыток – генераций новых начальных расписаний и количество совершённых для достижения решения переходов.

Школа 1.

Данный пример составлялся максимально приближенным к реальной обстановке дел в школах. В школе 20 классов и 30 учителей (5 из которых имеют частичную занятость). В неделе 30 уроков (5 дней по по 6 уроков). Классы имеют от 20 до 30 уроков в неделю, учебная нагрузка варьируется в зависимости от возраста учащихся и профиля класса. В результате 20 запусков алгоритма для разных начальных расписаний были получены величины, отражённые в таблице 5.

Школа 2.

Для проверки возможностей алгоритма был сконструирован пример, максимально неудобный для локального поиска. В школе 15 классов и 15 учителей (2 из которых имеют частичную занятость). В неделе 30 уроков. Классы имеют учебную нагрузку от 27 до 30 уроков в неделю в зависимости от профиля.

Сложность задачи заключается в том, что в решении будут заняты почти все периоды в расписании учителей и классов. В таких условиях алгоритму локального поиска трудно находить допустимые переходы и устранять коллизии.

Результаты 20 запусков алгоритма представлены в таблице 5. Следует отметить, что во всех прогонах алгоритм сумел найти решение.

		Школа 1	*Школа 2*
Время работы	Максимальное	0:00.0142	1:36.6872
	Минимальное	0:00.0007	0:03.7521
	Среднее	0:00.0036	0:35.4993
Количество попыток	Максимальное	4	185079
	Минимальное	1	7268
	Среднее	3	68316
Количество переходов	Максимальное	98	5659751
	Минимальное	22	221907
	Среднее	72	2088229

Табл. 5. Результаты работы программы составления школьного расписания для моделей «*Школа 1*» и «*Школа 2*»

В задаче составления расписания алгоритм локального поиска имеет дополнительное преимущество. Поскольку алгоритм локального поиска даёт возможность начать поиск из любого состояния пространства поиска, однажды полученное решение задачи может быть

использовано в качестве начального состояния для алгоритма, если изменился набор ограничений (например, некоторые пожелания учителей). При этом новое расписание, скорее всего, не будет кардинально отличаться от исходного.

Заключение

Можно сделать вывод о применимости стратегии локального поиска для задач удовлетворения ограничений. Для конструирования алгоритма локального поиска нужно сформулировать исходную задачу в терминах поиска решения в пространстве состояний.

Оценочную функцию, определяющую порядок просмотра состояний, имеет смысл задать как количество нарушенных ограничений. При этом исходная задача сведётся к поиску состояния, обращающего оценочную функцию в ноль.

Можно существенно ускорить поиск решения, сузив пространство поиска. В обеих рассмотренных задачах удалось выделить из множества ограничений задачи подмножество ограничений, которые легко удовлетворить. Поиск решения происходил только среди состояний, удовлетворяющих выделенным ограничениям. Начальное состояние выбиралось из суженного подпространства, операции перехода не выводили из этого подпространства. Такое сужение позволило сократить количество переменных, описывающих состояние, а также количество отношений, истинность которых необходимо проверять для вычисления оценочной функции состояния на каждом шаге алгоритма.

В построенных реализациях алгоритма локального поиска не производилось вычисление множества всех соседей состояния с последующим их оцениванием; была применена следующая стратегия: из текущего состояния выполнялся переход в первое найденное соседнее состояние, улучшающее значение оценочной функции. Испытания программ продемонстрировали, что стратегия оправдала себя.

Литература

1. Talbi E-G. Metaheuristics: from design to implementation // Wiley, 2009.
2. Aarts E., Lenstra J. K. Local Search in Combinatorial Optimization // John Wiley & Son, Chichester, 1997.
3. Tsang E. Foundations of Constraint Satisfaction // Academic Press, 1993.
4. Garey M. R., Johnson D. S. Computers and Intractability - A guide to NP-completeness // W.H. Freeman and Company, San Francisco, 1979.

Бурцев А. А., Рамиль Альварес Х.

Реализация средств объектно-ориентированного программирования в кросс-компиляторе языка ДССП-Т

Введение

В НИЛ троичной информатики ВМК МГУ (ранее НИЛ ЭВМ) созданы программный комплекс ТВМ (Троичная Виртуальная Машина) [1], имитирующий функционирование современного варианта троичного процессора двухстековой архитектуры, а также ДССП-ТВМ [2] – система разработки программ для ТВМ на языке ДССП-Т – троичном варианте языка ДССП (Диалоговой Системы Структурированного Программирования) [3-7].

ДССП – интерпретируемая система программирования, а язык ДССП-Т реализуется кросс-компилятором. Увы, не все возможности, обеспечиваемые интерпретатором, удаётся реализовать кросс-компилятором. Поэтому выразительные средства языка ДССП-Т приходится существенно ограничивать (по сравнению с языком обычной ДССП).

Так, например, интерпретатор ДССП позволяет определить в программе новое слово, а затем сразу же его исполнить. Это даёт возможность определять в ДССП-программе особые слова, так называемые слова-компиляторы, которые можно тут же применять для формирования новых слов с компиляцией их тел. А кросс-компилятор, увы, не позволяет определять в программе такие компилирующие слова, которые могли бы сразу же исполняться в момент компиляции самой этой программы.

Но как раз именно такая возможность использовалась ранее в интерпретаторе ДССП в качестве инструмента построения новых типов данных. Без неё язык ДССП-Т (версии 2d) тяготился серьёзными недостатками в отношении типов данных. Во-первых, его бедный ассортимент встроенных типов данных (TRYTE, TWORD) позволял оперировать только данными машинного уровня (трайтами и троичными словами). И во-вторых, что более досадно, в нём не было возможности построения новых типов данных, структурирования сложных высокоуровневых типов данных из имеющихся простых.

На ликвидацию этих недостатков языка ДССП-Т и была нацелена представляемая работа. В качестве поставленной задачи необходимо было обеспечить в кросс-компиляторе языка ДССП-Т средства построения новых типов данных, эквивалентные возможностям объектно-ориентированного программирования (ООП).

Поскольку механизм обеспечения ООП-средств, предложенный ранее для интерпретатора ДССП/С [8,9], напрямую реализовать в кросс-компиляторе не представлялось возможным, предстояло разработать иной способ решения поставленной задачи, сохранив при этом совместимость предложенных ООП-средств на уровне языка ДССП.

Далее в статье сначала представляются предложенные в ДССП ООП-средства построения новых типов данных и рассматриваются особенности их реализации в ДССП-интерпретаторе. А затем анализируются проблемы возможной реализации таких ООП-средств кросс-компилятором ДССП-ТВМ и предлагаются способы их разрешения.

1. Средства построения новых типов данных в ДССП

Встраиваемые в язык ДССП-Т средства построения новых типов данных, эквивалентные возможностям ООП, были сначала предложены и реализованы для версии ДССП-32р, а затем и для версии ДССП/С [8], ядро которой в целях обеспечения мобильности системы разрабатывалось на языке Си. Синтаксические языковые конструкции этих ООП-средств, а также механизм их реализации в ДССП-интерпретаторе подробно описаны в [9]. Здесь лишь кратко поясним их так, чтобы можно было ознакомиться далее с проблемами их реализации в кросс-компиляторе.

Для объявления нового типа в ДССП предлагаются две конструкции: структура и класс.

Объявление нового типа как структуры начинается словом **STRUCT:** и завершается словом **;STRUCT** , а располагаемые между ними определения объектов данных (с помощью слов **VAR**, **VCTR**, **ARR**) объявляют в этом случае не самостоятельные переменные и массивы, а поля определяемой структуры:

```
STRUCT: <имя_структуры> {имя типа новой структуры}
 VAR <имя_поля>... VAR <имя_поля> {поля структуры}
;STRUCT {вместо VAR можно использовать VCTR, ARR }
```

Пример объявления нового типа для работы с величинами-датами:

```
STRUCT: TDATE {тип для представления даты}
 VAR .Year  VAR .Month  VAR .Day
;STRUCT
```

После объявления нового типа данных можно использовать его имя для объявления объектов (переменных и массивов) такого типа:

```
TDATE VAR Date  TDATE 9 VCTR Dates
```

Имеющиеся в языке ДССП-Т префиксные операции над переменными и массивами можно теперь применять и для доступа как ко всему объекту типа структуры целиком:

```
Date 5 ! Dates {копирование даты Dates[5]:=Date }
```

так и к отдельным его полям:

```
Date {'Date} .Year {Date.Year}
5 Dates ! .Year {Dates[5].Year:= Date.Year}
```

Ещё одним средством построения новых типов данных в языке ДССП-Т являются классы. Объявление нового типа как класса начинается со слова **CLASS:** и заканчивается словом **;CLASS** , а внутри такого объявления можно задавать не только поля (как и при объявлении структуры), но и методы класса:

```
CLASS: <имя_класса> {имя типа нового класса}
 VAR <имя_поля>... VAR <имя_поля> {объявление полей}
  METHOD <имя_метода> {объявление методов}
  METHOD= <прежнее_имя_метода> <новое_имя_метода>
 N METHOD# <новое_имя_метода> {N-номер прежнего метода}
;CLASS
```

Словом **METHOD** объявляется новая операция, которую можно будет совершать над объектом этого типа-класса помимо тех методов (префиксных операций **!** и **'**), которые над ним уже предусмотрены изначально (как и над структурами). Слово **METHOD=** позволяет назначить новое имя методу, используя его прежнее имя, а слово **METHOD#** – задав номер метода.

Пример объявления типа-класса для работы с очередью:

```
:: CLASS:  QUEUE  { Класс ОЧЕРЕДЬ }
  {--- поля данных, составляющих очередь ---}
    VAR .cnt { кол-во элементов в очереди }
    VAR .n   { индекс-указатель начала очереди в массиве }
    VAR .k { индекс-указатель конца очереди в массиве }
 100 VCTR .MI { массив для хранения элементов очереди }
  {--- методы, представляющие операции над очередью ---}
 :: METHOD Init { инициировать, начать работу очереди }
 :: METHOD Show { показать на экране состояние очереди }
 :: METHOD Get { взять элемент из очереди на вершину стека }
 :: METHOD Put { поместить элемент из вершины стека в очередь}
 :: METHOD Empty? { проверить, пуста ли очередь }
 :: METHOD Full? { проверить, полна ли очередь }
 :: METHOD Done { завершить, закончить работу очереди }
 :: METHOD= ! => {новое имя метода для копирования очереди}
;CLASS
```

Для назначения процедуры, реализующей объявленный метод класса, предлагаются синтаксические конструкции:

```
<имя_класса> :M: <имя_метода> <тело_реализации_метода>... ;
```

```
<имя_класса>  :M=  <имя_метода>  <имя_процедуры_метода>
```

Имя типа класса следует задать перед употреблением слова **:M:** (или **:M=**). Слово **:M:** создаёт тело новой процедуры, компилируя его как и слово : (до ;), а слово **:M=** принимает имя уже существующей процедуры, чтобы назначать её в качестве исполнителя метода.

Примеры реализации некоторых методов класса QUEUE :

```
QUEUE  :M: Init {Q} 0 C2 ! .cnt { Q.cnt:=0 }
        0 C2 ! .k 1 E2 ! .n { } ; { Q.k:=0; Q.n:=1 }
QUEUE  :M= Empty? .Empty?
       : .Empty? {Q} .cnt 0 <= {1/0 Q.cnt<=0?};
QUEUE  :M: Get {Q}
           C .Empty? IF+ Empty! {проверка искл.ситуации}
           C .n C2 .MI E2 { y:= Q.M[Q.n] }
     {y,Q} C .cnt 1- C2 ! .cnt { Q.cnt:=Q.cnt-1 }
           C .n +1mod E2 {y,n+1,Q} ! .n
              {Q.n:=(Q.n+1)mod Maxlen } {y} ;
```

Примеры объявлений объектов типа QUEUE :

```
 QUEUE VAR Q      { Q - отдельный экземпляр класса QUEUE }
 9 QUEUE VCTR VQ  { массив очередей VQ[0:9] }
```

Примеры вызовов действий с объявленными объектами:

```
{вызов операций по имени объектов:} Q{адрес Q} 3 VQ {'VQ(3)}
Q .cnt {Q.cnt}{получение значения кол-ва элементов очереди }
0 Q ! .cnt {Q.cnt:=0}{обнуление счетчика элементов очереди}
3 VQ ! Q { Q:= VQ[3] копирование очереди целиком }
3 VQ => Q {то же, но применяя новое обозначение метода }
Init Q { вызов метода для инициализации очереди Q }
77 Put Q { помещение в очередь Q числа 77 }
3 Get VQ {y} { взятие значения (y) из очереди VQ[3] }
```

Внутри объявления нового типа как класса с помощью слова **SUBCLASS** можно задать, какой уже известный тип использовать для наследования:

```
CLASS: <имя_класса> { имя типа нового класса }
 SUBCLASS <имя_наследуемого_класса>
 VAR <имя_поля>... VAR <имя_поля> {объявление новых полей}
 METHOD <имя_метода> {объявление новых методов}
 METHOD= <прежнее_имя_метода>    <новое_имя_метода>
 N METHOD# <новое_имя_метода> {N-номер прежнего метода}
;CLASS
```

Такое наследование означает, что у нового объявляемого типа изначально считаются уже заданными все поля и методы, которыми обладает наследуемый тип, после чего можно добавлять к его определе-

нию новые поля, задавать новые методы, а также переопределять исполнителей прежних методов.

Пример объявления класса-наследника:

```
{--- Объявление класса ДЕК ---}
:: CLASS:  DEQ  SUBCLASS QUEUE { наследует класс QUEUE }
 {--- методы , дополняющие операции над очередью ---}
 :: METHOD Put_ { поместить элемент в начало очереди }
 :: METHOD Get_ { взять элемент с конца очереди }
 :: METHOD Count { узнать количество элементов в очереди }
;CLASS
```

2. О реализации ООП-средств в ДССП-интерпретаторе

Определение нового типа данных (см. пример QUEUE) в ДССП с помощью компилирующих слов:

STRUCT: ;STRUCT CLASS: SUBCLASS METHOD ;CLASS

предполагает создание в словаре:

- слова с именем нового типа данных: QUEUE ;
- слов с именами новых полей: .cnt .n .k .MI ;
- слов с именами новых методов:

  Init Show Get Put Done => Empty? Full?.

Требуется построить тела для всех этих слов, а также:

- тело дескриптора для нового типа данных;
- тела процедур-исполнителей новых методов.

Слова с именами новых полей и их тела (дескрипторы) формируются почти так же, как это делается для обычных переменных. Но для полей используется иная функция вычисления адреса, которая использует значение вершины стека как базовый адрес структуры, добавляя к нему смещение, хранимое в дескрипторе данного поля.

Слова новых методов помечаются Р-флагом как префиксные операции, а их тела формируются согласно единому образцу вида:

```
CALL_ExecMethod {адрес универсальной процедуры вызова метода}
.tword 8 { номер метода}
```

Строение тела слова имени типа (QUEUE) эквивалентно процедуре назначения системной переменной DTYPE адреса дескриптора типа QUEUE'Ptr :

```
: QUEUE QUEUE'Ptr ! DTYPE ;
```

Строение дескриптора типа детально изображено на рис.1. Оно состоит из полей, задающих адрес дескриптора наследуемого типа (**PredokPtr**), количество методов (**MtdNum**), размер объекта (**Size**), за которыми располагается таблица методов, содержащая коды команд или адреса для вызова процедур-исполнителей методов.

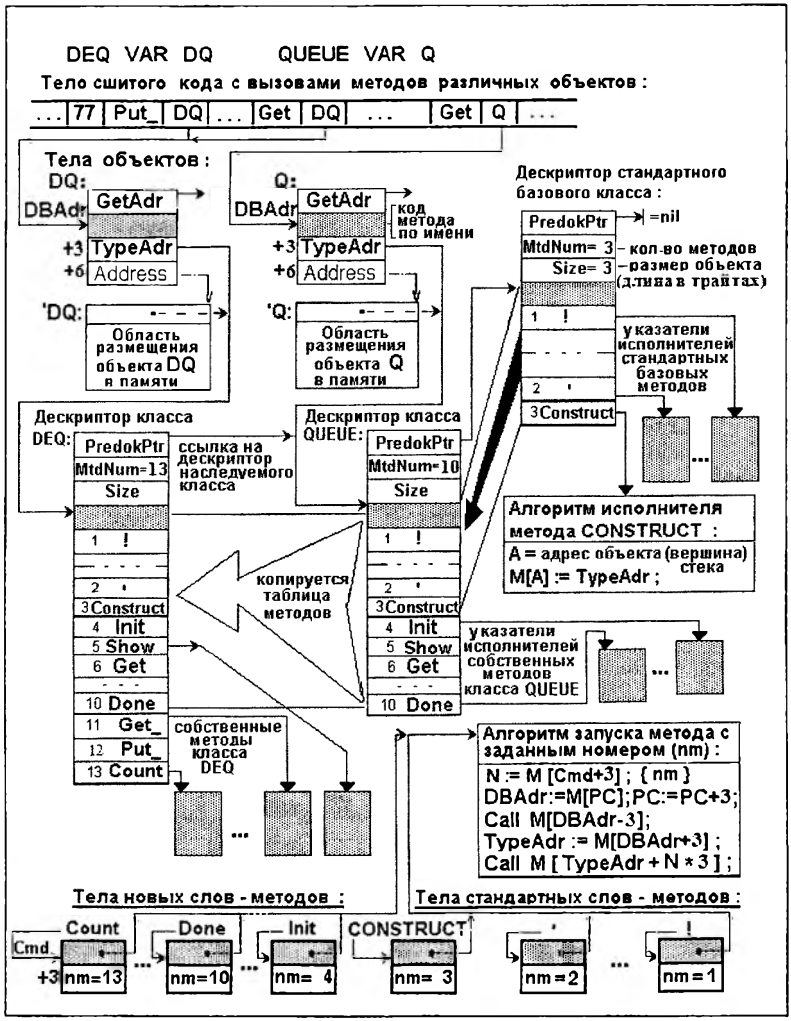


Рис. 1. Механизм реализации методов класса

На рис.1 наглядно показано, как осуществляется вызов метода над объектом класса, какие при этом используются структуры данных (дескрипторы объектов-переменных, дескрипторы типов).

Для построения дескриптора нового типа (например, DEQ) необходимо сначала заполнить поля PredokPtr, NtdNum, Size, затем скопировать таблицу методов у дескриптора наследуемого типа, после чего

дополнить эту таблицу ссылками на тела процедур-исполнителей новых методов.

Все описанные действия по построению необходимых структур данных достаточно легко осуществить в ДССП-интерпретаторе, т.к. во время его функционирования ему полностью доступна вся память, используемая ДССП-программой. А вот чтобы подготавливать требуемые структуры данных в ассемблерном коде, последовательно формируемом кросс-компилятором, потребовалось решить ряд проблем и существенно модифицировать работу кросс-компилятора.

3. О проблемах ООП-реализации в кросс-компиляторе

1. Компилятор функционирует в цикле обработки потока известных ему слов-компиляторов, для каждого из которых у него заранее заготовлена своя Си-процедура. Как же "научить " компилятор понимать новые компилирующие слова, представляющие имена вновь определённых типов (например, QUEUE, DEQ) ? (проблема №1)

2. Компилятор последовательно (за один проход!) генерирует ассемблерный код, отправляя его в выходной файл. Как обеспечить копирование дескриптора наследуемого класса ? (проблема №2)

3. Дескриптор требуемого типа как и номер метода теперь невозможно напрямую взять из тела (как ранее), т.к. сформированные тела в ассемблерном коде помещаются в выходной результирующий файл. Как же узнать номер метода при определении процедуры его исполнения? (проблема №3)

4. Модификация кросс-компилятора для реализации средств построения новых типов данных

Рассмотрим, какие модификации пришлось сделать в кросс-компиляторе для разрешения этих проблем.

4.1 Дополнительный словарь имён типов

Для слов, представляющих имена новых типов, образуется дополнительный словарь, в каждом узле которого содержатся:

- 1) строка **IName** слова имени типа;
- 2) индекс **ICode** для доступа в таблицу к телу его дескриптора;
- 3) строка **ILabel** метки для ассемблерного тела дескриптора.

При этом изменяется алгоритм обработки компилирующих слов: если встреченное компилирующее слово не найдено в основном словаре, то компилятор пытается найти это слово в дополнительном словаре типов. Если найдено слово типа, то его номер фиксируется в спецпеременной **nTYPE** так, чтобы использовать этот номер типа для

последующих объявлений данных (переменных и массивов).

Когда в программе объявляется новый тип данных (см. например, объявление класса QUEUE) компилятор создаёт новое слово (QUEUE) и в своём словаре типов, и в ДССП-словаре компилируемой программы, формируя для него такое ассемблерное тело в выходном файле, чтобы при его вызове спецпеременная **DTYPE** получала значение адреса дескриптора этого типа:

```
call_LIT  ; tword TN1008   {{ '' QUEUE'Ptr
DPushI_42 ; Store          {{ 'DTYPE !W
RET                        {{ ;
```

При этом сам дескриптор типа формируется компилятором в отдельной области памяти (называемой далее таблицей дескрипторов типов) и будет выводиться в файл ассемблерного кода вместе с назначенной ему меткой (**ILabel**=TN1008) лишь по завершении компиляции всей ДССП-программы.

4.2 Таблица дескрипторов типов

Такая таблица создаётся в компиляторе как массив (**TypeBody**) значений, которые должны представлять тела дескрипторов объявленных типов сначала в компиляторе, а затем и в ассемблерном коде формируемой программы. Индекс-ссылка (**ICode**) из каждого узла слова имени типа указывает на расположение тела его дескриптора в этом массиве.

При наследовании типа его дескриптор копируется внутри этого массива. Затем в нём изменяются поля счётчика методов и размера объекта, а в его таблицу методов заносятся номера слов-исполнителей методов.

После завершения компиляции ДССП-программы итоговое содержимое массива отправляется в файл ассемблерного кода с учётом того, что элементы массива (long int) имеют разный смысл (см. табл.1). Заметим, что тело дескриптора нельзя выдавать в ассемблерный файл сразу же после его создания, поскольку при дальнейшей компиляции ДССП-программы в его таблице методов ещё будут корректироваться ссылки на тела процедур, назначаемых в качестве исполнителей этих методов. В массиве такими ссылками служат номера слов, а в ассемблерный код помещаются соответствующие их словам мнемокоды команд или метки, которые превращаются далее в адреса тел (или команды вызова) процедур-исполнителей методов.

	в массиве:	на ассемблере:	
-3	номер типа-предка	.tword TN1007	адрес дескриптора-предка
-2	счётчик методов	.tword 10	числовое значение
-1	размер объекта	.tword 315	числовое значение
0	Таблица методов:	TN1008: NOP	Таблица методов:
1	содержит номера	call StructCopy	содержит ассемблерные
2	слов процедур,	NOP	коды команд исполните-
3	назначенных ис-	call PN2232	лей методов или команды
...	полнителями ме-	...	(адреса) вызовов их про-
10	тодов	call PN2373	цедур

Таблица 1. Строение тела дескриптора типа

4.3 Поле номера метода в узле слова ДССП-словаря

Номера слов, назначаемых в качестве исполнителей методов, требуется вносить в таблицу методов дескриптора типа по мере определения тел их процедур: `QUEUE :M: Get ... ;`

Для этого необходимо по имени метода (`Get`) узнавать его номер. В интерпретаторе для этого достаточно "заглянуть" в память на второе слово тела метода (см. рис.1). А в кросс-компиляторе для решения этой проблемы (№3) потребовалось добавить дополнительное поле в узел слова ДССП-словаря, чтобы хранить в нём номер метода вместе с флагом-признаком метода.

Это позволило получить возможность корректировать поля таблицы методов в дескрипторе типа, применяя такой алгоритм:

1) после выполнения слова с именем типа `QUEUE` спецпеременная **nTYPE** получила номер слова-типа nTYPE:= nQUEUE;
2) по этому номеру из узла словаря типов получаем ссылку ICode на его дескриптор в массиве TypeBody;
3) по имени метода (Get) получаем (из поля его узла ДССП-словаря) номер метода **nMtdGet**;
4) изменяем элемент таблицы методов в массиве тел дескрипторов TypeBody[ICode+nMtdGet], записывая в него номер слова, назначенного в качестве исполнителя метода QUEUE::Get.

Заключение

В результате осуществлённой модификации кросс-компилятора в языке ДССП-Т (версии 3а) обеспечены возможности построения новых типов данных, эквивалентные средствам объектно-ориентированного программирования.

На их основе в ДССП-библиотеке были определены новые типы данных: **ACT** для объявления процедурных переменных; **TSET** для работы с нумерованными тритами троичного слова; а также **TCLASS** как базовый тип для всех классов с методом **CONSTRUCT**.

Литература

1. Сидоров С.А., Владимирова Ю.С. Троичная виртуальная машина. // Программные системы и инструменты. Тематический сборник №12, М.: Изд-во факультета ВМиК МГУ, 2011. с.46-55.
2. Бурцев А.А., Рамиль Альварес Х. Кросс-система разработки программ на языке ДССП для троичной виртуальной машины // Прграмм-ные системы и инструменты. Тематический сборник №12, М.: Изд-во факультета ВМиК МГУ, 2011. с.183-193.
3. Брусенцов Н.П., Златкус Г.В, Руднев И.А. ДССП - диалоговая система структурированного программирования. // Программное оснащение микрокомпьютеров. М.: Изд-во МГУ, 1982, с.11-40.
4. Брусенцов Н.П., Захаров В.Б., Руднев И.А., Сидоров С.А., Чанышев Н.А. Развиваемый адаптивный язык РАЯ диалоговой системы программирования ДССП. М.: Изд-во Моск. Ун-та, 1987 г. – 80 с.
5. Сидоров С.А., Шумаков М.Н. ДССП как открытая система. // Дискретные модели. Анализ, синтез и оптимизация. Спб.: СпбГУ, 1998. с.191-201.
6. Бурцев А.А. ДССП – среда структурированной разработки программ как сложных систем. // Вторая Международная конференция "Системный анализ и информационные технологии" САИТ-2007 (10-14 сентября 2007 г., Обнинск, Россия): Труды конференции. Изд-во ЛКИ, 2007. т. 2. с.190-194.
7. Бурцев А.А., Сидоров С.А. История создания и развития ДССП: от "Сетунь-70" до троичной виртуальной машины. // Вторая Международная конференция "Развитие вычислительной техники и её программного обеспечения в России и странах бывшего СССР" SORUCOM-2011 (12-16 сентября 2011 г., г. Великий Новгород, Россия): Труды конференции. В.Новгород: Изд-во НовГУ, 2011. с. 83-88.
8. Бурцев А.А., Франтов Д.В., Шумаков М.Н. Разработка интерпретатора сшитого кода на языке Си. // Вопросы кибернетики. Сб. статей под ред. В.Б.Бетелина. М., 1999. с.64-76.
9. Бурцев А.А., Рамиль Альварес Х. Средства объектно-ориентированного программирования в ДССП. // Программные системы и инструменты. Тематический сборник №4, М.: Изд-во факультета ВМК МГУ, 2003. с.166-175.

Ершов Н. М.

Неоднородные клеточные генетические алгоритмы

1. Введение

Клеточные генетические алгоритмы [1] обладают рядом преимуществ по сравнению с обычными генетическими алгоритмами. Во-первых, за счет локальности взаимодействия между особями популяции удается более долгое время поддерживать разнообразие в популяции, что потенциально ведет к получению более качественного решения. Во-вторых, благодаря регулярности расположения особей в клеточном пространстве и отсутствию глобальных операций клеточные генетические алгоритмы хорошо и масштабируемо распараллеливаются [2]. Однако, как и в обычных генетических алгоритмах, в клеточном варианте остается актуальной проблема попадания алгоритма в локальные экстремумы. В настоящей работе предлагается подход к решению этой проблемы, основанный на введении зависимости работы операторов генетического алгоритма (прежде всего, мутации) от положения особи в клеточном пространстве.

2. Оптимизационная задача

Принципы работы неоднородных клеточных генетических алгоритмов рассмотрим на примере решения задачи многомерной *непрерывной* оптимизации функции $F(x)$ в области $x_k \in [-100, 100]$, $k \in \{1 \ldots n\}$. Рассматривались две функции --- бимодальная и мультимодальная. Первая функция:

$$F(\theta, x, y) = G(1, x + 5, y + 5) + 2\,G(\theta, x - 5, y - 5),$$

является двумерной и представляет собой сумму (с отрицательным знаком) двух гауссианов:

$$G(\theta, x, y) = 1 - e^{-\frac{x^2+y^2}{2\theta}},$$

первый из которых имеет фиксированную ширину, а ширина второго определяется параметром θ. Функция имеет два минимума. Минимум в точке (–5, –5) является локальным и имеет фиксированную ширину. Минимум в точке (5, 5) является глобальным, значение функции в нем равно примерно 1. Параметр θ в численных экспериментах был меньше 1 ($0.001 \leq \theta \leq 0.5$), поэтому область «притяжения» глобального минимума является более узкой по сравнению с локальным минимумом.

Вторая функция, с которой проводилось исследование – функция Растригина [3]:

$$R(x) = \frac{1}{100}\sum_{k=1}^{n}\left[(x_k - 20)^2 + A\left(1 - \cos\frac{\pi x_k}{5}\right)\right].$$

Эта функция является мультимодальной с единственным глобальным минимумом в точке $x_k = 20$, $k \in \{1...n\}$. В двумерном случае в заданной области функция Растригина имеет 400 локальных минимумов, а 10-мерная функция – уже порядка 10^{13} минимумов. В численных экспериментах использовались функции с параметром $A = 2 \cdot 10^5$. Коэффициенты функции Растригина подобраны так, что локальные минимумы располагаются в точках вида $x_k = 10l$, $l \in Z$, а значения функции в этих точках является суммой n квадратов целых чисел.

3. Однородные генетические алгоритмы

Сравнение эффективности неоднородного клеточного генетического алгоритма (NCGA) производилось с генетическим алгоритмом (GA) и клеточным генетическим алгоритмом (CGA). Во всех рассматриваемых вариациях алгоритмов использовались одинаковые генетические операторы (отбор, скрещивание, мутация и миграция). Для отбора применялась турнирная схема, в которой пара выбранных особей решает вопрос о выживании одной из них. Особь с лучшим значением целевой функции выживает с вероятностью p_{win}. Победитель замещает своей копией место побежденного. Для скрещивания использовалась равномерная схема, когда с заданной вероятностью p_{swap} меняются местами два соответствующих гена двух заданных особей. В силу специфики решаемой задачи (непрерывная оптимизация) оператор мутации выполняет случайное изменение каждого гена заданной особи с достаточно большой вероятностью p_{mut}:

$$x_k \leftarrow x_k + \rho d_{mut},$$

где ρ – случайное действительное число из диапазона [-1, 1], d_{mut} – величина мутации. Чтобы обеспечить разнообразие популяции на начальном этапе работы алгоритма и сохранить его сходимость, применялась технология *имитации отжига* по параметру d_{mut}. Оператор миграции использовался только в клеточных вариантах генетического алгоритма. Этим оператором две выбранные особи меняются местами (т.е. обмениваются своими геномами).

Для однородного генетического алгоритма использовалась следующая схема работы. Выполнялось заданное число итераций. На каждой итерации выполнялись последовательно три оператора: отбор,

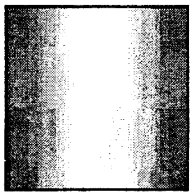
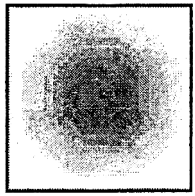
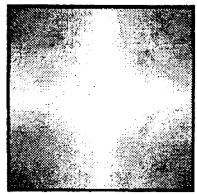
скрещивание и мутация. После каждого оператора производилось полное перемешивание популяции. Для отбора и скрещивания брались пары соседних (по номерам) особей $2i$ и $2i + 1$, $i \in [0...n/2)$.

Однородный клеточный генетический алгоритм работал по следующей схеме. Все особи размещались по одной в клетках прямоугольной области. На каждой итерации алгоритма сначала выполнялись операторы отбора, скрещивания и миграции. Для этого клетки случайным образом делились на пары, так чтобы две клетки в одной паре всегда были соседними (т.е. имели бы общую сторону). Последним шагом на итерации выполнялась мутация всех особей.

4. Неоднородные клеточные генетические алгоритмы

Как уже упоминалось выше, проблемой однородных (клеточных или нет) генетических алгоритмов является то, что в конце-концов они сходятся к однородной (гомогенной) популяции. В такой популяции практически все особи имеют одинаковые геномы, поэтому наиболее мощный генетический оператор – скрещивание – перестает работать. Если алгоритм попал в локальный экстремум, то выйти из него можно только за счет мутации. Если мутация локальна, а более хорошее решение значительно удалено от найденного, то вероятность выхода из данного локального минимума оказывается очень низкой. Таким образом, проблема заключается в потере генетического разнообразия.

Используя клеточные генетические алгоритмы, оказывается возможным поддерживать разнообразие популяции сколь угодно долгое время, не теряя при этом сходимости. Суть идеи заключается в том, что в некоторые параметры генетического алгоритма делаются зависимыми от положения особи в клеточном пространстве. В простейшем варианте такой модификации подвергается величина мутации d_{mut}. В одних областях мы делаем значение этого параметра высоким, в других – низким. Поэтому, первые области будут *все время* генерировать достаточно случайные решения, поддерживая, таким образом, необходимое разнообразие всей популяции. Области второго типа будут использоваться по своему основному назначению – селекции и скрещиванию лучших решений. Для экспериментов было выбрано три варианта зависимости параметра d_{mut} от координат i и j клеток.

а) NCGAs б) NCGAm в) NCGAc

Рис. 1. Градиент величины мутации

В первом варианте (NCGAs, рис. 1а) параметр d_{mut} зависит только от горизонтальной координаты, так что по краям поля значение этого параметра является высоким, в середине – низким. Во втором варианте мутация в центре поля является высокой, а по его краям – низкой (NCGAm, рис. 1б). Наконец, в третьем случае, области с высокой величиной мутации располагались в четырех углах поля (NCGAc, рис 1в). Таким образом, никакие параметры генетического алгоритма в неоднородном варианте не зависят от времени t.

5. Минимизация бимодальной двумерной функции

В задаче минимизации бимодальной целевой функции $F(\theta, x, y)$ измерялось частота (в процентах) обнаружения генетическим алгоритмом глобального минимума функции. Для этого выполнялось 100 запусков каждой версии генетического алгоритма. Полученные результаты показаны в таблице 1.

θ	GA	CGA	NCGAs	NCGAm	NCGAc
0.001	0	0	4	4	4
0.002	0	0	5	5	5
0.005	0	0	6	8	9
0.010	0	0	10	15	14
0.020	0	0	28	24	25
0.050	3	15	61	47	54
0.100	30	76	87	72	80
0.200	74	100	100	95	97
0.500	100	100	100	100	100

Табл. 1.

Все алгоритмы в данном случае выполняли одинаковое число (500) итераций. Видно, что частота обнаружения глобального минимума неоднородными алгоритмами существенно выше по сравнению с однородными версиями. При этом, заметим, что показатели неоднородного алгоритма (последние три столбца таблицы) могут быть улучшены, за счет увеличения числа итераций, в то время, как показатели однородных алгоритмов таким образом уже не улучшаются.

6. Минимизация мультимодальной двумерной функции

Минимизация двумерной функции Растригина $R(x, y)$ выполнялась с помощью однородных генетических алгоритмов (GA и CGA) и неоднородного алгоритма NCGAs. Вычислялось частота обнаружения алгоритмом того или иного локального минимума R. Результаты численного эксперимента приведены в таблице 2. В первом столбце таблицы показаны первые пять локальных минимумов (0 – глобальный минимум), заметим, что в данной задаче нет локального минимума со значением 3. В остальных столбцах таблицы показаны частоты попадания указанных алгоритмов в заданные локальные минимумы. Для неоднородного алгоритма показано, как изменяются частоты ответов в зависимости от числа итераций.

	GA	CGA	NCGA	NCGA	NCGA	NCGA
R	500	500	500	1000	2000	3000
0	51	44	65	82	90	98
1	41	46	29	18	10	2
2	8	6	5	0	0	0
4	0	3	1	0	0	0
5	0	1	0	0	0	0

Табл. 2.

Таким образом, видно, что уже при 3000 итераций неоднородный алгоритм практически гарантированно находит глобальный экстремум (из имеющихся 400 локальных экстремумов). При этом, сходимость однородных алгоритмов таким образом не улучшается.

7. Минимизация мультимодальной многомерной функции

Наконец, в последнем эксперименте исследовалась частота попадания генетического алгоритма в локальные минимумы 10-мерной функции Растригина $R(x)$. В этом случае имеется порядка 10^{13}

локальных минимумов, из которых всего один является глобальным. Все минимумы имеют целые неотрицательные значения.

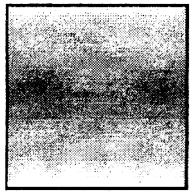


Рис. 2. Градиент вероятности отбора p_{sel}

Первые расчеты показали, что даже неоднородные алгоритмы не способны обнаружить глобальный минимум (а также близкие к нему локальные минимумы). Поэтому в алгоритм NCGAs была внесена еще одна неоднородность: вероятность выполнения оператора отбора для двух особей из соседних клеток была сделана зависящей от вертикальной координаты i:

$$p_{sel} = 1 - | i / w - 1|.$$

Таким образом, сверху и снизу клеточного поля отбор практически не работает (рис. 2), что дает образующимся (за счет высокой мутации) в угловых областях особям больше времени на улучшение целевой функции.

Численные результаты показаны в таблице 3. В первом столбце таблицы перечислены первые 8 минимумов функции, последняя строка соответствует минимумам с большими значениями функции. В однородных алгоритмах выполнялось по 500 итераций, в неоднородном – 1000, 3000 и 5000 итераций.

	GA	CGA	NCGA	NCGA	NCGA
R	500	500	1000	3000	5000
0	0	2	0	17	64
1	5	7	0	36	31
2	15	16	3	29	4
3	26	19	7	12	1
4	9	20	3	2	0
5	16	13	13	4	0
6	10	5	13	0	0
7	3	10	11	0	0
>7	16	8	50	0	0

Табл. 3.

Видно, что с увеличением числа итераций неоднородный алгоритм все чаще обнаруживает глобальный экстремум, в то время, как однородные алгоритмы глобальный минимум практически не находят.

8. Заключение

В результате выполненной работы были получены следующие результаты:

- введено понятие неоднородного клеточного генетического алгоритма;
- проведено численное сравнение работы однородного генетического алгоритма, однородного клеточного генетического алгоритма и неоднородного генетического алгоритма на трех задачах непрерывной многомерной оптимизации;
- показаны преимущества предложенного подхода.

Литература

1. Alba, E. and Dorronsoro, B. Cellular Genetic Algorithms, Springer, 2008.
2. Whitley D. A genetic algorithm tutorial, Statistics and Computing, Vol. 4, No. 2. (1 June 1994), pp. 65-85.
3. Ke Tang, Xiaodong Li, Suganthan P.N., Zhenyu Yang, Weise T. Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization, Technical report, University of Science and Technology of China (USTC), 2010.

Малышко В. В., Манжосов А. Н.

Решение задач инженерии знаний средствами объектно-ориентированной инженерии программного обеспечения[1]

1. Введение

Наша работа выполнена в рамках исследования новых методов работы с базами знаний систем планирования, посвящена вопросам моделирования знаний и верификации знаний. Подход, которого мы придерживаемся, основывается на адаптации современных языковых и программных средств инженерии программного обеспечения к использованию в области искусственного интеллекта.

Развитие языка объектно-ориентированного моделирования программных систем UML [Арлоу, Нейштадт 2007] реализовало возможность создания сложных, подробных и точных моделей для любых предметных областей. Элементы таких моделей можно связывать со строгими формальными ограничениями, составленными в виде выражений языка OCL [Warmer, Kleppe 2003]. Среды проектирования, поддерживающие UML, стали достаточно функциональными и широко применяются в индустрии ПО.

На наш взгляд, есть предпосылки к тому, чтобы использовать UML не только при разработке программных систем. Перечисленные возможности можно результативно применять в инженерии знаний. На смену специализированным инструментам инженера знаний могут прийти универсальные UML-средства из обихода разработчика программного обеспечения. Для этого необходимо предложить представление знаний с помощью UML-моделей и создать средства их перевода в описания знаний на традиционных языках искусственного интеллекта.

Первые шаги к практическому использованию UML в инженерии знаний были сделаны ранее. Так, в рамках проекта itSIMPLE (the Integrated Tools Software Interface for Modeling PLanning Environments) [Vaquero, Tonaco et al 2012], ведущегося в университете Сан-Паулу, создана интегрированная среда для моделирования предметных областей планирования. В среде itSIMPLE предметные области планировщиков и условия задач планирования описываются совокупностью UML-диаграмм. Среда содержит средства анализа знаний, симуляции выполнения планов.

[1] Работа выполнена при поддержке РФФИ (№11-01-00638-а)

В отличие от бразильских коллег, мы полагаем, что моделировать предметные области и задачи можно не в специализированной среде инженерии знаний, а в универсальной среде программной инженерии, поддерживающей UML, например, в Eclipse [Steinberg, Budinsky et al 2008]. Для этого мы предлагаем схему, приведённую на рис. 1.

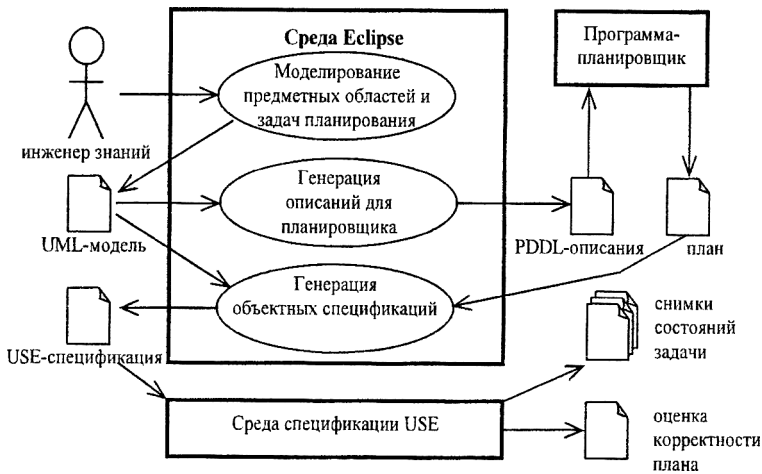


Рис. 1. Схема применения комплекса средств для решения задач инженерии знаний

Среда Eclipse позволяет создавать UML-модели. С её помощью инженер знаний сможет составить базу знаний в виде UML-описаний. По созданной UML-модели могут быть сгенерированы описания на входном языке планировщика. Стандартным языком планировщиков, de facto, является PDDL [Gerevini, Long 2005]. Получив знания на языке PDDL, планировщик может найти план решения задачи. Полученный план может быть исследован. В схеме указано, что по UML-модели и найденному плану могут быть созданы спецификации для среды USE (UML-based Specification Environment) [Gogolla, Büttner et al 2007], осуществляющей симуляцию выполнения объектных моделей и их верификацию. Таким образом, среда USE сможет осуществить оценку найденного плана и имитацию его выполнения. В результате её работы будет получен набор диаграмм, описывающих финальное и промежуточные состояния задачи, а также вердикт о корректности или некорректности плана.

Рассматриваемый комплекс инструментальных средств позволит решать задачи, стоящие перед инженером знаний, при условии, что будет предложен способ представления знаний на языке UML, будет реализован генератор PDDL-описаний по UML-моделям и будет реализован генератор USE-спецификаций. Все эти условия нами выполнены. В оставшейся части статьи будут описаны способ представления и два генератора, а также пример их использования.

Охарактеризуем предлагаемый подход. В его рамках стирается граница между инженерией ПО и инженерией знаний. Тем самым, модели знаний могут применяться при разработке ПО, а программные модели – в инженерии знаний. Например, модель, первоначально предназначенная для задания входных данных программы-планировщика, может быть повторно использована при разработке программного обеспечения, связанного с моделируемой предметной областью, так как она ничем принципиально не отличается от любой другой UML-модели. Следовательно, опыт, накопленный разработчиками ПО, может быть использован в инженерии знаний и наоборот.

Теперь рассмотрим применение языка UML для представления знаний о предметной области. В качестве примера выберем, так называемую, задачу о лифте. В этой предметной области выделяются три типа объектов: этаж, лифт и пассажир. Этажи образуют упорядоченный набор. Лифт перемещается между этажами вверх и вниз, перевозя одновременно не более некоторого фиксированного числа пассажиров. Пассажир может зайти в лифт, если они находятся на одном и том же этаже, и в лифте достаточно свободных мест. Также пассажир может покинуть лифт на каком-либо этаже.

UML-модель предметной области рассматриваемой задачи приведена на рисунке 2. В ней присутствуют три класса: *Elevator* (лифт), *Floor* (этаж), *Passenger* (пассажир). Каждый класс соответствует типу объектов предметной области. Отношения между объектами задачи представлены ассоциациями: *at* (пассажир находится на некотором этаже), *atFloor* (лифт находится на некотором этаже), inside (пассажир находится в лифте), *floorToFloor* (этаж расположен над другим этажом). Атрибутом *capacity* класса *Elevator* обозначено свойство объектов-лифтов – текущее количество свободных мест в лифте. Операции класса *Passenger getIn(e)* и *getOut()* обозначают действия пассажира, а операции класса *Elevator moveUp()* и *moveDown()* – подъём и спуск лифта на соседний относительно текущего этаж.

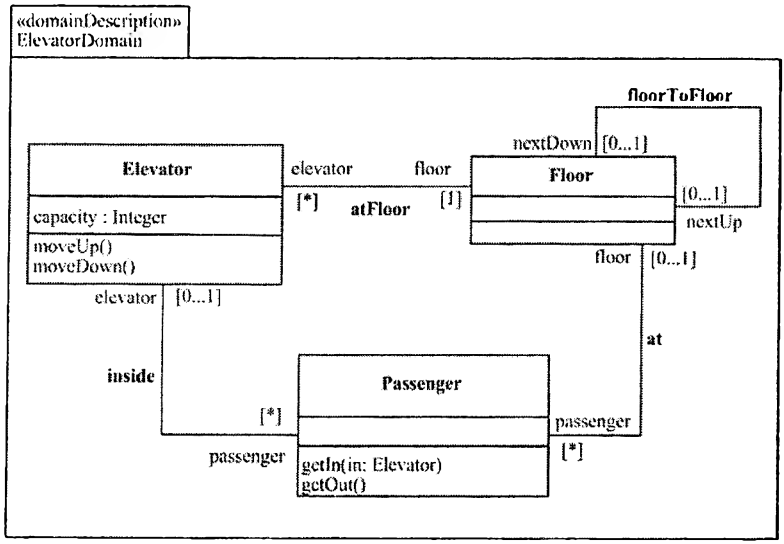


Рис. 2. UML-модель предметной области задачи о лифте

Эффекты и предусловия действий описываются следующими OCL-выражениями:

context Elevator::moveUp()
pre: self.floor.nextUp->notEmpty()
post: self.floor = self.floor@pre.nextUp
context Elevator::moveDown()
pre: self.floor.nextDown->notEmpty()
post: self.floor = self.floor@pre.nextDown
context Passenger:getIn(e:Elevator)
pre: self.floor = e.floor and e.capacity > 0
post: self.elevator = e and self.floor->isEmpty()
and e.capacity = e.capacity@pre-1
context Passenger:getOut()
pre: self.elevator->notEmpty()
post: let e = self.elevator in (e->isEmpty() and
self.floor = e@pre.floor and
e@pre.capacity = e@pre.capacity@pre + 1)

Начальные условия конкретной задачи описываются диаграммой объектов, на которой представлены экземпляры классов, их свойства и связи. Рассмотрим экземпляр задачи, в котором присутствуют четыре этажа *floor1*, *floor2*, *floor3*, *floor4*, три пассажира: *Fred*, *George* и *Harry*, первые двое которых находятся на четвёртом

этаже, а третий – на втором. Лифт находится на первом этаже, и в нём два свободных места. Соответствующая диаграмма представлена на рисунке 3.

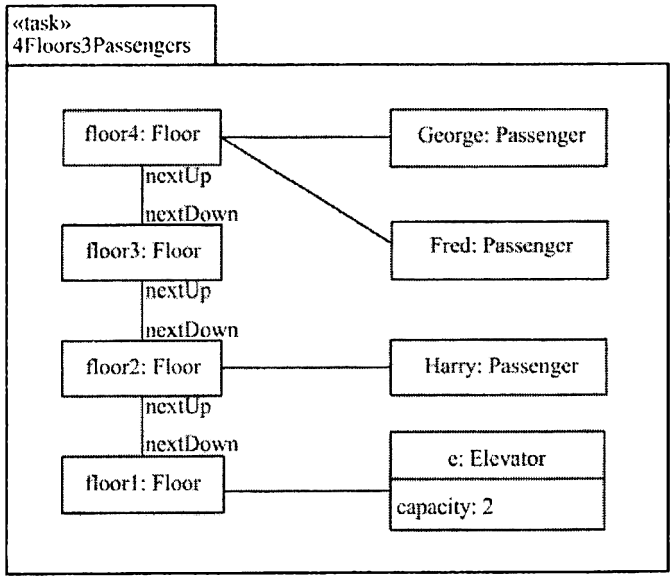


Рис. 3. Диаграмма объектов с начальными условиями задачи

Цель задачи задаётся OCL-выражением, которое должно быть истинным при достижении целевого состояния. Например, выражение:

context 4Floors3Passengers def:

goal = (Fred.at = floor1) and (Harry.at = floor1) and (George. at = floor1)

задаёт цель спустить всех пассажиров на первый этаж.

Можно видеть, что описания предметных областей и задач планирования могут быть составлены на языке UML. Получающиеся модели достаточно наглядны. В тоже время, модели подробны и формальны, так что они могут быть проверены в UML-среде, чтобы выявить расхождения между условиями задачи и описанием предметной области, если таковые имеются. Также средствами Eclipse могут быть проверены OCL-ограничения. Например, можно убедиться, что в начальном состоянии не выполнено условие достижения цели задачи, не выполнено предусловие операции *Elevator::moveDown()*, но выполнено предусловие операции *Elevator::moveUp()* и т. д.

Выше было отмечено, что для использования знаний, представленных в виде UML-модели, программой-планировщиком, их нужно перевести в стандартное представление, т. е. записать на языке PDDL. Это преобразование можно осуществить автоматически с помощью разработанного генератора PDDL-текста по модели.

Для генерации текстов по объектным моделям концерном OMG разработан стандарт MOFM2T [OMG MOFM2T 2008]. Стандарт не ориентирован на какой-либо специальный тип генерируемых документов, например, исходных кодов на языках программирования и может использоваться для генерации спецификаций, а также документов на естественном языке. В основе стандарта лежит подход, согласно которому генерируемый текст формируется на основе шаблонов. При этом в качестве параметров в шаблоны подставляются элементы исходной модели.

В рамках среды Eclipse есть инструментальное средство, базирующееся на стандарте MOFM2T, – Acceleo. Генератор PDDL-описаний реализован нами в виде набора из 19 шаблонов Acceleo. Рассмотрим из них два шаблона, генерирующие описание типов объектов предметной области по исходной диаграмме классов. Первый шаблон *generateTypes* создаёт открывающую и закрывающую скобки и циклически обращается к вспомогательному шаблону *generateType* для получения списка типов. В коде шаблонов курсивом выделены строки, из которых строится итоговый текст. Кроме этих строк в результат включаются имена классов из UML-модели предметной области, обозначенные элементом шаблона *[c.name.toLower()/]*.

```
[template private generateTypes(aPackage : Package) ]
(:types
[for ( elem : Class | (getClasses(aPackage)) )]
        [generateType(elem)/]
[/for]
)
[/template]
[template private generateType( c : Class ) ]
[c.name.toLower()/] –
[if ( c.superClass->size() > 0 )]
        [for ( sc : Class | c.superClass ) separator( ' ')]
                [sc.name.toLower()/]
        [/for]
[else]
        object
[/if]
[/template]
```

В результате работы этих двух шаблонов с UML-моделью задачи о лифте будет получено следующие PDDL-описание:

```
(:types
        passenger - object
        elevator - object
        floor - object)
```

Другие шаблоны в составе генератора PDDL-описаний обеспечивают перевод ассоциаций в предикаты, целочисленных атрибутов классов – во флюенты, операций – в действия. Часть шаблонов генератора переводят диаграмму объектов, описывающую начальные условия экземпляра задачи, и OCL-выражение, задающее цель, в PDDL-текст. Например, для рассмотренной выше задачи будет сгенерировано следующее описание:

```
(define (problem 4Floors3Passengers)
        (:domain elevatordomain)
        (:objects floor1 - floor
                  floor2 - floor
                  floor3 – floor
                  floor4 - floor
                  Fred - passenger
                  George - passenger
                  Harry – passenger
                  e - elevator)
        (:init    (floorToFloor floor3 floor2)
                  (floorToFloor floor2 floor1)
                  (at George floor4)
                  (atFloor e floor1)
                  (at Fred floor4)
                  (at Harry floor2)
                  (floorToFloor floor4 floor3)
                  ( = (elevatorCapacity e) 2))
        (:goal    (and      (at George floor1)
                            (at Fred floor1)
                            (at Harry floor1))))
```

Такого рода описание вместе с PDDL-описанием предметной области может быть подано на вход программе-планировщику. Для рассматриваемой задачи планировщик может предложить план: *((moveUp e) (moveUp e) (moveUp e) (getIn George e) (getIn Fred e) (moveDown e) (moveDown e) (moveDown e) (getOut Fred) (getOut George) (moveUp e) (getIn Harry e) (moveDown e) (getOut Harry))*.

В инженерии знаний систем планирования часто встаёт задача проверить корректность плана, при этом следует убедиться, что, стартуя из начального состояния задачи, можно выполнить действия, предписанные планом, не нарушая их ограничений. Также надо

проверить, что в последнем состоянии выполняется условие достижения цели.

Рассмотрим, как решается задача валидации плана при помощи инструментов программной инженерии. Специализированная среда USE осуществляет верификацию и симуляцию выполнения объектных моделей. Чтобы использовать её в нашей задаче, нужно средство, переводящее описания предметной области, задачи и плана в спецификации среды USE. Таковым средством является генератор спецификаций, реализованный нами на базе Acceleo. Он, также как генератор PDDL-описаний, представляет собой набор шаблонов.

Результатом работы генератора является спецификация, содержащая описания классов, ассоциаций, операций с предусловиями, постусловиями и телами, а также набор команд для создания начального состояния задачи и OCL-запрос, поверяющий, достигнута ли цель.

Приведём фрагменты сгенерированной спецификации:

```
class Elevator
attributes
capacity : Integer
operations
        moveUp()
        begin   declare f : Floor;
                f := self.floor;
                delete(self, self.floor) from atFloor;
                insert(self, f.nextUp) into atFloor;
        end
        pre: self.floor.nextUp->notEmpty()
        post: self.floor = self.floor@pre.nextUp
        moveDown()
        begin   declare f : Floor;
                f := self.floor.nextDown;
                delete(self, self.floor) from atFloor;
                insert(self, f) into atFloor;
        end
        pre: self.floor.nextDown->notEmpty()
        post: self.floor = self.floor@pre.nextDown
  end
```

Первый фрагмент описывает класс Elevator, его атрибут и операции.

```
association atFloor between
        Elevator[*] role elevator
        Floor[1] role floor
end
```

Во втором фрагменте описана ассоциация, задающая отношение между лифтом и этажом, на котором тот находится.

```
!create George : Passenger
!create Fred : Passenger
!create Harry : Passenger
!create Floor1 : Floor
!create Floor2 : Floor
!create Floor3 : Floor
!create Floor4 : Floor
!create e : Elevator
!set e.capacity := 2
!insert(George, Floor4) into at
!insert(Fred, Floor4) into at
!insert(Harry, Floor2) into at
!insert(e, Floor1) into atFloor
!insert(Floor1, Floor2) into floorToFloor
!insert(Floor2, Floor3) into floorToFloor
!insert(Floor3, Floor4) into floorToFloor
```

Третий фрагмент является последовательностью команд среды USE, создающей три экземпляра-пассажира, четыре экземпляра-этажа, объект-лифт и связи между ними, так как они определены в начальных условиях задачи.

```
?George.floor = @Floor1 and Fred.floor = @Floor1 and Harry.floor = @Floor1
```

Четвёртый фрагмент является OCL-запросом, истинным тогда, когда достигнута цель, поставленная в задаче.

```
!e.moveUp()
!e.moveUp()
!e.moveUp()
!George.getIn(e)
!Fred.getIn(e)
!e.moveDown()
!e.moveDown()
!e.moveDown()
!Fred.getOut()
!George.getOut()
!e.moveUp()
!Harry.getIn(e)
!e.moveDown()
!Harry.getOut()
```

Последний фрагмент является последовательностью команд, соответствующей найденному плану. Среда позволяет имитировать поочерёдное выполнение команд, предоставляя возможность наблюдать на диаграмме объектов меняющееся состояние задачи. В ходе

выполнения контролируется соблюдение предусловий и постусловий действий. Есть возможность проверить выполнение условий достижения цели в начальном, финальном или любом промежуточном состоянии.

Финальное состояние задачи, построенное средой после выполнения плана, показано на рисунке 4. Очевидно, что цель достигнута. В ходе выполнения все ограничения, содержащиеся в UML-модели, соблюдались. Таким образом, при помощи среды USE найденный план проверен и признан корректным.

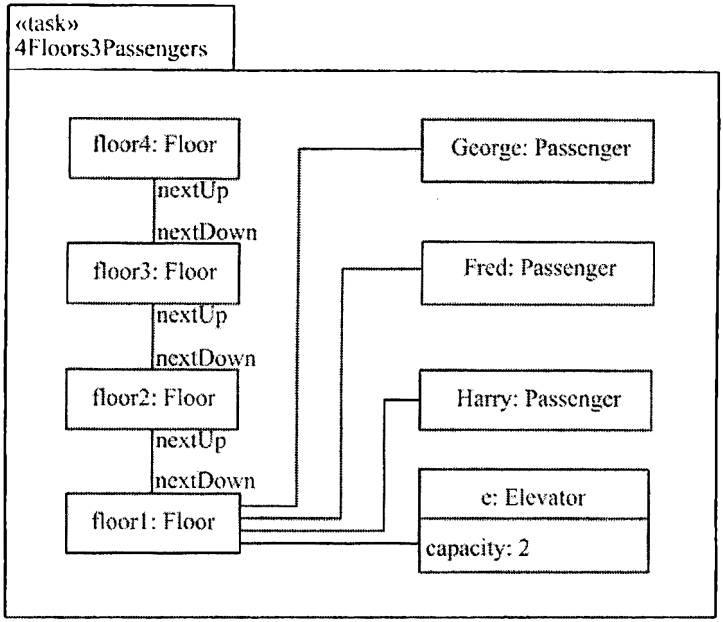


Рис. 4. Диаграмма объектов, визуализирующая финальное состояние задачи

Подведём итоги. Для решения задач инженерии знаний при помощи универсальных средств программной инженерии нами предложен способ представления описаний предметных областей и задач планирования в виде UML-моделей. Определён состав комплекса инструментальных средств. В рамках реализации комплекса созданы генератор PDDL-описаний, подаваемых на вход программы-планировщика, и генератор объектных спецификаций среды USE для визуализации и валидации планов решений, найденных планировщиком. На примере рассмотрены моделирование знаний с помощью UML,

генерация входных данных планировщика, генерация спецификации по плану решения и её исследование, приведшее к выводу о корректности найденного плана.

Литература

1. Арлоу Д., Нейштадт И. UML 2 и унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е издание. СПб.: Символ-Плюс, 2007. 624 с.
2. Warmer J., Kleppe A. The Object Constraint Language: Getting Your Models Ready for MDA, Second Edition. Boston: Addison-Wesley, 2003. p. 240.
3. Vaquero T. S., Tonaco R. et al. itSIMPLE4.0: Enhancing the Modeling Experience of Planning Problems. In: Proceedings of the ICAPS 2012 System Demonstration, São Paulo. 2012. pp. 11-14.
4. Steinberg D., Budinsky F., et al. EMF: Eclipse Modeling Framework, Second Edition. Boston: Addison-Wesley, 2008. 744 p.
5. Gerevini, A., Long, D.: Plan constraints and preferences in PDDL3. Technical Report, Univ. Brescia, Italy, 2005. 12p.
6. Gogolla M., Büttner F., Richters M. USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming, vol. 69, no. 1-3. 2007. pp. 27-34.
7. Object Management Group MOF Model to Text Transformation Language, v 1.0 2008 (http://www.omg.org/spec/MOFM2T/1.0/PDF)

Раздел II
СуперЭВМ и методы параллельного программирования

Бурцев А. П., Курин Е. А.

Исследование различных подходов к разработке параллельных алгоритмов моделирования распространения акустических волн для задач сейсморазведки

В настоящее время поиск полезных ископаемых, таких как нефть и газ, является очень важной задачей. Промышленность и транспорт потребляют всё больше и больше углеводородов, а развитие альтернативных источников энергии, пока не позволяет отказаться от использования традиционных природных ресурсов.

Одним из методов поиска месторождений полезных ископаемых является сейсморазведка. Сейсморазведка — геофизический метод изучения геологических объектов с помощью упругих колебаний — сейсмических волн. В специальных установках (сейсмостанциях) электрические колебания, созданные в сейсмоприемниках очень слабыми колебаниями почвы, усиливаются и автоматически регистрируются на сейсмограммах [4]. Данные собранные одним приёмником за фиксированный период времени называют — трассой. В результате интерпретации сейсмограмм полученных данных можно определить глубины залегания и сейсмогеологические границы различных пород. Моделирования распространения акустических волн в неоднородной среде — одна из важных задач с которой сталкиваются геофизики. Она возникает как на этапе построения модели реальной среды, так и на этапе проверки адекватности построенной модели, полевым экспериментам. Сейчас геофизики собирают сотни гигабайт данных при обследовании даже небольших площадей, а требуется исследовать сотни и сотни квадратных километров [2]. Обработка такого объема полученных данных требует применения мощных компьютеров и больших затрат машинного времени, что часто невозможно в полевых условиях. Ускорение процесса обработки данных даст огромный экономический эффект. Решением проблемы обработки больших объёмов данных может быть использование супер-вычислителей: многоядерных машин, кластеров, грид-систем.

В данной статье рассматриваются три основных комплекса программно-аппаратных средств для разработки параллельных решений: OpenMP для SMP архитектуры, MPI для кластеров и CUDA для графических ускорителей компании NVIDIA [1]. Был разработан параллельный алгоритм моделирования распространения акустических волн в однородной среде. Предположение об однородности среды снижает сложность реализации алгоритма счёта, но на принципиальном уровне ничего не меняет (математические операции фактически останутся теми же, но их число возрастёт), что позволяет делать корректные предположения о эффективности решения данной задачи при добавлении условия неоднородности среды. Цель данной работы — исследование эффективности различных подходов к разработке параллельного алгоритма решения данной задачи.

Распространения акустической волны в однородной среде описывается следующим уравнением [3]:

$$\frac{1}{\rho \cdot v^2}\frac{\partial^2 p}{\partial t^2} = div(\frac{1}{\rho} grad(p)) + S$$

Где:

$\rho(x,z) \equiv const$ — плотность в каждой точке пространства.

$v(x,z,t)$ — скорость акустической волны в каждой точке пространства во все моменты времени.

$p(x,z,t)$ — давление в каждой точке пространства во все моменты времени.

$S(x,z,t)$ — внешняя сила (функция источника) в каждой точке пространства во все моменты времени.

Аппроксимация волнового уравнения проводилась конечно-разностной схемой 2-ого порядка по времени и 4-ого порядка по пространственным координатам. Решалась двухмерная задача по пространственным координатам. В центре расчетной области располагался точечный источник возмущения, на границах предполагалось полное гашение возмущения [2].

Итоговое уравнение приняло вид:

$$U(x,z,t+1) = (\Delta U(x,z,t) - f(x,z,t))v^2 \Delta t^2 + 2U(x,z,t) - U(x,z,t-1)$$

Где:

Δt — шаг по времени, а $f(x,z,t) = r(x,z)w(t)$,

$r(x,z)$ — положения источника,

$w(t)$ — функция излучения источников.

$$\Delta U(x,z,t) = -\frac{30}{12}U(x,z,t)\left(\frac{1}{\Delta x^2}+\frac{1}{\Delta z^2}\right)+$$
$$+\frac{16}{12}(U(x-1,z,t)+U(x+1,z,t))\frac{1}{\Delta x^2}-\frac{1}{12}(U(x-2,z,t)+U(x+2,z,t))\frac{1}{\Delta x^2}+$$
$$+\frac{16}{12}(U(x,z-1,t)+U(x,z+1,t))\frac{1}{\Delta z^2}-\frac{1}{12}(U(x,z-2,t)+U(x,z+2,t))\frac{1}{\Delta z^2}$$

В основе программной реализации лежал последовательный алгоритм предложенный в open-source пакете Madagascar [5].

При реализации программы с использованием OpenMP параллельно выполняется основной цикл расчётов, т. е. вычисление значения функции $U(x,z,t)$ вычисляется независимо для каждой точки (x,z) в рамках одного временного слоя. Это обеспечивает эффективное решение задачи, если все данные помещаются в оперативную память. В качестве платформы был выбран комплекс «regatta» (IBM pSeries 690).

Для использования MPI была выбрана система с распределённой памятью «Ломоносов». Для подобных систем важен способ распределения данных между вычислительными узлами. В данной работе была выбрана простая декомпозиция данных: процессы объединяются в двумерную решётку, каждый процесс обрабатывает свою прямоугольную область данных. Для проведения корректных расчётов после каждого шага по времени необходимо проводить синхронизацию и обмен данными между соседними в решётке процессами: слева, справа, снизу и сверху.

Для GPU было реализовано два CUDA ядра: с использованием Shared памяти и без неё[1]. Процесс копирования данных с GPU на CPU включён в общее время выполнения программы, что позволяет оценить реальную производительность алгоритма. Тесты проводилась на машине Asus N61Vn с видеокартой NVIDIA GeForce GT 240M.

Критериями оценки производительности были: эффективность ($E = T_1 / pT_p$) и ускорение ($S = T_1 / T_p$). Где T_1 — время выполнения последовательного кода, T_p — время выполнения параллельного кода, p — число независимых потоков команд.

Тесты для машины «regatta» (IBM pSeries 690) на OpenMP реализации

Конфигурация: 16xPower4 процессора с 1,3 ГГц , 64 GB памяти.

Размерность (сетка)	Эффективность (4 проц. / 8 проц.)	1 проц.	4 проц.	8 проц.
4801x4801(x, y) 21(t)	0.964 / 0.894	22.302152	5.783290	3.118620
4801x4801(x, y) 201(t)	0.955 / 0.862	227.732160	59.635969	33.038326
8001x8001(x, y) 201(t)	0.969 / 0.880	652.957004	168.377352	92.754869

Табл. 1. Время выполнения на «regatta» (в сек.)

Тесты показывают, что разработанная программа эффективно использует возможности многопроцессорной вычислительной системы. А увеличении размерности задачи практически не меняет показатель эффективности, хотя его изменения связанны с количеством задействованных процессоров (потоков), что может быть объяснено увеличением накладных расходов на их синхронизацию.

Подобная схема вычислений применим в том случае, когда исходная задача целиком помещается в оперативную память. Он позволяет рассчитывать достаточно крупные сетки, что приемлемо для решения задач на сравнительно небольшой площади. Простота реализации делает подобный метод оптимальным для частных компаний, занимающихся обработкой небольших объёмов данных, например строительных (анализ геологии площадки строительства).

Тесты для машины «Ломоносов» на MPI реализации

Конфигурация: процессоры Xeon X5570/X5670 2.93 ГГц с 12 GB памяти.

Размерность (сетка)	Эффективность (16 проц. / 32 проц.)	1 проц.	16 проц.	32 проц.
4801x4801(x, y) 21(t)	0.986 / 0.836	5.572700	0.353259	0.208258
14401x96001(x, y) 21(t)	0.964 / 0.944	33.326932	2.160630	1.103155
14401x96001(x, y) 201(t)	0.977 / 0.966	348.656745	22.306087	11.276295

Табл. 2. Время выполнения на «Ломоносове» (в сек.)

Результаты тестов демонстрируют динамику увеличения (сохранения) эффективности при увеличении размерности задачи и допустимое снижение эффективности при увеличении числа процессоров. Рост эффективности объясняется увеличением времени на счёт, и как следствие повышение его доли в общем времени вычислений. Снижение эффективности обусловлено увеличением времени обмена данными между процессорами. Темпы снижения эффективности показывают, что данный алгоритм хорошо масштабируется, и работает эффективно при условии наличия достаточного объёма данных, обрабатываемых одним узлом. Следует заметить, что все процессоры находились в одной стойке вычислителя, поэтому при увеличении числа стоек может наблюдаться более заметная потеря эффективности.

Данный подход прекрасно подходит для решения «максимальных» задач геологоразведки (задач большой размерности), но для его использования необходимо иметь дорогостоящее оборудование.

Тесты для машины Asus N61Vn
с видеокартой NVIDIA GeForce GT 240M, CUDA реализация

Конфигурация GeForce GT 240M: CUDA Cores – 48, Processor Clock – 550 MHz (Processor) 1210 MHz (Shader), RAM – DDR3 1024 Mb.

Размерность (сетка)	Ускорение (без Shared mem. / с Shared mem.)	«regatta» (1 проц.)	Копир-е	GPU	GPU (Shared)
4801x4801 (x, y) 21 (t)	5.570 / 3.724	22.302152	2.520398	4.003365	5.988465
4801x4801 (x, y) 201 (t)	6.589 / 4.187	227.732160	16.393032	34.564557	54.384238
5001x5001 (x, y) 401 (t)	6.878 / 4.292	505.864659	34.755456	73.543779	117.849737

Табл. 3. Время выполнения на GPU в сравнении с «regatta» (в сек.)

Данные тесты показывают, что даже видеокарта персонального компьютера (ноутбука) способна составлять конкуренцию многопроцессорной системе. А ускорение растёт с увеличением размерности задачи, таким образом при появлении GPU с большим

размером оперативной памяти можно ожидать прирост производительности. Таким образом использование GPU для решения поставленной задачи допустимо, единственной проблемой подобной реализации является небольшой запас оперативной памяти на GPU вычислителе, что позволяет решать задачи сильно ограниченной размерности. Отсутствие выигрыша по времени при использовании Shared памяти обусловлен отсутствием повторных обращений к данным. Значительные затраты на копирование данных между CPU и GPU можно сократить используя параллельное копирование одновременно с расчётами, что должно увеличить ускорение ещё почти в два раза.

Подобная организация вычислений позволяет решать сложные вычислительные задачи за приемлемое время, но из-за малого объёма оперативной памяти на GPU, решение исходной задачи большой размерности на одном GPU устройстве невозможно.

Использование гибридных технологий параллельного программирования MPI вместе с CUDA и/или OpenMP позволяет снять ограничение на размерность задачи, так как данные уже не обязаны целиком помещаться в памяти одного процессорного узла или GPU.

Литература

1. Боресков А. В. Основы работы с технологией CUDA. / А. В. Боресков, А. А. Харламов. - М.: ДМК, 2010. 230 с.
2. Курин Е.А. Презентация компании ООО «GEOLAB» на летней суперкомпьютерной школе МГУ им. Ломоносова: Сейсморазведка и высокопроизводительные вычисления, 2011. http://school.hpc-russia.ru/files/materials/geolab.pdf.
3. Тихонов А.Н. Уравнения математической физики. / А.Н. Тихонов, А.А. Самарский. - 7-е изд. - М.: Изд-во МГУ: Наука, 2004. 742 с.
4. Хмелевской В.К. Геофизические методы исследований. / В.К. Хмелевской, Ю.И. Горбачев, А.В. Калинин, М.Г. Попов, Н.И. Селиверстов, В.А. Шевнин. - Петропавловск-Камчатский: КГПУ, 2004. 232 с.
5. Guide to programming with madagascar. http://reproducibility.org/wiki/Guide_to_programming_with_madagascar.

Гришанин А. А.

Численное моделирование задач для квантовых точек на суперкомпьютере

1. Введение

Квантовые точки – это составляющие гетероструктуру наноразмерные кристаллы полупроводника. То есть квантовая точка представляет из себя полупроводник, находящийся в полупроводнике другого типа (осажденный на нем или окруженный им со всех сторон).

Свободные носители заряда удерживаются в области кристалла за счет потенциальных барьеров, окружающих область квантовой точки по всем трем направлениям. Если размер этой области меньше, чем длина волны электрона, то электронные состояния становятся квантовыми, с дискретными уровнями энергии, как у простого атома. Это дает возможность применять их в различных областях индустрии. В настоящее время с их помощью конструируются фотодиоды, лавинные фотодиоды, фототранзисторы и фоторезисторы, преобразователи инфракрасного излучения в видимое. В 1994 году в физико-техническом институте имени А. Ф. Иоффе были созданы лазеры на основе квантовых точек [2], [3]. Очень перспективно использование квантовых точек для создания солнечных батарей, с высоким КПД [4].

Необходимость численного решения обусловлена потребностью нахождения энергетических уровней и волновых функций электронов в квантовой точке с заданной формой и размером, что позволило бы моделировать квантовую точку с заданными оптическими свойствами. В трехмерном случае, это задача, требующая объемных вычислений, что вызывает необходимость использования суперкомпьютера.

2. Математическая постановка задачи

Обезразмеренное стационарное уравнение Шредингера, запсанное в атомных единицах имеет следующий вид [1]:

$$-\frac{1}{2}\Delta\psi(\vec{r})+U(\vec{r})\psi(\vec{r})=\lambda\psi(\vec{r})$$

где m_e – масса электрона, $\hbar$ – постоянная Планка, $\psi(\vec{r})$ – волновая функция электрона, $U(\vec{r})$ – функция потенциала, λ – энергия электрона.

Причем в случае моделирования квантовой точки функция $U(\vec{r})$ имеет достаточно сложную структуру (например, множество прямоугольных барьеров, имеющих различную высоту в зависимости от материала). Задача нахождения спектра квантовой точки принимает вид задачи на собственные функции $\psi_n(\vec{r})$ и соответствующие им собственные значения E_n.

После введения равномерной сетки, с количеством точек N, M и K по осям x, y и z соответственно, и аппроксимации вторых производных центральными трехточечными схемами:

$$\frac{\psi_{i+1,j,k} - 2\psi_{i,j,k} + \psi_{i-1,j,k}}{2h_1^2} + \frac{\psi_{i,j+1,k} - 2\psi_{i,j,k} + \psi_{i,j-1,k}}{2h_2^2}$$

$$+ \frac{\psi_{i,j,k+1} - 2\psi_{i,j,k} + \psi_{i,j,k-1}}{2h_3^2} - u_{i,j,k}\psi_{i,j,k} = -\lambda\,\psi_{i,j,k}$$

и постановки граничных условий

$\psi_{i,j,k}\big|_{\Gamma} = 0$, где Γ – граница области моделирования,

задача сводится к нахождению собственных значений и собственных векторов матрицы A, порожденной разностным оператором имеющей блочно-диагональную структуру.

Для нахождения собственных значений λ_n использовался метод обратной итерации:

$$\psi^{p+1} = \left(A - \lambda^* I\right)^{-1} \lambda^* \psi^p,$$

где λ^* – параметр, определяющий к какому собственному значению будет сходится метод.

Таким образом, на каждом шаге необходимо решать уравнение с правой частью – вектором, полученным на предыдущей итерации и матрицей $\left(A - \lambda^* I\right)$. Затем полученный вектор нормируется, чтобы не допускать экспоненциального роста функции. В одномерном случае матрица, которую необходимо оборачивать имеет небольшой размер, легко оборачивается, метод легко реализовывается и дает хорошие результаты.

При переходе к задаче с большим количеством измерений вычислительная сложность значительно возрастает (в двухмерном

случае – размер матрицы N*M, в трехмерном – N*M*K), поэтому для решения поставленной задачи было решено использовать суперкомпьютерные вычисления.

3. Алгоритм параллельных вычислений

Так как прямые методы плохо поддаются распараллеливанию, было принято решение использовать итерационные методы, некоторые из которых позволяют в полной мере использовать структуру трехмерного тора суперкомпьютера IBM Blue Gene/P [5]. В данной работе использовались методы Якоби и метод верхней релаксации(«SOR»), с красно-черным разбиением, как наиболее удобные для распараллеливания.

Для метода SOR, с красно-черным разбиением [6] и [7], численная схема в трехмерном случае будет иметь вид:

$$\psi^{l+1}_{i,j,k} = (1-\omega)\psi^{l}_{i,j,k} + \frac{\omega}{2\left(\frac{1}{h_1^2}+\frac{1}{h_2^2}+\frac{1}{h_3^2}-\lambda^{*}+u_{i,j,k}\right)}\times$$

$$\times\left(\frac{1}{h_1^2}\left(\psi^{l}_{i+1,j,k}+\psi^{l}_{i-1,j,k}\right)+\frac{1}{h_2^2}\left(\psi^{l}_{i,j+1,k}+\psi^{l}_{i,j-1,k}\right)\right.$$

$$\left.+\frac{1}{h_2^2}\left(\psi^{l}_{i,j,k+1}+\psi^{l}_{i,j,k-1}\right)-\lambda^{*}\tilde{\psi}_{i,j,k}\right)$$

для точек, у которых остаток от деления суммы индексов по всем измерениям на двойку равен нулю, и

$$\psi^{l+1}_{i,j,k} = (1-\omega)\psi^{l}_{i,j,k} + \frac{\omega}{2\left(\frac{1}{h_1^2}+\frac{1}{h_2^2}+\frac{1}{h_3^2}-\lambda^{*}+u_{i,j,k}\right)}\times$$

$$\times\left(\frac{1}{h_1^2}\left(\psi^{l+1}_{i+1,j,k}+\psi^{l+1}_{i-1,j,k}\right)+\frac{1}{h_2^2}\left(\psi^{l+1}_{i,j+1,k}+\psi^{l+1}_{i,j-1,k}\right)\right.$$

$$\left.+\frac{1}{h_2^2}\left(\psi^{l+1}_{i,j,k+1}+\psi^{l+1}_{i,j,k-1}\right)-\lambda^{*}\tilde{\psi}_{i,j,k}\right)$$

для точек, у которых остаток от деления суммы индексов по всем измерениям на двойку равен единице.

При проведении расчетов на каждом вычислительном узле обрабатывались значения своей части пространства. В двухмерном случае – это прямоугольная область, в трехмерном – прямоугольный параллелепипед. Кроме области, на которой производится расчет, каждый процесс хранит данные в граничных ячейках(теневые грани), расчет которых производит соседний процесс, но значения в которых используются им для нахождения значений в своих ячейках на новом слое. Это схематически изображено на рисунке 1. Распараллеливание осуществлялось с помощью технологии MPI.

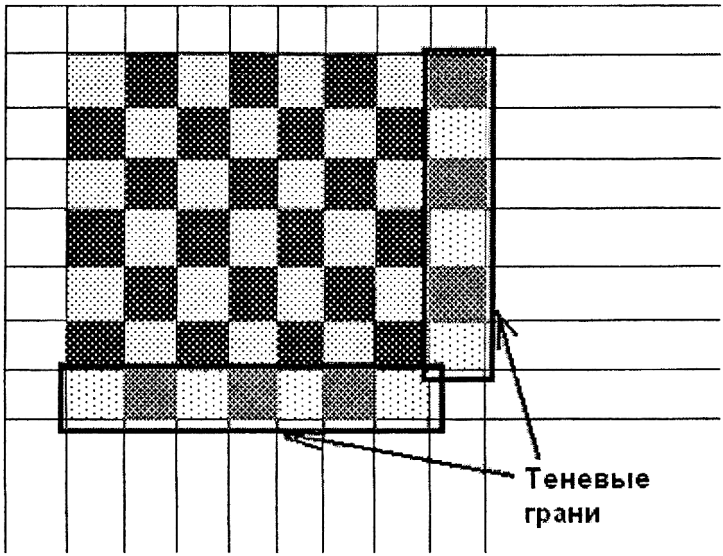


Рис. 1. Изображение данных на одном вычислительном узле

4. Результаты моделирования

В результате численных расчетов были получены собственные функции и собственные числа для потенциальной функции, которая моделирует квантовую точку (в двухмерном случае – квантовую нить, в одномерном – квантовую пластину).

На рисунке 2 показан квадрат волновой функции для функции потенциала, имитирующей квантовую пластину, рассчитанный на сетке с 24 тысячами точек, и параметром $\lambda^* = 9.5$.

Расчеты, проведенные в двухмерном случае, показали некоторые недостатки комбинации итерационных методов с методом обратных итераций. Поэтому для поиска собственных функций

соответствующих высоким собственным значениям приходится ортогонализировать относительно функций соответствующих меньшим собственным значениям.

На рисунках 3 и 4 показаны результаты расчетов для собственных функций в двухмерном случае, эксперимент проводился при N = M = 200.

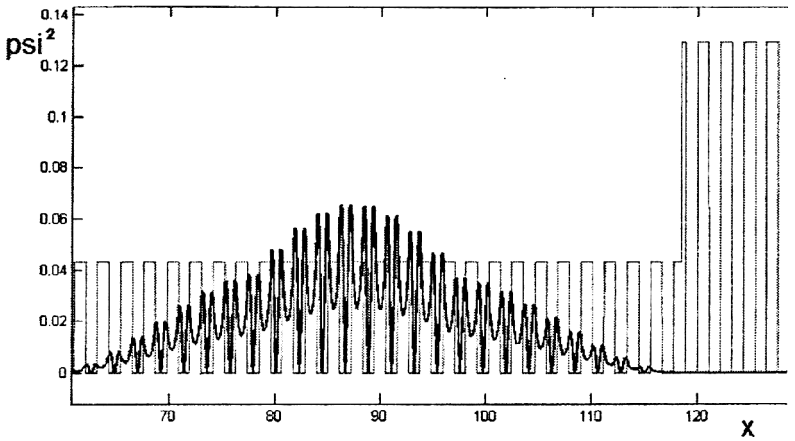


Рис. 2. Изображение квадрата собственной функции (жирной линией) и потенциальной функции (тонкой линией)

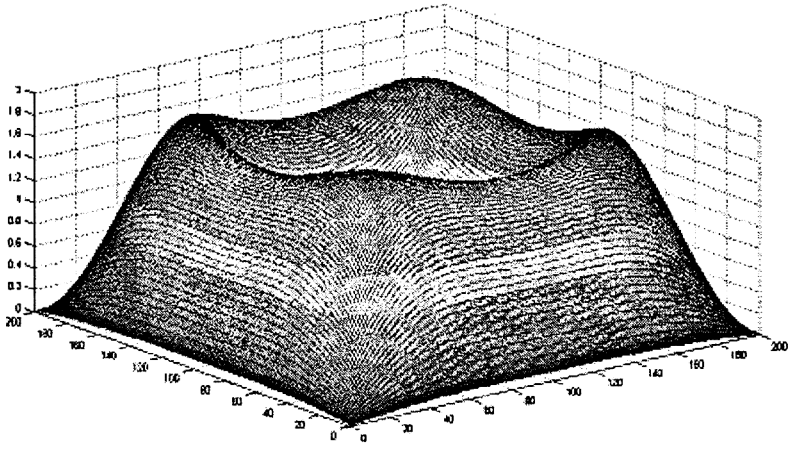


Рис. 3. Волновая функция соответствующая первому собственному значению

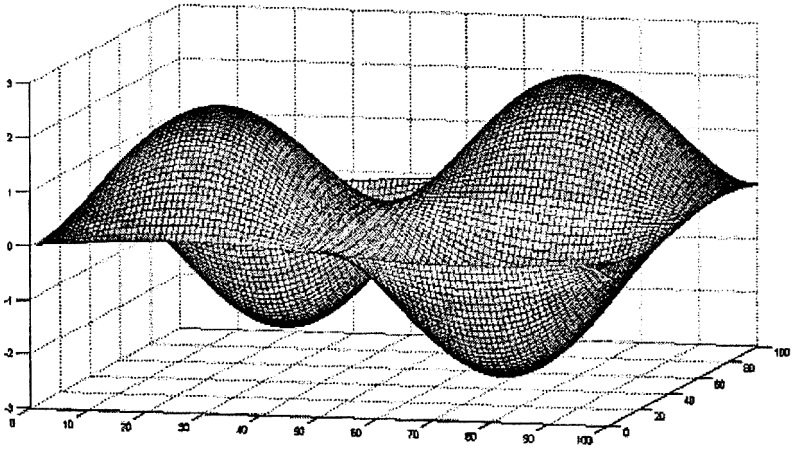


Рис. 4. Волновая функция соответствующая третьему собственному значению

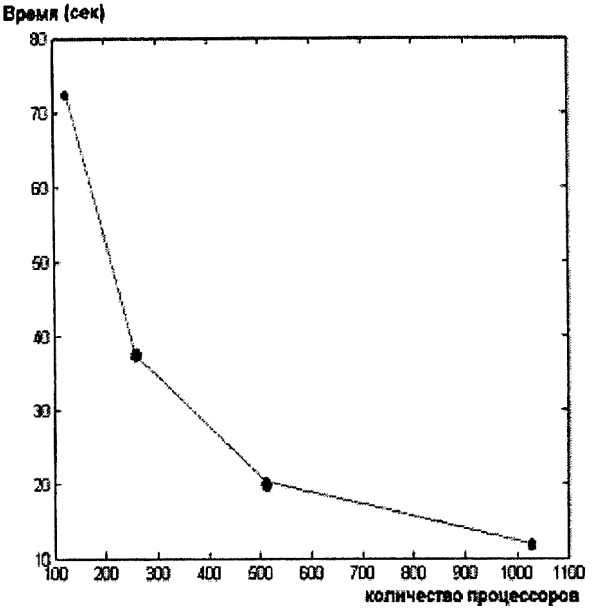


Рис. 5. Зависимость времени от количества процессоров

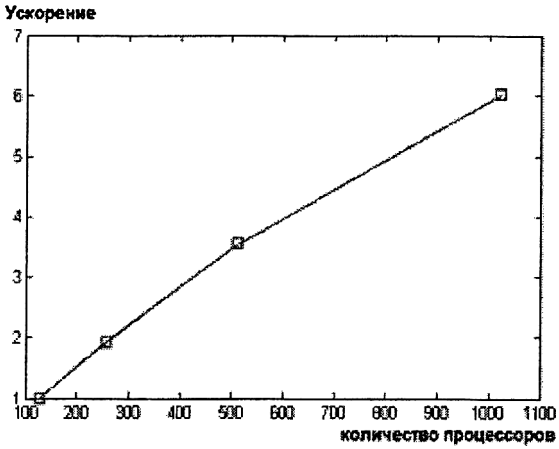


Рис. 6. Зависимость ускорения от количества процессоров

5. Оценки производительности, ускорения. Характеристика параллельного исполнений

Описанный алгоритм решения обладает очень хорошей масштабируемостью, что было проверено испытаниями на суперкомпьютере Blue Gene/P. Это связано в первую очередь с тем, что при указанном разбиении пространства информация, которой обмениваются процессоры относительно невелика. Кроме того за счет автоматического распределения виртуальной топологии достигается максимальная эффективность взаимодействия системы.

Зависимости времени выполнения и ускорения от количества процессоров представлены на рисунках 5, 6.

Выводы

Таким образом, исследование показывает, что имеется возможность численно моделировать квантовые точки, используя супер-ЭВМ. Для улучшения метода планируется протестировать другие итерационные методы, для анализа совместимости их с методом обратной итерации. Возможно, не сильно теряя в масштабируемости вычислений, удастся эффективней искать собственные значения произвольного порядка.

Литература

1. А.М. Попов, Вычислительные нанотехнологии. МАКС Пресс 2009, С. 231.
2. Н. Н. Леденцов, В. М. Устинов, В. А. Щукин, П. С. Копьев, Ж. И. Алферов, Д. Бимберг, Гетероструктуры с квантовыми точками: получение, свойства, лазеры.Физика и техника полупроводников, 1998, том 32, № 4.
3. Ж. И. Алферов, История и будущее полупроводниковых гетероструктур. Физика и техника полупроводников, 1998, том 32, № 1.
4. A.J. Nozik, Quantum dot solar cells. Physica E 14 (2002) 115 – 120.
5. Carlos P Sosa, IBM System Blue Gene Solution: Blue Gene/P Application Development. Draft Document for Review October 19, 2007 (http://www.redbooks.ibm.com/redbooks/pdfs/sg247287.pdf).
6. Н. Н. Попова, Суперкомпьютерные вычислительные технологии, (http://angel.cs.msu.su/~popova/SuperComp2012/Lect%206%20_121012.pdf).
7. D.J. Evans, Parallel S.O.R. iterative methods, Parallel Computing I (1984) 3-18.

Жарков А. В.

Разработка веб-интерфейса для суперкомпьютерного решения задач оптимизации развития транспортной сети

Введение

В данной работе рассматривается создание веб-интерфейса для поддержки суперкомпьютерного решения задач оптимального централизованного управления развитием динамической транспортной сети с однородным непрерывным потоком. Математическая модель такой сети, постановка задачи оптимального развития и параллельный алгоритм её решения подробно описан в [1]. Основной целью решения задачи является выбор оптимальной стратегии развития транспортной сети на протяжении дискретного модельного времени из числа возможных стратегий на основе критерия качества, который характеризует величину удовлетворения спроса потребителей. Параметрами, характеризующими вычислительную сложность решения задачи предложенным алгоритмом, являются $N_1 = N_V N_L^2$, (где N_V – число вершин в графе, N_L - число дуг графа транспортной сети) и $N_2 = 2^Q E$ (Q - число моментов дискретного модельного времени, E – число состояний транспортной сети). Решение задачи для моделей реальных транспортных сетей требует большого количества вычислений, поэтому актуально их решение на суперкомпьютерных системах. Для транспортной сети с $N_1=63\cdot10^9$, $N_2=2^{21}$ время решения задачи последовательным алгоритмом составляет 24 дней. Применение суперкомпьютерного алгоритма позволяет сократить время решения задачи до 1 часа, а создание веб-интерфейса позволит использовать вычислительные мощности суперкомпьютера широкому кругу специалистов в области оптимального управления.

Предметная область и типичные задачи

Под транспортной сетью в данной работе понимается сеть поставок, объединяющая некоторое количество поставщиков, потребителей и распределителей. Поставщики генерируют некоторый поток, под которым, в каждом конкретном случае, можно понимать, например, энергию, энергоносители, товары и т. п. Потоки от поставщиков либо доходит непосредственно до потребителей, либо

достигает распределителей, которые перераспределяют входящие в них потоки по нескольким направлениям, после чего потоки доходят до потребителей. Поток, дошедший до потребителя, поглощается им.

Целью рассматриваемой задачи является оптимизация управления развитием сети поставок от поставщика к потребителю. Сеть представляет собой набор заданных связей между поставщиками, распределителями и потребителями. Под оптимизацией развития сети поставок понимается увеличение пропускных способностей ее связей, с целью максимального удовлетворения спроса потребителей.

Задача рассматривается в предположении, что объемы поставок и спросы потребителей изменяются со временем. Развитие сети поставок должно учитывать этот факт и увеличивать пропускные способностей тех элементов сети, в направлении которых осуществляется и/или будут осуществляться основные поставки. При этом в рассматриваемой постановке задачи не предполагается, что задан единственный сценарий для изменение спроса и объема поставок в будущем. Предполагается, что для каждого потребителя/поставщика задан целый набор возможных сценариев (изменения спроса/предложения в будущем) и вероятность каждого возможного сценария. В условиях описанной неопределенности под решением задачи понимается адаптивная стратегия управления развитием сети поставок, то есть предполагается, что на каждом шаге рассматриваемого процесса наблюдаются текущие объемы поставок от всех поставщиков и текущие спросы всех потребителей и на основе этой информации принимается решение о направлениях дальнейшего развития сети.

Для решения задачи алгоритмом, который предложен в [1], пользователю необходимо подготовить входные данные в бинарном формате, поставить на счёт задачу на суперкомпьютере, дождаться её завершения и выгрузить выходные данные. Для поддержки решения задачи широким кругом специалистов веб-интерфейс должен отвечать следующим требованиям:

1. Наличие базы пользователей, работающих с интерфейсом
2. Наличие ввода для входных данных в удобном для пользователя формате
3. Возможность повторно использовать входные данные, созданные другими пользователями
4. Автоматическая постановка задач на суперкомпьютер
5. Уведомление пользователя о завершении счёта по электронной почте
6. Визуализация результатов на серверной стороне и выгрузка выходных данных

Архитектура веб-интерфейса

Пользователь взаимодействует с интерфейсом с помощью браузера через сеть интернет. Ответы на запросы браузера осуществляются веб-сервером интерфейса. Веб-сервер взаимодействует с двумя хранилищами данных: «File database» и «SQL Database». Хранилище «File database» представляет собой набор файлов в файловой системе, организованный иерархически для хранения входных данных для различных задач, созданных пользователями. Хранилище «SQL Database» построено на основе релятивной базы данных MySQL и хранит информацию о пользователях и задачах. Для постановки задач в очередь, загрузки данных на суперкомпьютер и получения результатов с него используется модуль «SSH connector». Пользователь может создавать, удалять и редактировать задачи оптимального управления, ставить их на счёт и просматривать результаты.

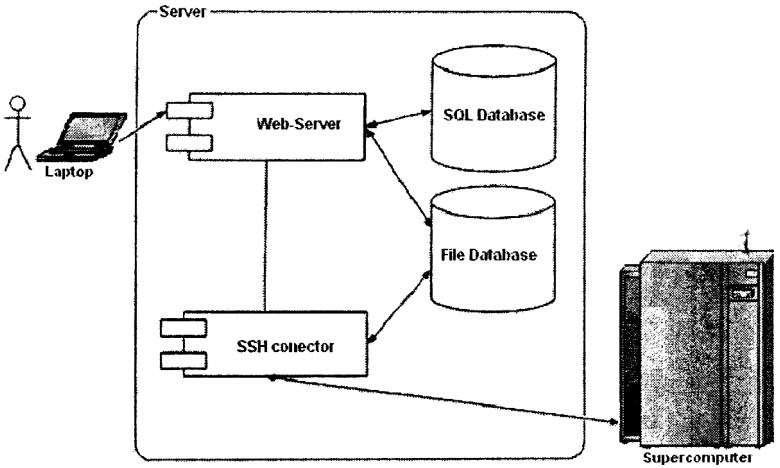


Рис. 1. Схема работы веб-интерфейса программного комплекса

Основные возможности, предоставляемые веб-интерфейсом:

1. Пользователи. Список пользователей, авторизованных для работы в системе. Добавление и удаление пользователей.
2. Графы сетей. База данных графов транспортных сетей, создаваемых пользователями. Редактирование, добавление и удаление графов.

3. Пропускные способности дуг. База данных файлов, в которых хранятся пропускные способности дуг.
4. Матрицы переходных вероятностей. Список матриц, создаваемых пользователями.
5. Наборы дуг. Список возможных наборов оптимальных управлений.
6. Состояния сети. База данных наборов состояний транспортной сети.
7. Задачи. Список задач, созданных пользователями. Добавление, удаление и редактирование задачи. Постановка задач на счёт на суперкомпьютер. Выгрузка результатов.

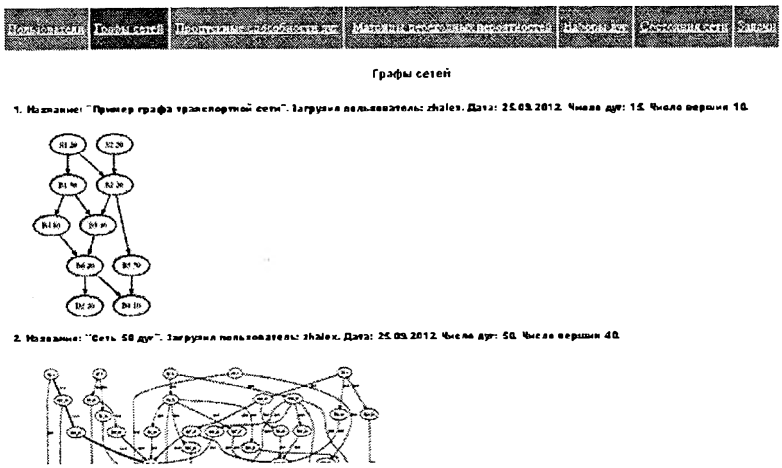


Рис. 2. Страница веб-интерфейса

Заключение

В работе описан программный комплекс с веб-интерфейсом для проведения моделирования развития транспортной сети. Создание веб-интерфейса позволило использовать вычислительные мощности суперкомпьютера широкому кругу специалистов. Программный комплекс был апробирован специалистами кафедры оптимального управления ВМК МГУ и позволяет решать практические прикладные задачи оптимального управления, а так же проводить теоритические исследования формальных моделей. С другой стороны, данная работа имеет теоритическую значимость, расширяя спектр решаемых на суперкомпьютерах задач, который традиционно представлен задачами вычислительной гидро- и газодинамики, механики твердого и жидкого тела, задачами биоинформатики и нанотехнологий.

Работа выполнена при поддержке грантов РФФИ № 11-07-00756 и №11-07-00614.

Литература

1. А. В. Жарков, Д. Г. Пивоварчук. Разработка и исследование параллельного алгоритма решения задачи оптимизации развития инфраструктуры типа поставщик-потребитель // Труды Пятой Международной конференции «Параллельные вычисления и задачи управления» PACO'2010. Институт проблем управления им. В.А. Трапезникова РАН, 2010, С. 433-446.
2. Jack Edmonds, Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM 19 (2), 1972, pp. 248–264.
3. Ravindra K. Ahuja, Murali Kodialam, Ajay K. Mishra, James B. Orlin. Computational investigations of maximum flow algorithms. European Journal of Operational Research 97, 1997, pp. 509-542.

Захаров В. Б., Махнычев В. С.

О решении проблемы шахматных 7-фигурных окончаний на суперкомпьютере «Ломоносов»

Введение

В данной статье рассказывается об этапах решения проблемы 7-фигурных шахматных окончаний. Теоретические основы описываемого решения были изложены в работах [1] и [2]. В статье показано, что вычисления, связанные с задачами большого объема, требуют преодоления многочисленных трудностей и принятия сложных решений на всех этапах выполнения проекта. Недостаточно продуманные решения на любом из этапов могут сделать решение поставленной задачи либо невозможным, либо значительно повысить сложность и соответственно цену вычислений.

Если оценить время, требуемое на реализацию собственно параллельного алгоритма ретроанализа, и время, потраченное на создание программы, реально решающей задачу при заданных ограничениях на компьютерные ресурсы, то соотношение затрат получится более чем на порядок больше.

Мы можем оценить, что время, потраченное на создание программы, реально решающей задачу при заданных ограничениях на компьютерные ресурсы, более чем на порядок превышает время, требуемое на реализацию собственно параллельного алгоритма ретроанализа.

Постановка задачи

Требуется найти оптимальную стратегию при игре в шахматы для множества позиций, в которых на доске находится не более N фигур. При небольших значениях N и достаточных вычислительных ресурсах эта задача может быть решена ретроанализом. Проблемой является лишь необходимость оперирования громадными объемами данных в процессе расчетов и большое количество проводимых вычислений. Задача 6-фигурных шахматных окончаний, имеющая вычислительную сложность порядка 10^{21} операций, была решена в 2006 г. Евгением Налимовым [3]. С ростом N на единицу объем данных в задаче увеличивается более чем в 100 раз, а объем вычислений – в 1000 раз. Поэтому долгое время считалось, что задача 7-фигурных шахматных окончаний не может быть решена ранее 2015 г. В работе [1] был предложен параллельный алгоритм ретронализа с использованием механизма обратных ходов, позволяющий решить задачу 7-фигурных шахматных окончаний с использованием суперкомпьютера.

Решение задачи на суперкомпьютере «Ломоносов»

Описанный в [1] алгоритм ретроанализа был реализован и испытан на суперкомпьютере IBM Blue Gene/P, установленном на факультете вычислительной математики и кибернетики МГУ имени М.В.Ломоносова. В результате в 2009 г. было получено решение для 7-фигурного окончания "король+ферзь+конь против король+ладья+слон+конь" и найден самый длинный из известных матов – мат в 545 ходов.

Но всего существует 875 разных типов 7-фигурных окончаний, которые в сумме требуют для хранения не менее 100 ТБ дискового пространства (с учетом сжатия записываемых на диск данных). Дисковое хранилище комплекса Blue Gene/P на тот момент не позволяло хранить такие объемы данных. Поэтому старт проекта по полному решению проблемы стал возможен только в 2012 г. благодаря получению доступа к суперкомпьютеру «Ломоносов». Все расчеты были проведены в марте-августе 2012 г.

Суперкомпьютер «Ломоносов», входящий в топ-500 самых мощных компьютеров мира, установлен на факультете ВМК МГУ, содержит 78000 ядер Xeon 2.93 Ггц., соединенных быстрой 10 Гбит сетью, его суммарная оперативная память – 100 ТБ, а объем дискового массива – более 350 ТБ. Процессорные ядра суперкомпьютера «Ломоносов» значительно превосходят по вычислительной мощности ядра IBM Blue Gene/P, использовавшегося в первоначальных вычислениях. Это позволило существенно сократить календарное время вычислений.

Вся задача 7-фигурных окончаний была разбита на 3000 подзадач меньшего объема, часть из которых могла решаться параллельно, а часть требовала предварительного завершения других подзадач. Подзадачи требовали для своего решения от 400 до 2600 процессорных ядер. Время вычислений колебалось от пяти минут (для простых окончаний, содержащих несколько фигур одного достоинства) до 3 часов (для сложных пешечных окончаний) непрерывной работы. Чтобы сократить место, занимаемое окончаниями на диске, сразу же после окончания вычислений данные сжимались параллельно в памяти каждого узла и только после этого записывались на диск.

Экспериментальным путем было выявлено, что при использовании задачей более 2000 ядер резко возрастает вероятность отказа сетевого оборудования. Поэтому было принято решение запускать задачи на минимально возможном количестве узлов, обеспечивающем требуемую для решения память. Кроме того, было проведено несколько доработок программы, направленных на снижение интенсивности межпроцессорных обменов; большинство из этих доработок снизили производительность, но зато повысили вероятность

успешного завершения задачи. Несколько подзадач потребовали использования 2500 ядер в течение 2-3 часов, процент успешных запусков для них оказался ниже 50%. Расчет одного из первых сложных окончаний завершался сбоями четыре раза подряд, вызвав легкую панику и лихорадочные действия по поиску и устранению возможной ошибки. Если учесть, что перед запуском на счет задачи стоят в очереди (нередко более суток), то можно понять переживания разработчиков, вызванные этой проблемой.

Балансировка нагрузки

На вычислительные ядра могут попадать данные, требующие разного количества операций по обработке. Поэтому для равномерного распределения вычислений был реализован специальный алгоритм динамической балансировки нагрузки [2]. Применение описанного алгоритма позволило ускорить расчеты на 25-30%. В конечном итоге можно сказать, что удалось достичь очень хорошей степени масштабируемости алгоритма: при использовании до 2000 ядер алгоритм показывает рост эффективности немногим хуже линейной.

Проблемы, возникающие при расчетах большого объема

При обработке большого объема данных неизбежно возникает вопрос обеспечения корректности и сохранности данных. Например, при записи многих терабайт данных на диск даже при очень надежной аппаратуре возникают единичные ошибки. Для контроля целостности каждый двухмегабайтный блок данных снабжался контрольной суммой, вычисленной по алгоритму MD5. Таким образом становится возможным проверять правильность файла при дальнейших операциях с ним. Данный метод контроля многократно оправдал себя. Многоуровневая система буферизации записи на диск суперкомпьютера, с одной стороны, обеспечивает высокую скорость записи, но с другой стороны – нередко приводит к порче данных. При этом ошибка записи иногда не выявляется, если данные используются сразу же после сохранения, так как при этом они берутся из промежуточной кэш-памяти, а не с диска. Поэтому проверка целостности данных через несколько дней после сохранения стала обязательной процедурой в ходе проекта. Также проверка целостности осуществляется при копировании данных с носителя на носитель.

Нелегкой проблемой является тестирование алгоритма генерации. На данный момент проверка была осуществлена несколькими методами. Во-первых, было осуществлено полное сравнение всех 6-фигурных таблиц, полученных с помощью нового алгоритма, с 6-фигурными таблицами, рассчитанными в 2006 г. Во-вторых, большое число шахматных задач было решено другими

методами, и результаты сравнивались с результатами из новых таблиц. Успешные результаты многочисленных тестов дают основание говорить о высокой степени доверия к полученным данным.

Однако хотя на данный момент и не известно о наличии хотя бы одной ошибки в 7-фигурных таблицах, можно допустить, что при обработке такого объема данных даже при безошибочном алгоритме генерации всё-таки могли произойти незамеченные сбои, например, ошибки при пересылке данных в оперативной памяти. Поэтому запланированы написание и прогон программы для полной верификации таблиц.

8-фигурные окончания?

После решения задачи 7-фигурных окончаний возникает вопрос, а когда может быть решена в 1000 раз более сложная задача 8-фигурных окончаний. Сейчас технически возможно рассчитать простейшие 8-фигурные окончания с большим количеством одинаковым фигур. Но простейшие окончания не имеют большого практического смысла. При значительных усилиях можно рассчитать и некоторые практически значимые 8-фигурные окончания, но затраты вычислительной мощности при этом будут очень велики: десятки тысяч ядер и дни обсчета, а каждое подобное окончание потребует для хранения целый 4-терабайтный жесткий диск. В этой связи начинать в ближайшие год-два генерацию 8-фигурных окончаний не представляется целесообразным.

Использование 7-фигурных таблиц

На данный момент предполагается, что доступ шахматистов к таблицам 7-фигурных окончаний будет осуществляться через сервер. Соответствующий сервер и сетевой API на момент написания статьи находится в стадии тестирования энтузиастами.

В самом начале тестирования было выявлено, что используемое в файлах таблиц разбиение на блоки размером 2 МБ, позволившее эффективно сжимать таблицы при генерации, не позволяет обеспечить приемлемую скорость выдачи веток-решений, требующей неупорядоченного доступа к произвольным байтам файлов. Нахождение веток состоит в проверке всех возможных ходов из текущей позиции и выборе лучшего из них, то есть, происходит обращение к разным элементам таблицы. Очень часто эти элементы лежат вне пределов 2-мегабайтных блоков, которыми происходит чтение. И хотя блоки кэшируются в оперативной памяти, разброс данных оказывается достаточно большим. Следовательно, большинство данных в 2-мегабайтных блоках читаются и распаковываются напрасно. Для решения проблемы, во-первых, были проведены исследования,

позволившие переупорядочить элементы и увеличить вероятность расположения данных из близких позиций ближе к друг к другу, а во-вторых, было принято решение о переходе к блокам меньшего размера — 8 КБ. Выбор правильного размера блока оказался нелегкой задачей. С одной стороны, чем меньше размер блока, тем эффективнее работает механизм кэширования дисковых данных, с другой стороны, при маленьких блоках резко падает степень сжатия данных, на 40% и более. Если учесть, что 40% от всех таблиц - это 40 ТБ данных (для их хранения требуется 10 наиболее емких – 4 ТБ – из доступных на сегодняшний день винчестеров), то можно понять, что решение о переходе на 8-килобайтные блоки далось нелегко.

Значительное время было потрачено на сравнение разных алгоритмов сжатия. Победителем по большинству параметров оказался алгоритм LZMA. Основным его недостатком является лишь сравнительно медленная распаковка. По скорости распаковки особо хочется отметить результаты LZ4 и LZHAM, но первый, к сожалению, заметно уступает LZMA по степени сжатия, а второй все еще находится в разработке.

Заключение

Целью данной статьи было не теоретическое исследование алгоритма ретроанализа, а рассказ о некоторых практических шагах, которые было необходимо выполнить, чтобы завершить большую работу по генерации 7-фигурных окончаний. Хочется обратить внимание, что для достижения результата помимо собственно разработки алгоритма ретроанализа потребовались значительные оптимизации многих программных компонентов, и эта работа оказалась многократно большей и не менее сложной, чем концептуальная реализация алгоритма.

Литература

1. Захаров В.Б., Махнычев В.С. Алгоритм ретроанализа в суперкомпьютерных системах на примере задачи игры в шахматы // В сб. Программные системы и инструменты. № 11, М.: Изд.отделения ф-та ВМК МГУ, 2010. С. 45–52.
2. Захаров В.Б., Махнычев В.С. Распараллеливание алгоритма ретроанализа для задач сверхбольшой размерности. Получение точного решения для 7-фигурных шахматных окончаний // В сб. Отчет по использованию суперкомпьютеров МГУ в 2009—2010 гг. М., МГУ, 2010. С. 20—22.
3. E.V. Nalimov, G.M. Haworth, E.A. Heinz. Space-efficient indexing of endgame tables for chess. // Advances in Computer Games 9 (2001) — Стр. 93—113.

Корж О. В.

Анализ масштабируемости методов системы визуализации для суперкомпьютера «Ломоносов»

Введение

Использование вычислительных систем петафлопсного уровня в современной научной деятельности актуально и необходимо. Не менее актуальным является разработка высокоуровневых средств программирования для такого рода вычислительных систем, а так же проектирование программного обеспечения для систем экзафлопсного диапазона. Использование систем визуализации больших данных при вычислительном эксперименте на суперкомпьютерах позволяет существенно сократить объем передаваемых по сели данных при анализе результатов вычислительного эксперимента. Для систем петафлопсного уровня вычислительной мощности проблема анализа полученных результат становится особенно острой, т.к. в существующих коммуникационных сетях не возможна полноценная передача данных на локальную машину пользователя. В данной статье были проанализированы основные особенности вычислительных систем такого уровня с точки зрения разработки для них систем визуализации больших данных. Основную сложность представляет собой организация передачи данных между вычислительными узлами для построения итогового изображения. Проанализированы возможности использования различных стратегий компоновки изображения для построения итоговой картинки.

Было проведено тестирование разработанных ранее методов для системы визуализации[1-3] с точки зрения возможности их масштабируемости при вычислительном эксперименте на нескольких тысячах ядер. Исследование показало необходимость разработки специализированных методов параллельной визуализации в условиях системы петафлопсного уровня, т.к. существующие решения не обладают достаточной масштабируемостью.

1. Функциональная схема работы системы визуализации для суперкомпьютера «Ломоносов»

Аппаратная архитектура системы с одной стороны требует больше затрат на организацию работы с данными, но с другой стороны возможно использование преимуществ работы с системой. В частности возможно использованием GPU процессоров для построения изображения по данным с одного блейда. Также возможно использование специализированного сервера визуализации. Таким

образом архитектура системы визуализации, разработанной ранее для архитектуры BlueGene /P [4-5], может быть дополнена следующим образом. Возможно использовать два подхода: визуализация на графических процессорах на одном с вычислительным узлом блейде и визуализация на специализированном сервере визуализации через интерфейсную машину с сохранением данных на внешний носитель (в случае системы «Ломоносов» – параллельное хранилище данных). Функциональная схема такой системы показана на Рис.1.

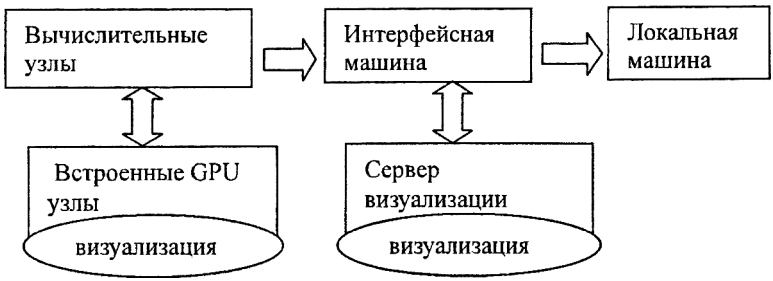


Рис. 1. Функциональная схема работы системы визуализации для суперкомпьютера «Ломоносов»

2. Оценка масштабируемости методов

Далее приводятся результаты тестирования масштабируемости основных методов, используемых в системе визуализации разработанной ранее для суперкомпьютера BlueGene /P [4-5], при запуске этих методов на суперкомпьютере «Ломоносов».

На Рис.2 Показана зависимость времени (в секундах) работы алгоритма построения изображения от размера исходной решетки данных и количества процессорных ядер, которые для этого использовались. Можно видеть, что предел масштабируемости для этого алгоритма составляет около 250 ядер. Далее эффективность работы алгоритма существенно падает.

Следующим проанализированным методом является метод компоновки изображения. Использовалась компоновка методом циклического сдвига[6]. Особенность этого метода в том, что для построения итогового изображения по распределенным данным требуется существенное количество коммуникаций. Поэтому метод может использоваться для оценки эффективности работы коммуникационной сети. Время работы метода показано на Рис.3. Можно заметить нестабильность работы метода, обусловленную реализацией сети для коллективных операций передачи сообщений.

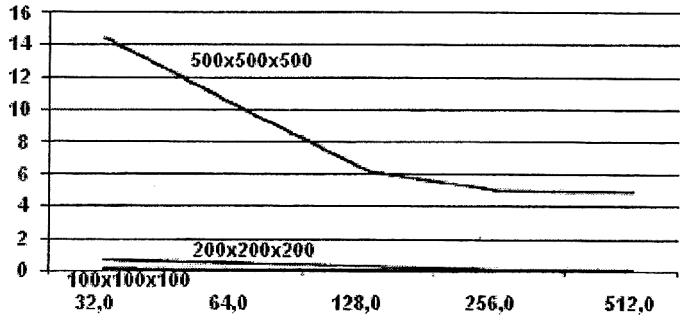


Рис. 2. Время работы рендеринга изображения

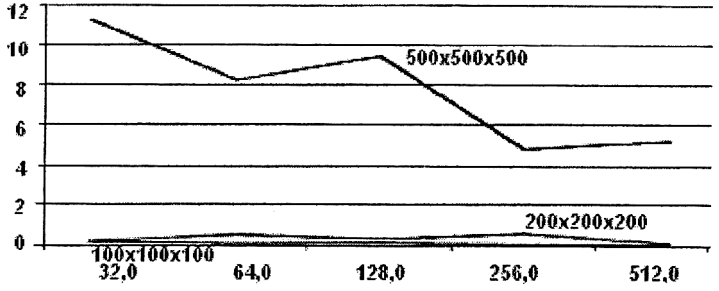


Рис. 3 Время работы алгоритма компоновки изображения

Далее был проанализирован метод сжатия данных для ускорения передачи изображений по сети[3]. Результат работы этого метода показан на Рис.4. Полученный график показывает, что эффективность использования достигается на больших размерах изображений. Однако на значении 256 ядер получен предел масштабируемости.

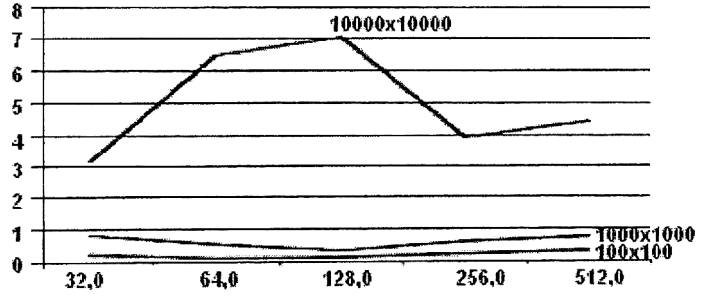


Рис. 4 Время работы алгоритма сжатия изображений

Заключение

Таким образом, проведенный анализ позволяет сделать вывод, что применение существующих методов построения, компоновки и сжатия изображений в системах петафлопсного уровня не позволяет использовать преимущества вычислителя. Алгоритмы имеют низкую масштабируемость и требуют существенной оптимизации с точки зрения использования архитектурных особенностей построения коммуникационной сети и механизмом ввода вывода данных.

Статья подготовлена при поддержке РФФИ: гранты 11-07-00614, 11-07-00756, 12-07-31229, 12-07-00778.

Литература

1. Корж О.В., Киселев К.Н., Параллельный метод визуализации в динамике выполнения вычислительного эксперимента на суперкомпьютере // Труды международной конференции Научный сервис в сети Интернет, Россия, Новороссийск, –М.: Изд-во МГУ, 2011, сс.235-243.
2. Андреев Д.Ю., Джосан О.В., Анализ эффективности масштабируемых подходов к решению задач с преобладанием ввода-вывода // Труды международной научной конференции Параллельные вычислительные технологии (ПаВТ'2011), Москва, Россия, 2011, сс. 387–395.
3. Джосан О.В., Оценка сложности стратегий параллельного построения изображений для систем визуализации на суперкомпьютерах // Труды международной научной конференции Параллельные вычислительные технологии (ПаВТ'2011), Москва, Россия, 2011, сс. 137–145.
4. Oxana Dzhosan, Nina Popova, Anton Korzh, Hierarchical Visualisation System for High Performance Computing // Advances in Parallel Computing, Volume 19, 2010, IOS Press, pp.177- 184, ISBN 978-1-60750-529-7, DOI: 10.3233/978-1-60750-530-3-177.
5. Джосан О.В., Попова Н.Н., Параллельный метод сжатия изображений для визуализации данных на массивно-параллельных вычислительных системах // Труды международной научной конференции Параллельные вычислительные технологии (ПаВТ'2010), Уфа, Россия, 2010, сс. 469–477, ISBN 978-5-696-03987-9.
6. Джосан О.В., О сложности стратегий параллельного построения изображений для систем визуализации на суперкомпьютерах // Жур. Вестник ЮУрГУ. Компьютерные технологии, управление, радиоэлектроника, No 23 (240), вып. 14, 2011, изд. центр ЮУрГУ, сс.87-97. ISSN 2071-0216.

Лихогруд Н. Н., Микушин Д. Н.

KernelGen – прототип распараллеливающего компилятора C/Fortran для GPU NVIDIA на основе технологий LLVM

Аннотация

Проект KernelGen имеет цель создать на основе современных открытых технологий компилятор Fortran и C для автоматического портирования приложений на GPU без модификации их исходного кода. Анализ параллелизма в KernelGen основан на инфраструктуре LLVM/Polly и CLooG, модифицированном для генерации GPU-ядер и alias-анализе времени исполнения. PTX-ассемблер для GPU NVIDIA генерируется с помощью бекенда NVPTX. Благодаря интеграции LLVM-части с GCC с помощью плагина DragonEgg и модифицированного линковщика, KernelGen способен при полной совместимости с компилятором GCC генерировать исполняемые модули, содержащие одновременно CPU- и GPU-варианты машинного кода. В сравнительных тестах с компилятором PGI/OpenACC KernelGen демонстрирует большую гибкость по ряду возможностей, обеспечивая при этом сравнимый или незначительно более высокий уровень производительности.

1. Введение

Массовое использование GPU в кластерных вычислительных системах требует массовой адаптации множества сложных приложений. Программная модель CUDA достаточно хорошо подходит для небольших программ с ярко выраженным вычислительным ядром. Однако для сложных приложений, состоящих из множества отдельных блоков, таких как математические модели, сложность настройки взаимодействия оригинального кода и кода для GPU многократно возрастает. Для того чтобы упростить процесс портирования на GPU сложных приложений, развивается ряд специализированных программных моделей:

1. Директивные расширения существующих языков высокого уровня с ручным управлением параллелизмом, по аналогии с OpenMP. Для стандартизации набора директив основными разработчиками подобных коммерческих технологий создан консорциум OpenACC [6]. Существуют аналогичные открытые системы F2C-ACC [7] и KernelGen версии 0.1 [1] для преобразования исходного кода на языке Fortran в аналогичную гибридную форму, но они не могут автоматически оценивать параллельность переносимых на GPU

вычислительных циклов.

2. Специализированные языки (domain-specific languages, DSL), со встроенными средствами параллелизма, ориентированные на определённый класс задач. В последние годы было предложено множество различных DSL- и Embedded DSL-языков со встроенными средствами параллелизма. В частности, в работе [2] предложен Си-подобный DSL-язык для выражения вычислений на сетках с учетом начальных и граничных условий. В работе [5] аналогичная задача решается при помощи eDSL, основанного на шаблонах C++. Однако неясно, насколько существенным может быть выигрыш в эффективности DSL-языков по сравнению с директивными расширениями. Кроме того специализация ограничивает конкурентную среду, так как у каждого языка как правило существует только один разработчик. Глубокое сравнительное тестирование затруднено необходимостью реализации бенчмарков на каждом используемом языке.

3. Автоматический анализ параллельности кода с помощью эвристик или методов выпуклой оптимизации. В работе [8] для компилятора GCC реализовано расширение для автоматической идентификации параллельных циклов и генерации для них кода на OpenCL. Аналогичное расширение PPCG для компилятора Clang (LLVM) [9] способно преобразовывать код на C/C++ в CUDA-ядра. Обе системы при помощи Chunky Loop Generator (CLooG) [10] преобразуют вычислительные циклы из внутреннего представления компилятора в код на OpenCL или CUDA.

Явное программирование на CUDA, директивные расширения и DSL-языки в любом случае предполагают модификацию или переработку исходного кода программы. По этой причине портирование больших приложений на GPU с помощью этих технологий по-прежнему сильно затруднено. Если же приложение портировано лишь частично, то синхронизация данных между хостом и GPU может значительно влиять на общую производительность. Так, при портировании только одного блока WSM5 модели WRF с помощью директив PGI Accelerator, время обменов данными составляет 40-60% общего времени [18].

1. 1. Цели проекта

Настоящая работа является значительной переработкой существующей системы компиляции KernelGen v0.1 [1]. На данном этапе были поставлены и решены следующие задачи:

1. Минимизировать обмен данными между памятью системы и GPU за счёт переноса всего кода на GPU. Во всех существующих в настоящее время системах компиляции разметка исходного кода определяет отдельные участки для выполнения на GPU, что делает неизбежным явный обмен данными. Более того, и в

интегрированных системах типа AMD Fusion (если их поддержка будет включена в OpenACC), где память CPU и GPU используют один и тот же физический носитель, адресные пространства логически разделены и требуют копирования данных. Данная разработка впервые использует противоположную технику: по умолчанию на GPU переносится *весь* код приложения, предполагая, что даже последовательные части кода может быть более выгодно исполнять на GPU, чем обмениваться данными и выполнять их на CPU. Тем не менее, полностью перенести весь код на GPU невозможно, например, из-за наличия системных вызовов или операций ввода-вывода. Для этого случая необходимо предусмотреть режим обратного вызова, при котором исполнение GPU-ядра останавливается до завершения хост-вызова. Таким образом, элементы привычной модели «основная хост-система и переферийный графический процессор» меняются местами: «центральный графический процессор – переферийная хост-система» (рис. 1).

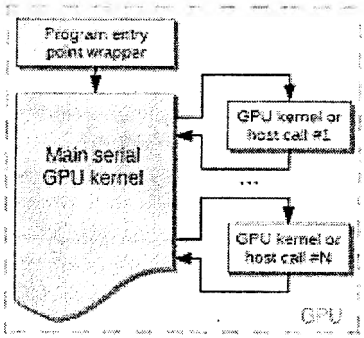


Рис. 1. Схема исполнения «центральный графический процессор – переферийная хост-система»

2. Обеспечить поддержку широкого множества языков программирования. Компилятор CUDA использует специальный препроцессор *cudafe* для разделения исходного файла, комбинирующего C++-код хоста и GPU-код ядер, в два отдельных модуля компиляции. KernelGen версии 0.1 использовал аналогичную технику для исходных файлов на языке Fortran, выделение циклов в GPU-ядра производилось с помощью XSLT-преобразований кода, размеченного в XML (pretty-printed AST). Тем не менее, непосредственное преобразование кода на высокоуровневом языке довольно сложно и часто приводит к тому, что разработчикам удаётся обеспечить надёжную поддержку только подмножества языка. Гораздо более простым решением является работа с кодом на уровне внутреннего представления компилятора (IR - intermediate representation), к которому обычно сводятся все языки, поддерживаемые компилятором. KernelGen версии 0.2 использует LLVM

[13] - инфраструктуру компилятора с простым внутренним представлением LLVM IR SSA и DragonEgg [14] - плагин GCC, генерирующий LLVM IR для программы на любом из языков, поддерживаемых GCC. Схема этапов генерации кода компилятора KernelGen показана на рис. 2.

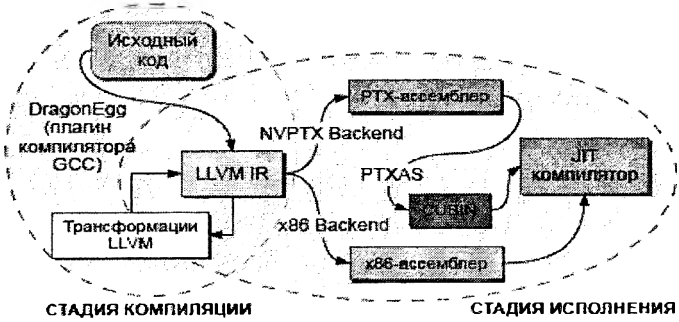


Рис. 2. Схема этапов генерации кода компилятора KernelGen

3. Производить автоматическую оценку параллельности и преобразование вычислительных циклов. Для определения параллельных участков кода задействован CLooG, но не напрямую, а посредством LLVM Polly [3], который встраивает выпуклый анализ циклов в структуру оптимизирующих преобразований (passes) LLVM. В настоящее время LLVM Polly поддерживает генерацию эффективного кода только для CPU и OpenMP. В данной работе он был изменён таким образом, чтобы распараллеливать код с учётом специфики GPU: отображение многомерных параллельных циклов на многомерную вычислительную решётку и запуск ядер средствами CUDA API. Так как известных на этапе компиляции свойств кода часто бывает недостаточно для однозначного определения параллельности (например, слабый alias analysis), KernelGen производит заключительный шаг анализа и генерацию GPU-ядер во время исполнения, привлекая дополнительную информацию, известную из контекста вызова.

При разработке гибридных систем компиляции также необходимо предусмотреть возможность применения к кластерным приложениям с множеством GPU, которые уже распараллелены по вычислительным элементам. Если используется многопроцессное распределение (например, MPI), то необходимо переключение GPU, если многопоточное (OpenMP, POSIX Thread), то также необходима потоковая безопасность (thread-safety) управляющей системы (runtime-библиотеки). В следующих разделах настоящей статьи дана характеристика этапов преобразования исходного кода и генератора параллельных циклов.

2. Этапы преобразования кода

KernelGen работает напрямую с оригинальным приложением, не требуя каких-либо изменений ни в исходном коде, ни в системе сборки. KernelGen включает плагин для фронтенда незначительно модифицированной версии GCC и поэтому полностью совместим с ним. Для наилучшей поддержки больших приложений эти свойства достаточно важны и часто недооцениваются. Чтобы обеспечить обычный процесс сборки, в KernelGen используется схема, напоминающая {LTO} (link time optimization – инфраструктура компилятора для дополнительной оптимизации кода во время линковки): код для GPU сначала добавляется в отдельную секцию объектных файлов, затем объединяется и снова разделяется на отдельные ядра на этапе линковки. Окончательная компиляция GPU-ядер в ассемблер происходит при необходимости, уже во время работы приложения (JIT, just-in-time compilation).

В результате работы компилятора, исходное приложение преобразуется во множество GPU-ядер: одно или несколько *основных* ядер и множество *вычислительных* ядер. Основные ядра исполняются на GPU в одном потоке. Их задача – хранить данные, исполнять небольшие последовательные участки кода и производить вызовы вычислительных ядер и отдельных CPU-функций, которые невозможно или неэффективно переносить на GPU (рис. 1). Вычислительные ядра исполняются на GPU множеством параллельных нитей с полной загрузкой мультипроцессоров. Таким образом, максимальная доля кода выполняется на GPU, а CPU лишь координирует исполнение. В частности, при работе MPI-приложения каждый рабочий процесс в данном случае будет представлять собой GPU-ядро с небольшим числом CPU-вызовов MPI. Использование MPI дополнительно облегчается за счёт поддержки GPU-адресов в командах обмена данными [16].

2.1. Компиляция

При компиляции отдельных объектов, генерируется как x86-ассемблер (таким образом, приложение по-прежнему работоспособно при отсутствии GPU), так и представление LLVM IR. Для разбора исходного кода используется компилятор GCC, чьё внутренее представление *gimple* преобразуется в LLVM IR с помощью плагина DragonEgg. Затем в IR-коде производится выделение тел циклов в отдельные функции, вызываемые через универсальный интерфейс вида

```
__device__ int kernelgen_launch(
        unsigned char* name, unsigned long long szargs,
        unsigned long long szargsi, unsigned int* args);
```

где name – имя или адрес функции (вместо имён в начале работы программы подставляются адреса), arg – структура, агрегирующая

аргументы вызова, szarg и szargi – размер списка аргументов и списка целочисленных аргументов (последний используется для вычисления хеша функции и поиска ранее скомпилированных ядер во время исполнения).

Стандартный механизм выделения каскадов вложенных циклов в функции LLVM\linebreak *LoopExtractor* расширен, так чтобы цикл не заменялся, а дополнялся вызовом функции по условию:

```
if (kernelgen_launch(name, szargs, szargsi, args) == -1) {
        // Launch original loop.
}
```

Таким образом, *kernelgen_launch* запускает параллельное ядро только при выполнении определённых условий, в противном случае (например, если в цикле слишком мало итераций или присутствуют вызовы CPU-функций) оригинальный цикл выполняется последовательно в основном GPU-ядре.

С помощью данного условия runtime-библиотека KernelGen может переключать выполнение между различными версиями цикла. Например, если цикл определён как непараллельный, то *kernelgen_launch* возвращает -1, и код цикла начинает выполняться основным ядром в последовательном режиме. Тем не менее, данный цикл может содержать вложенные параллельные циклы, обработка которых будет проведена аналогичным образом. В конце концов, если весь каскад тесно вложенных циклов непараллелен, несовместим (например, содержит вызовы внешних CPU-функций) или оценен как неэффективный для GPU, то вся функция выгружается для работы на хосте с помощью вызова *kernelgen_hostcall*:

```
__device__ void kernelgen_hostcall(
        unsigned char* name, unsigned long long szargs,
        unsigned long long szargsi, unsigned int* args);
```

при котором GPU-приложение останавливает свою работу и передаёт данные и адрес функции для выполнения на CPU. Функции *kernelgen_launch* и *kernelgen_hostcall* работают в GPU-ядре и вызывают остановку его выполнения. После завершения работы другого ядра или CPU-функции, основное ядро продолжает работу. Хост-часть управляющих функций компилирует и выполняет заданную функцию с помощью интерфейса FFI (Foreign Function Interface).

Одним из специфических свойств KernelGen является хранение всех данных приложения в памяти GPU. Для того чтобы обеспечить его совместимость с наличием CPU-вызовов, реализована простая система синхронизации памяти. При попытке CPU-функции обратиться к памяти по адресу из диапазона GPU возникающий сигнал сегментации обрабатывается дублированием страниц из памяти GPU в страницы CPU-памяти, расположенные по тем же адресам. После завершения

работы CPU-функции, изменённые CPU-страницы синхронизируют изменения с памятью GPU.

2.2. Линковка

Во время линковки отдельных объектов в приложение или библиотеку, LLVM IR также линкуется в один общий IR-модуль для main-ядра и по одному IR-модулю на каждый вычислительный цикл. IR-код погружается в исполняемый файл и в дальнейшем оптимизируется и компилируется в GPU код по мере необходимости во время работы приложения.

Специальной обработки требуют глобальные переменные и выделение стековой памяти в GPU-ядрах. Так как линковка GPU-ядер в текущей версии не поддерживается, каждое ядро имело бы собственную компию всех используемых глобальных переменных, и для корректной работы приложения требовалась бы их синхронизация. Вместо этого, во всех вычислительных ядрах использование глобальных переменных заменяется их адресами, передаваемыми через список аргументов функции. Таким образом становится возможно расположить единственную копию глобальной переменной в модуле main-ядра (что, тем не менее, некорректно в случае нескольких точек входа). Другой проблемой является стековая память. В GPU отсутствует автоматический стек. В некоторых случаях, таких как рекурсия и нехватка регистров, заменой стеку служит локальная переменная-буфер. Инструкции *alloca* бекендом не обрабатываются, поэтому на этапе линковки для каждого ядра они заменяются одной общей глобальной переменной. Альтернативным вариантом могла бы быть замена *alloca* на *device-malloc*, однако это вызывает deadlock-и.

2.3. Модель исполнения

Основное ядро запускается в самом начале выполнения приложения и работает на GPU постоянно. Во время работы вычислительного ядра или CPU-функции основное ядро переходит в состояние активного ожидания и продолжает работу после завершения внешнего вызова. Для реализации данной схемы GPU должно поддерживать одновременное исполнение нескольких ядер (concurrent kernel execution) или временную выгрузку активного ядра (kernel preemption). Одновременное исполнение ядер доступно в GPU NVIDIA, начиная с Compute Capability 2.0, в GPU AMD такой возможности нет, но есть вероятность появления kernel preemption в одной из следующих версий OpenCL. По этой причине в данный момент KernelGen работает только с CUDA.

Вызовы *kernelgen_launch* и *kernelgen_hostcall* состоят из двух частей: device-функции на GPU и одноимённого вызова в CPU-коде, который выполняет, соответственно, окончательную генерацию кода и

запуск вычислительного ядра или загрузку данных с GPU и запуск CPU-функции средствами Foreign Function Interface (FFI). Взаимодействие между частями может быть организовано посредством глобальной памяти GPU или pinned-памяти хоста. Однако для гарантированной передачи корректного значения необходимо обеспечить *атомарный* режим операций чтения и записи, доступность которого является определяющим фактором. По этой причине был реализован метод, использующий глобальную память.

На GPU Kepler K20 запуск ранее скомпилированых ядер может быть проведён без взимодействия с CPU, непосредственно из основного ядра, с помощью технологии динамического параллелизма.

Дополнительное препятствие взаимодействию GPU-ядра с другим ядром или CPU состоит в том, что данные нити (CUDA thread) хранятся в регистрах или локальной памяти. Это означает, что аргументы, переданные из основного GPU-ядра не могут быть использованы где-либо, кроме как в нём самом. Для преодоления этого ограничения, бекенд NVPTX изменён так, чтобы локальные переменные помещаются не в *.local*-секцию, а в *.global*, делая их доступными всем GPU-ядрам и хосту.

3. Генерация CUDA-ядер для параллельных циклов с помощью Polly

Polly [3] (от polyhedral analysis – выпуклый анализ) – это оптимизирующее преобразование циклов, основанное на CLooG и инфраструктуре LLVM. Оно способно распознавать параллельные циклы в IR-коде, оптимизировать кеширование за счёт добавления блочности, оптимизировать доступ к памяти за счёт перестановки циклов и генерировать код, использующий OpenMP. Для заданного кода CLooG строит *абстрактное синтаксическое дерево* (AST), проводя расщепление циклов по некоторым измерениям. Благодаря возможности расщепления частично-параллельных измерений, для исходного цикла может быть найдено эквивалентное представление из одного или нескольких циклов, часть которых параллельна. Подобный подход используется довольно редко, большинство современных компиляторов ограничиваются проверкой параллельности измерений существующих циклов без глубокого анализа.

Элементами AST являются *статические части потока управления* (static control parts – SCoPs) – части программы, в которых поток управления и шаблоны доступа к памяти могут быть вычислены во время компиляции. Часть программы представляет собой SCoP при выполнении следующих условий:

1. Единственными операторами управления являются циклы-счётчики (for) и условные операторы.

2. Каждый цикл-счётчик имеет только одну целочисленнную индексную переменную, которая изменяется в теле цикла с константным шагом. Верхняя и нижняя границы цикла заданы афинными выражениями, зависящими от параметров и индексных переменных внешних циклов, где под параметрами понимаются любые целочисленные переменные, не изменяющиеся внутри SCoP.

3. Условные операторы сравнивают только значения двух афинных выражений.

4. Помимо управляющих конструкций присутствуют только операторы присваивания, выражения и индексный доступ к массивам. Выражениями могут быть операторы или вызовы функций без побочных эффектов, аргументами которых являются параметры, индексные переменные или элементы массивов.

5. Обращения к массивам проводятся по индексам, являющимися афинными выражениями от параметров и переменных.

При генерации LLVM IR высокоуровневые конструкции исходных языков преобразуются в последовательности низкоуровневых инструкций. Так, циклы описываются условными переходами, формирующими эквивалентный поток управления, обращения к массивам выражаются адресной арифметикой, афинные выражения расщепляются на последовательности трёхадресных инструкций. Для восстановления высокоуровневой информации из промежуточного представления Polly использует средства LLVM. Преимуществом такого подхода является возможность извлечения высокоуровневой информации из низкоуровневой, даже если она присутствует в высокоуровневом коде программы лишь неявно. Другими словами, Polly может оптимизировать не только циклы, присутствующие в явном виде, но и любой код семантически эквивалентный циклам, например программы на языке Fortran, использующие goto.

В KernelGen из выделенных на этапе компиляции функций с тесно-вложенными циклами средствами LLVM/Polly выбираются параллельные по одному и более измерениям. С помощью класса *polly::clastExpGen* для каждого цикла генерируется код расчёта числа итераций, используемый при задании вычислительной сетки ядра. К исходным циклам также применяются стандартные упрощающие преобразования LLVM и пространственно-временные оптимизации Polly. Данные этапы могут быть применены как на этапе компиляции, так и во время исполнения приложения. В первом варианте можно сразу же провести оптимизацию и отбросить циклы, определённые как непараллельные. В то же время, второй вариант позволяет выполнить дополнительную подстановку значений скалярных переменных из контекста исполнения, что в некоторых случаях улучшает качество анализа за счёт того, что неафинные выражения становятся афинными.

В Polly имеется генератор кода для OpenMP. Если внешний цикл является параллельным, то его содержимое перемещается в отдельную функцию, с добавлением вызовов функций библиотеки libgomp – GNU реализации OpenMP. При этом распределение итераций по ядрам производит среда выполнения и распараллеливается только внешний цикл. Адаптация этого метода для генерации GPU-ядер потребовала следующих изменений:

1. Отображение пространства итераций на нити GPU, с учётом необходимости объединения запросов в память нитей варпа (coalescing transaction).

2. Рекурсивная обработка вложенных циклов с целью использования возможностей GPU по созданию многомерных сеток нитей.

Пусть в заданной группе циклов можно распараллелить N тесно-вложенных циклов. Тогда ядро может быть запущено на решётке с числом измерений N (для CUDA $N \leq 3$). Для каждого измерения, распределяемого между нитями GPU, генерируется код, рассчитывающий положение нити в блоке и блока в сетке. Каждому параллельному циклу ставится во взаимно однозначное соответствие измерение решетки, причём в обратном порядке -- внутреннему циклу соответствует измерение X (это позволяет объединять запросы в память). Для каждого параллельного цикла генерируется код, определяющий нижнюю и верхнюю границы части пространства итераций, которая должна быть выполнена нитью. Затем генерируется последовательный код цикла с изменёнными границами и шагом.

4. Дополнительные средства времени исполнения

Некоторые типы функций CUDA API, такие как выделение GPU-памяти и загрузка другого CUDA-модуля, всегда приводят к неявной синхронизации асинхронных операций. Поскольку схема работы KernelGen требует постоянного поддержания основного ядра в состоянии выполнения, необходимость выделения памяти или загрузки новых ядер в процессе его работы приведёт к deadlock-у. В этом отношении существующие версии CUDA создают для развития KernelGen определённые препятствия, вынуждая реализовывать нестандартные эквиваленты базовой функциональности.

Если синхронность выделения памяти на GPU со стороны хоста ещё можно считать разумным ограничением, то подтверждённая экспериментами синхронность вызовов malloc внутри GPU-ядер явно избыточна, так как память для индивидуальных потоков выделяется заранее. Так как оба стандартных варианта не могут быть использованы, KerneGen преаллоцирует собственный пул памяти и управляет его работой.

JIT-компиляция вычислительных ядер предполагает, что вновь скомпилированные GPU-ядра будут загружаться на GPU в фоне работающего основного ядра. Обычно динамическую загрузку ядер можно произвести с помощью стандартных функций *cuModuleLoad* и *cuModuleGetFunction* CUDA Driver API. Однако обе эти функции являются синхронными, предположительно, из-за неявного выделения памяти для хранения кода и статических данных. В данной ситуации при разработке KernelGen не оставалось иного выбора, кроме как реализовать загрузчик кода новых ядер вручную, предварительно создав для них пустую функцию-контейнер. Загрузчик основан на технологиях проекта AsFermi [4] и действует следующим образом. В начале работы приложение на GPU загружается достаточно больше пустое ядро (содержащее инструкции NOP). По мере того, как в процессе работы приложения требуется запускать вновь скомпилированные ядра, их код копируется как данные в адресное пространство контейнера, которое известно благодаря инструкции LEPC (получить значение Effective Program Counter). Контейнер размещает код множества небольших ядер друг за другом, создавая своеобразный динамический пул памяти для кода. При этом необходмо учитывать, что различные ядра могут использовать различное число регистров. Для этого загрузчик создаёт 63 фиктивных ядра (точки входа), использующих от 1 до 63 регистров с единственной инструкцией JMP для перехода по адресу начала требуемого ядра в контейнере.

Система синхронизации памяти между GPU и хостом использует вызов *mmap*, ограничивающий возможные диапазоны адресов величинами, кратными размеру страниц (4096 байт). Поэтому выравнивание всех данных GPU по границе 4096 было бы очень удобным упрощением на данном этапе. К сожалению, текущая реализация CUDA (5.0) учитывает настройки выравнивания данных при компиляции, но при этом игнорирует их во время исполнения (подтверждённый баг). Обход этого дефетка реализован посредством выравнивания размеров всех данных по границе 4096 вручную с помощью функций библиотеки *libelf*.

5. Тестирование

KernelGen тестируется на трех типах приложений: тесты корректности, тесты производительности и работа на реальных приложениях. Тесты корректности предназначены для контроля регрессивных изменений в генераторе кода, тесты производительности позволяют анализировать эффективность текущей версии KernelGen в сравнении с предыдущими версиями и другими компиляторами. На рис. 3-4 представлены результаты сравнительного тестирования ядер перемножения матриц и метода Якоби на GPU Fermi (C2070) и Kepler

(GTX680M), скомпилированных с помощью KernelGen и PGI 12.8. Производительность KernelGen на перемножении матриц на порядок хуже, поскольку пока не реализована поддержка использования разделяемой памяти.

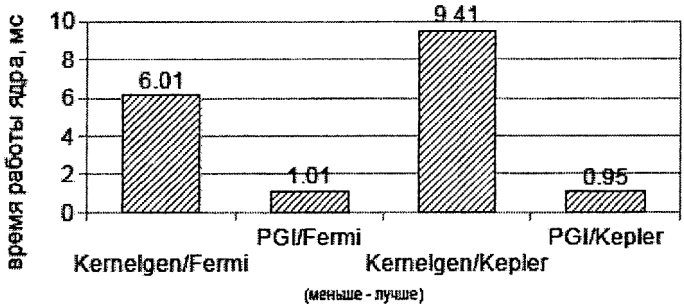


Рис. 3. Сравнение производительности KernelGen и PGI OpenACC на вычислительном ядре произведения матриц

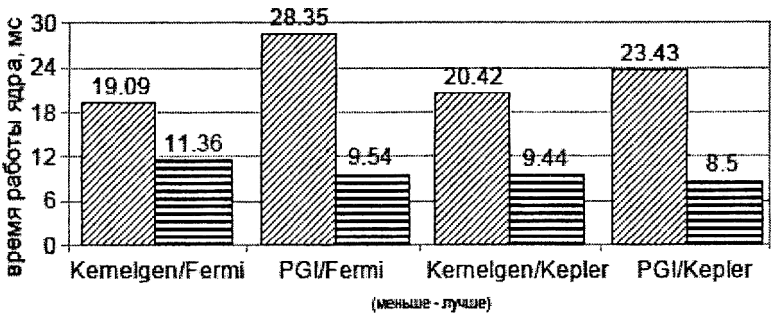


Рис. 4. Сравнение производительности KernelGen и PGI OpenACC на методе Якоби (диагональная штриховка – время ядра вычислений, горизонтальная штриховка – время ядра копирования данных)

При тестировании KernelGen и PGI на методе Якоби, компиляторы показывают сравнимые времена. Тест на методе Якоби примечателен тем, что KernelGen автоматически распознаёт наличие вложенных параллельных циклов внутри непараллельного цикла по числу итераций, когда как PGI работает в соответствии с расставленными вручную директивами.

Тестирование на больших вычислительных приложениях COSMO [17] и WRF [18] показало, что KernelGen способен генерировать корректные исполняемые файлы с поддержкой GPU за разумное время.

6. Заключение

В проекте KernelGen сделана попытка реализовать оригинальную схему автоматического портирования кода на GPU, подходящую для сложных приложений. Не требуя никаких изменений в исходном коде, компилятор переносит на GPU максимально возможную часть кода, включая выделение памяти, тем самым создавая эффективную схему для преимущественно GPU-вычислений. KernelGen реализует средства автоматического анализа параллелизма циклов, основанные на LLVM, Polly и других проектах, расширяя их поддержкой генерации кода для GPU. Генератор GPU-кода основан на NVPTX-бекенде для LLVM, совместно развиваемом компанией NVIDIA и силами сообщества LLVM.

Для того чтобы начать использовать компилятор в прикладных задачах, остается реализовать некоторые функциональные элементы. В частности, в нынешней версии отсутствует механизм оценки эффективности GPU-кода и накопления статистики запусков ядер для более эффективного переключения между CPU- и GPU- реализациями. В генераторе параллельных циклов желательно добавить возможность использования разделяемой памяти и распознавание в коде идиомы редукции. Запуск вычислительных ядер на архитектуре Kepler может быть организован более эффективно за счёт использования динамического параллелизма.

Код KernelGen распространяется по лицензии BSD (за исключением плагина для GCC) и доступен на сайте проекта (http://kernelgen.org).

Литература

1. D. Mikushin, "KernelGen – naïve GPU kernels generation from Fortran source code," COSMO General Meeting, Presentation, (https://hpcforge.org/scm/viewvc.php/*checkout*/doc/cosmo_gm11/cosmo_gm11.pdf?revision=447&root=kernelgen), September 2011, Rome.
2. M. Christen, "Generating and Auto-Tuning Parallel Stencil Codes," Ph.D. dissertation, University of Basel, Switzerland, 2011.
3. T. Grosser, H. Zheng, R. Aloor, A. Simb¨urger, A. Gr¨o_linger, L.-N. Pouchet, "Polly – Polyhedral Optimization in LLVM," IMPACT 2011 (at CGO 2011), Charmonix France, April 2011.
4. Y. Hou et al, "AsFermi: An assembler for the NVIDIA Fermi Instruction Set," http://code.google.com/p/asfermi/ accessed May 2012.
5. T. Gysi, "HP2C Dycore," Presentation, http://mail.cosmo-model.org/pipermail/pompa/attachments/20120306/079fadc1/DWD_HP2C_Dycore_120305.pdf, Workshop on COSMO dynamical core rewrite and HP2C project, March 2012, Offenbach, Germany.

6. The OpenACCTMApplication Programming Interface. Version 1.0, November, 2011, http://www.openacc-standard.org.
7. M. Govett, "Development and Use of a Fortran ! CUDA translator to run a NOAA Global Weather Model on a GPU cluster," Path to Petascale: Adapting GEO/CHEM/ASTRO Applications for Accelerators and Accelerator Clusters, Presentation, http://gladiator.ncsa.uiuc.edu/PDFs/accelerators/day2/session3/govett.pdf, April 2009, National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign.
8. A. Kravets, A. Monakov, A. Belevantsev, "GRAPHITE-OpenCL: Automatic parallelization of some loops in polyhedra representation," GCC Developers' Summit, October 2010, Ottawa, Canada.
9. S. Verdoolaege et al, "PPCG – C to CUDA processor," http://repo.or.cz/w/ppcg.git accessed May 2012.
10. C. Bastoul, "Code Generation in the Polyhedral Model Is Easier Than You Think," PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques, September, 2004, Juan-les-Pins, France.
11. M. Murphy, "NVIDIA's Experience with Open64," Open64 Workshop at CGO 2008, Presentation, http://www.capsl.udel.edu/conferences/open64/2008/Papers/101.doc, April 6, 2008, Boston, Massachusetts.
12. The Open64 compiler, http://www.open64.net/documentation/publications.html accessed May 2012.
13. C. Lattner, "LLVM: An Infrastructure for Multi-Stage Optimization," Masters Thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, December 2002.
14. D. Sands, "Reimplementing llvm-gcc as a gcc plugin," Third Annual LLVM Developers' Meeting, Presentation, http://llvm.org/devmtg/2009-10/Sands_LLVMGCCPlugin.pdf, October 2009, Apple Inc. Campus, Cupertino, California.
15. M. Wolfe, C. Toepfer, "The PGI Accelerator Programming Model on NVIDIA GPUs Part 3: Porting WRF," http://www.pgroup.com/lit/articles/insider/v1n3a1.htm accessed May 2012.
16. J. Squyres, G. Bosilca, S. Sumimoto, R. vandeVaart, "Open MPI State of the Union," Open MPI Community Meeting, Presentation, http://www.open-mpi.org/papers/sc-2011/Open-MPI-SC11-BOF-1up.pdf, Supercomputing 2011.
17. Consortium for Small-scale Modeling, http://www.cosmo-model.org/ accessed May 2012.
18. The Weather Research & Forecasting Model, http://www.wrf-model.org/index.php accessed May 2012.

Раздел III
Обработка текстовой информации

Полякова И. Н., Крутов В. А.

Разрешение неоднозначности в функциональной омонимии

Одной из серьезных проблем, которые необходимо решать в рамках широкого круга систем, включающих автоматическую обработку текстов на естественном языке, является проблема автоматического разрешения многозначности, то есть выбора между разными значениями слов и словосочетаний, перечисленных в лингвистическом ресурсе. В частности, в поисковых системах можно повысить точность обработки некоторых классов запросов и/или сократить объем хранимой информации.

В последние годы проблема разрешения многозначности (омонимии) стала исследоваться как отдельная задача. Подходы к ее разрешению достаточно разнообразны. Так, могут использоваться некоторые внешние источники информации, например, электронные словари и тезаурусы. Кроме того, для разрешения омонимии активно исследуется возможность применения методов машинного обучения, для чего обычно используются семантически размеченные корпуса. Применяются и различные комбинации отдельных методов.

Актуальность проблемы определяется тем, что практически все существующие алгоритмы снятия омонимии включаются в состав синтаксического анализа, что создает трудноразрешимое противоречие, когда для успешного снятия омонимии необходимы точные результаты синтаксического анализа, для получения которых, в свою очередь, нужно предварительно снять омонимию.

Предлагаемый в данной работе алгоритм позволяет разрешать проблему омонимии до начала работы синтаксического анализатора - на предсинтаксическом этапе.

В лингвистической литературе пока нет единства взглядов на явление омонимии. Наиболее общая классификация подразделяет омонимы на лексические, т.е. принадлежащие одной части речи, и грамматические, т.е. принадлежащие различным частям речи. В литературе известен термин "функциональная омонимия". Этот термин предложен О.С. Ахмановой [1]. Функциональные омонимы определяют как "слова, совпадающие по звучанию, этимологически родственные, относящиеся к разным частям речи". В работе Кобзаревой Т.Ю. [2] приведена классификация 58 типов функциональных омонимов. При

этом наиболее частотными являются первые десять типов (общее число омонимов - 2965), для 26 типов выделено менее пяти представителей каждого типа.

Результат прикладных исследований в компьютерной лингвистике во многом зависит от наличия соответствующих лингвистических ресурсов. Описания функциональных омонимов даны в специальном словаре омонимов О.М. Ким [3], содержащем около 5000 омонимических рядов, который также был использован в данной работе.

Постановка задачи

Предлагаемая работа посвящена задаче автоматического разрешения функциональной омонимии двух наиболее частотных и сложных для разрешения типов:

1) N*/A*, где N*={существительные}, A*={A-полные прилагательные}; Количество выделенных омонимов 379 (оценка Кобзаревой Т.Ю.) .

Примеры:

*У него был **больной** вид.* (Омоним ***больной*** разрешается как ***A****).*

*По закону **больной** холерой подлежит немедленной изоляции.* (Омоним ***больной*** разрешается как ***N****).*

2) N*/ Vf, где N*-существительные, Vf - формы глаголов. Количество выделенных омонимов 878 (оценка Кобзаревой Т.Ю.).

Примеры:

*Делами всеми **правил** приказчик.* (Омоним ***правил*** разрешается как Vf).

*Важно строгое соблюдение **правил**.* (Омоним ***правил*** разрешается как ***N****).

В рамках данной работы требовалось разработать алгоритм разрешения омонимии выбранных типов. Для этого было необходимо:

1. Классифицировать и рассмотреть имеющиеся методы снятия омонимии. Выбрать метод для решения данной задачи.
2. Проанализировать и адаптировать работы лингвистов в данной области для решения поставленной задачи.
3. Выполнить программную реализация, показывающую достаточно высокую эффективность.

Явление омонимии в русском языке

Разными авторами достаточно подробно исследованы различные виды омонимии, однако непротиворечивая и полная классификация ее видов до сих пор отсутствует.

Так, например, Ахманова О.С. в [1] рассматривает такие виды функциональной омонимии как омонимия прилагательного и существительного, «склоняющегося как прилагательное»; омонимия наречия, оканчивающегося на «–о» и краткой формы прилагательного среднего рода , но при этом не рассматривает вопросы о том, какие части речи могут быть омонимами, а также о частичной омонимии существительного и глагола.

Наиболее полная классификация видов омонимии представлена в [4]. По характеру омонимических языковых единиц выделяется:

- омонимия морфем;
- омонимия слов;
- омонимия словосочетаний;
- омонимия разнотипных единиц

По степени полноты омонимии соответственно выделяется:

- полная омонимия
- частичная омонимия: существительное *печь* только в одной из своих словоформ омонимично одной словоформе глагола *печь*.
- неравнообъемная омонимия (полная в отношении одного из омонимов и частичная в отношении другого: слово *лук* (растение) омонимичен *луку* (оружие), но не наоборот, так как *лук* (оружие) образует формы множественного числа, а *лук* (растение) – нет.

В большинстве методов разрешения многозначности используются внешние источники информации (словари и тезаурусы). Поэтому встает вопрос о фиксации и представлении омонимов в словарях. Какие типы омонимов фиксируются словарями?

Естественно, что в словарях отражается только омонимия слов. Естественно, что словари фиксируют только омографические омонимы. Наконец, исходя из формы слова, принимаемой в качестве «представительной» (инфинитив глагола, существительное в именительном падеже единственного числа), словари регистрируют только те слова-омонимы, которые являются омонимами в своих «представительных» формах, только полные и лишь некоторые частичные и неравнообъемные слова-омонимы. Так, существительное *печь* и глагол *печь* попадают в число омонимов. Существительное *стих* и глагол *стих* представляют собой неравнообъемные омонимы, но часто не рассматриваются в словарях как омонимы, поскольку инфинитив глагола (стихать) здесь не затронут. Все это тоже приходилось учитывать при работе над поставленной задачей.

Методы снятия омонимии

Существующие в настоящий момент в компьютерной лингвистике методы снятия омонимии можно подразделить на:

1. методы, основанные на контекстных правилах, составляемых экспертами-лингвистами;
2. методы, основанные на контекстных правилах, выводимых из текстов (с управляемым и с неуправляемым обучением);
3. методы, основанные на вероятностных моделях.

Для решения поставленной задачи применен метод, основанный на контекстных правилах.

Одним из методов разрешения функциональной омонимии для русского языка является метод синтаксического анализа ближайшего контекста омонима. В основе метода контекстного анализа лежат эвристические правила, которые учитывают наиболее типичные для русского языка контекстные ситуации для каждого типа функциональных омонимов. [5]. Очевидно, что точность метода напрямую зависит от разнообразия разрешающих контекстов, а также типа омонима, который определяется на основе приписанной омониму совокупности грамматических характеристик.

Метод контекстного разрешения функциональной омонимии (далее - метод КРФО) сводится к разработке для каждого функционального типа омонимии группы правил, задающих синтаксический контекст разрешения омонима, и построение управляющей структуры группы, определяющей порядок применения правил.

Подход, основанный на правилах, является чрезвычайно трудоемким, требует проведения тщательной лингвистической экспертизы каждого типа омонимии и включает разработку лингвистических и вычислительных моделей следующих задач:

Лексикографические задачи включают уточнение набора грамматических характеристик функциональных омонимов русского языка и построение классификации типов функциональных омонимов. Задача уточнения набора грамматических характеристик функциональных омонимов возникла при разработке метода КРФО и связана с недостатками существующих лексикографических описаний (грамматических словарей омонимов), на которых базируется метод. Явление омонимии в русском языке описано в различных словарях.

Сопоставительный анализ словарей выявил проблему словарного описания явления функциональной омонимии [6]. Прежде всего, различия связаны с описанием сложных случаев функциональной омонимии, например, таких, как «краткое прилагательное/ наречие». В словаре О.М.Ким [3] их 560. Грамматические характеристики слов из этого списка сравнивались с характеристиками по словарям

Аношкиной, Зализняка и Национального корпуса русского языка. Оказалось, что только в трех случаях из 560 все четыре словаря приписали омонимам одинаковые характеристики.

Решение задачи уточнения набора грамматических характеристик функциональных омонимов было получено на основе сопоставления и анализа лексикографических источников и последующего уточнения выявленных характеристик на материале корпусных данных. В результате в [2] были уточнены количественные характеристики функциональной омонимии русского языка (оценка числа типов и количественный состав типов/подтипов омонимии). Словарь типов функциональных омонимов был представлен в виде сводной таблицы, содержащей следующие поля:

- тип функциональной омонимии, заданный списком частей речи;
- текстовая расшифровка имени типа;
- упорядоченные списки омонимов каждого типа, распределенные по соответствующим словарям.

Дополнительно для омонимов частотных типов построено разбиение на подклассы по признаку частотности использования грамматических характеристик.

Вычислительные задачи содержат в своем составе:

- задачу построения минимального множества разрешающих контекстов для каждого функционального типа омонимов;

Минимальность множества означает, что для каждого типа функционального омонима следует оценить сложность распознавания каждой части речи, принадлежащей данному типу. Затем необходимо построить множество разрешающих контекстов (МРК), имеющих минимальную сложность распознавания. Под разрешающим контекстом омонима $K_R(X)$ понимается минимальный контекст (минимальная последовательность словоформ), содержащий в своем составе омоним X и разрешающий элемент R, грамматические характеристики которого достаточны для установления грамматических характеристик омонима в этом контексте. При этом любое расширение разрешающего контекста (слева или справа) не влияет на результат разрешения омонима. В алгоритмической записи данное требование выражается следующим правилом: если для функционального омонима X, имеющего тип T1 или T2, применено правило из МРК, то тип омонима X определяется примененным правилом, иначе приписывается альтернативный тип;

- задачу формализации контекстных условий;
- задачу построения управляющей структуры обобщенного правила, обеспечивающего максимальную точность распознавания для каждого функционального типа омонимов.

Обобщенное правило представляет собой упорядоченную совокупность правил, записанных на специальном формальном языке. Каждое правило фиксирует некоторый разрешающий контекст. Для наглядности набор правил можно представить в виде двоичного дерева, в листьях которого хранятся значения частей речи, а в узлах логические условия. Правый потомок узла выбирается, если логическое условие ложно, левый – если истинно.

Управляющая структура задает порядок применения правил, который обычно базируется на оценке частотности контекстов. Оценки частотности контекстов могут быть получены путем применения вероятностных методов.

Разработка обобщенного правила состоит из решения следующих подзадач:

a) Определение значения по умолчанию. Обобщенное правило всегда имеет значение по умолчанию, которое приписывается омониму, если ни одно из локальных правил не применено.

b) Построение оценки сложности алгоритмического разрешения каждого функционального типа омонима, которая включает определение минимального множества разрешающих контекстов для каждого типа. Функциональный тип с минимальной сложностью распознавания становится опорным для обобщенного правила. Обобщенное правило ориентируется на распознавание опорного типа, т.е. выстраивается последовательность локальных правил для распознавания опорного типа, и если ни одно из локальных правил не удается применить, омониму приписывается альтернативный (дефолтный) тип.

c) Упорядочивание локальных правил в структуре обобщенного правила необходимо для достижения максимальной точности распознавания омонимов заданного типа. Первыми отрабатывают правила с высокой частотностью и устойчивым (бесконфликтным) разрешением. Метод контекстных правил использует ограниченный контекст и поэтому резкие изменения контекстных расстояний (под влиянием существенных изменений порядка слов) влияют на точность метода. Поэтому при выборе последовательности правил учитываются возможные инверсии порядка слов и потенциальные контекстные флуктуации. Чем сильнее потенциальная динамика контекста, тем позднее применение соответствующего правила. Таким образом, упорядочивание локальных правил обычно производится на основе критериев частотности и динамичности локальных контекстов. Для выбора оптимального порядка проводились компьютерные эксперименты.

В результате проведенных исследований разработана программа на языке C#, в которой реализован алгоритм, основанный на двух обобщенных правилах разрешения омонимии. На первом этапе

работы программа использует морфологический анализатор Mystem (разработка Яндекса [7]) для получения грамматических характеристик всех слов обрабатываемого предложения. Результаты работы анализатора являются входящей информацией для метода КРФО.

Программная реализация метода КРФО для двух типов омонимии была протестирована на достаточно большом наборе предложений. Отладка обобщенного правила была связана с настройкой изменяемых параметров обобщенного правила, к числу которых относится длина разрешающего контекста и порядок локальных правил.

Для типа омонимии «существительные/полные прилагательные» была получена точность разрешения омонимии с помощью обобщенного правил - не ниже 90% (т.е. имело место корректное определение части речи в предложении). В ряде случаев не удалось добиться автоматического разрешения омонимии. Дело в том, что в этих случаях программа Mystem (морфологический анализатор текста на РЯ), разработанная компанией «Яндекс», присваивала омониму только характеристики прилагательного, и не присваивала возможные характеристики существительного. Например, словам: *битый, избранный, истинное, крылатые, курсовая, курящий, лаборантская, ленивый, мудрые, обездоленный,* анализатор Mystem приписывает только характеристики полного прилагательного, в то время как характеристики их как существительного анализатор не выдает.

Для типа омонимии «существительные/ формы глаголов» были получены результаты корректного распознавания существительных не ниже 85% и форм глагола не ниже 60%.

Также как и в первом случае, морфологический анализатор Mystem работает не всегда корректно. Например, словам: *прости, три, шутила* присваивает только характеристики глагола, опуская характеристики этих слов, как существительных.

Более низкий результат распознавания для форм глагола по сравнению с существительными и прилагательными обусловлен тем, что формы глагола используются в основном в художественной литературе, с ее сложной структурой предложения. В технических же текстах форма глагола часто используется для написания пользовательских инструкций. Анализируемые предложения данного типа, как правило, короткие, и контекста бывает недостаточно, чтобы обобщенное правило.

Итоги

В работе рассмотрены прикладные проблемы разрешения функциональной омонимии в русском языке. Предлагаемый алгоритм позволяет разрешать проблему омонимии до начала работы

синтаксического анализатора - на предсинтаксическом этапе. Получены следующие основные результаты:

1. Выполнен обзор явления омонимии и способов ее разрешения, выявлены наиболее сложные и востребованные для разрешения виды функциональной омонимии.

2. Проведен тщательный анализ и модификация существующих правил контекстного разрешения функциональной омонимии отобранных типов.

3. Разработан алгоритм, осуществляющий разрешение морфологической неоднозначности указанных типов.

4. Выполнена программная реализация разработанного алгоритма на языке C#. Продемонстрирована достаточно высокая эффективность полученной программы.

Литература

1. Ахманова О.С. Очерки по общей и русской лексикологии. М.: Учпедгиз, 1957 – 295 стр.
2. Кобзарева Т.Ю., Афанасьев Р.Н. Универсальный модуль предсинтаксического анализа омонимии частей речи в РЯ на основе словаря диагностических ситуаций // Труды междунар. конференции Диалог'2002. М., 2002. С. 258-268.
3. Ким О.М., Островкина И.Е. Словарь грамматических омонимов русского языка. М, 2004.
4. Маслов Ю.С. Избранные труды. Аспектология. Общее языкознание. М.: Языки славянской культуры, 2004. – 840 с.
5. Мальковский М.Г., Арефьев Н.В. «Сочетаемостные ограничения в системе автоматического синтаксического анализа» // Тверь – 2012, №1 – С.28-31.
6. Невзорова О.А., Зинькина Н.В., Пяткин Н.В. Метод контекстного разрешения функциональной омонимии: анализ применимости // Труды межд. конф. Диалог'2006. М., Наука, 2006. С. 399 – 402.
7. http://company.yandex.ru/technology/mystem/

Раздел IV
Алгоритмы управления в системах реального времени

Антоненко В. А.[1], Вдовин П. М.[1], Волканов Д. Ю.[1], Глонина А. Б.[1], Захаров В. А.[1], Зорин Д. А.[1], Коннов И. В.[2], Пашков В. Н.[1], Подымов В. В.[1], Савенков К. О.[1], Смелянский Р. Л.[1], Чемерицкий Е. В.[1]

Система имитационного моделирования РВС РВ, основанная на стандарте HLA, и методика ее использования [3]

1. Введение

Распределённой вычислительной системой реального времени (РВС РВ) называют такую вычислительную систему, узлы которой распределены в пространстве, а правильность работы зависит от результатов вычислений и от продолжительности вычислений [1]. К числу РВС РВ относятся автоматизированные системы управления промышленным производством, энергетикой, транспортом, вооружением, банковскими операциями, медицинской аппаратурой. Поскольку ошибки в работе таких систем могут привести к большому ущербу, к РВС РВ предъявляются повышенные требования правильности и безотказности их функционирования.

Проектирование РВС РВ – это многоступенчатый процесс. Аппаратура и программное обеспечение РВС РВ создается разными производителями независимо друг от друга; характеристики некоторых компонентов системы являются строго заданными, параметры других могут варьироваться. На каждом этапе проектирования сложившиеся части системы подвергаются всестороннему анализу и испытаниям, по результатам которых в систему вносятся необходимые изменения и уточняются требования и задания, которые должны быть выполнены на следующих этапах. Правильность функционирования системы должна

---

[1] Московский Государственный Университет им. М.В. Ломоносова
[2] Technische Universität Wien

[3] Работа выполнена по Госконтракту 14.740.11.0399 с Минобрнауки РФ, в рамках федеральной целевой программы "Научные и научно-педагогические кадры инновационной России"

обеспечиваться на всех этапах ее создания. Поэтому проектирование РВС РВ – это отдельная отрасль программирования, в которой применяются особые методы и технологии.

Основными методами проверки правильности РВС РВ являются имитационное моделирование [2] и тестирование. Имитационному моделированию подвергается абстрактная схема или прототип, созданные на ранних этапах проектирования, а тестирование применяется непосредственно к самому продукту. На первом этапе разработки системы строится имитационная модель, позволяющая исследовать и оценить наиболее важные параметры системы, такие как директивные сроки выполнения заданий, допустимые задержки, порядок обмена сообщениями между процессами. В зависимости от степени готовности аппаратной части РВС РВ, здесь можно использовать либо программную модель всей проектируемой системы, либо комплекс, состоящий из натурных образцов аппаратуры и программных моделей приборов, собранных в единый стенд и сопряжённых через аппаратные каналы бортовых интерфейсов. Также можно варьировать уровень детальности моделирования – от так называемых "интервальных" моделей, которые отрабатывают заданные циклограммы обменов, не выполняя вычисление передаваемых прибором данных, до полных функциональных моделей, эквивалентных реальным приборам по составу и значениям выдаваемых в бортовые каналы данных и включающих в свой состав реальное ПО аппаратной части РВС РВ [3]. Такой тип имитационного моделирования называется полунатурным. Затем имитационная модель заменяется собственно аппаратной частью РВС РВ и программами, работающими на этой аппаратуре, и вся система подвергается всестороннему тестированию.

Для организации описанного выше процесса проектирования инженеру-разработчику необходимо иметь в своём распоряжении систему имитационного моделирования. Данная работа посвящена описанию такой системы моделирования и методике её применения.

2. Требования к системе моделирования РВС РВ

Вначале опишем детальные требования к системе моделирования (СМ) РВС РВ. Эти требования были сформированы на основе результатов исследований, приведённых в статьях [4,5], и личном опыте авторов данной работы.

- *Выбор оптимальной структуры СМ.* При создании СМ целесообразно как можно более широко использовать разработанные в открытых проектах программные компоненты. Для того чтобы это сделать СМ должно иметь чётко выделенную блочную структуру.

- *Организация выполнения набора моделей.* Разработанную модель необходимо запустить на выполнение. В имитационном моделировании выделяют распределённое имитационное моделирование (РИМ). РИМ имеет целый ряд достоинств над последовательным моделированием [6].
- *Совместимость средств моделирования и верификации.* При проектировании РВС РВ важной задачей является проверка соответствия поведения системы ее спецификации. Поэтому формат описания модели должен позволять проводить не только имитационное моделирование проектируемой РВС РВ, но и ее формальную верификацию.
- *Совместимость моделей.* Современная РВС РВ представляет собой сложный программно-аппаратный комплекс, состоящий из большого количества взаимодействующих между собой устройств. Обычно каждый компонент РВС РВ описывается отдельной моделью или группой моделей, и эти модели должны быть совместимы друг с другом.
- *Масштабируемость моделей.* Для проверки разных свойств поведения проектируемой РВС РВ могут потребоваться различные ее модели, имеющие разный уровень абстракции. Эти модели должны быть взаимосвязаны друг с другом. Поэтому в СМ целесообразно иметь средство, позволяющее проводить корректное масштабирование описаний модели РВС РВ [7].
- *Возможность создания имитационных моделей приборов РВС РВ, а также модели внешней среды.* Разработка РВС РВ с использованием имитационного моделирования проводится поэтапно. Вначале каждый компонент РВС РВ представляется простейшей программной моделью. Затем для более точного отражения поведение реальных устройств модели усложняются. На последних этапах создания комплекса программные модели замещаются разработанными прототипами устройств, вплоть до полного исключения программных компонент.
- *Автономное и интерактивное моделирование.* СМ должна предоставлять разработчику моделей средства для запуска, временной приостановки, возобновления и полной остановки проведения эксперимента. Также среда выполнения должна иметь встроенные средства для автономного запуска экспериментов без участия оператора.
- *Сопряжение с аппаратурой в модельном и в реальном времени по натурным каналам передачи данных.* СМ должна обеспечивать возможность подключения устройств с помощью подходящих натурных каналов и поддерживать скорость выполнения программных моделей на уровне, достаточном для соблюдения

спецификаций используемых протоколов передачи данных. Точность привязки модельного времени к физическому должна измеряться в десятках микросекунд. Для корректного построения имитационных моделей с такой точностью необходимо разработать среду выполнения с как можно меньшим временем отклика.

- *Возможность внесения отказов в каналы передачи данных.* При создании РВС РВ частым требованием является обеспечение заданного уровня устойчивости аппаратуры к отказам. Проверку данной характеристики распределённой системы можно частично осуществлять при запуске или прогоне модели, задавая уровень случайных ошибок, которые возникают при передаче данных между отдельными компонентами комплекса.
- *Регистрация и обработка результатов моделирования, в том числе взаимодействие с аппаратными мониторами каналов передачи данных.* Современные РВС РВ содержат сотни каналов связи, по которым данные должны передаваться корректно и своевременно [8]. Для проверки этих свойств необходимо регистрировать информацию обо всех событиях, происходящих при моделировании, и сохранять её в форме, удобной для последующей обработки.
- *Простота адаптации или автоматизация сторонних СМ к использованию совместно с библиотекой поддержки моделирования.* Разработчики отдельных компонентов РВС РВ могут располагать имитационными моделями своих устройств. Использование готовых имитационных моделей позволяет значительно облегчить разработку нового вычислительного комплекса. Поэтому среда выполнения должна поддерживать возможность подключения сторонних моделей.
- *Интероперабельность системы моделирования со сторонними системами.* Нужно иметь в виду, что отдельные устройства в составе РВС РВ могут быть созданы конкурирующими организациями, отказывающимися предоставить разработчикам программную модель своего компонента из опасения утечки их технологий в процессе моделирования.
- *Межмашинная синхронизация времени.* При проведении РИМ необходимо обеспечить корректный глобальный порядок имитационной модели могут выполняться на разных процессорах (компьютерах), между ними должна поддерживаться довольно точная синхронизация времени.
- *Открытые системы.* Открытость исходных кодов СМ позволяет повысить прозрачность ее функционирования, а также даёт большие возможности по поддержке и развитию СМ.

Анализ существующих СМ [5,8] показал, что перечисленным требованиям полностью не удовлетворяет ни одно из существующих средств, включая СМ ДИАНА [9] и стенд ПНМ [8], разработанные при участии некоторых из авторов данной работы. Начиная с 2010 года, в лаборатории вычислительных комплексов ф-та ВМК МГУ ведётся разработка СМ нового поколения ДИАНА-2012, совместимой с международными стандартами имитационного моделирования и современными методами анализа РВС РВ. В следующем разделе приводится формат описания РВС РВ, используемый в проекте.

3. Два уровня единого формата описания РВС РВ

Для обеспечения интероперабельности разрабатываемой СМ необходимо, чтобы модели, создаваемые в СМ, были совместимы со стандартом HLA [10]. В этом стандарте отдельные участники имитационного эксперимента, вне зависимости от их типа (программа, человек, аппаратное устройство), называются федератами. Совокупность федератов образует федерацию. Каждый федерат подключается к инфраструктуре RTI (Run-Time Infrastructure), которая обеспечивает их синхронизацию, выполняя, таким образом, функции среды выполнения. Фактически стандарт HLA описывает интерфейс между средой выполнения RTI и участниками моделирования (федератами) [11].

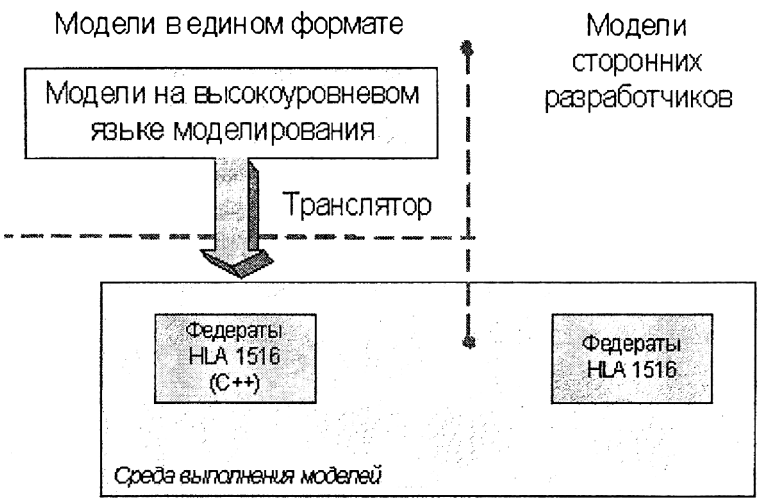


Рис. 1. Высокоуровневое описание моделей и единый формат

Интерфейс HLA содержит лишь набор низкоуровневых примитивов, облегчающих сопряжение имитационных моделей, предоставляемых различными разработчиками. Однако разработка новых моделей с применением лишь примитивов стандарта сложна и чревата ошибками. Для этой цели более пригоден язык высокого уровня (Рисунок 1). В качестве такого языка был выбран универсальный язык моделирования UML [12], позволяющий описывать как структуру системы, так и динамические аспекты её поведения. Архитектура системы моделирования ДИАНА-2012 приведена в следующем разделе.

4. Архитектура системы моделирования ДИАНА-2012

Архитектура разработанной системы моделирования приведена на рисунке 2. На рисунке синим цветом обозначены открытые средства, используемые в проекте без модификаций, жёлтым – средства модифицированные, для разрабатываемой СМ, а зелёным – средства, полностью разработанные и реализованные в рамках данной работы.

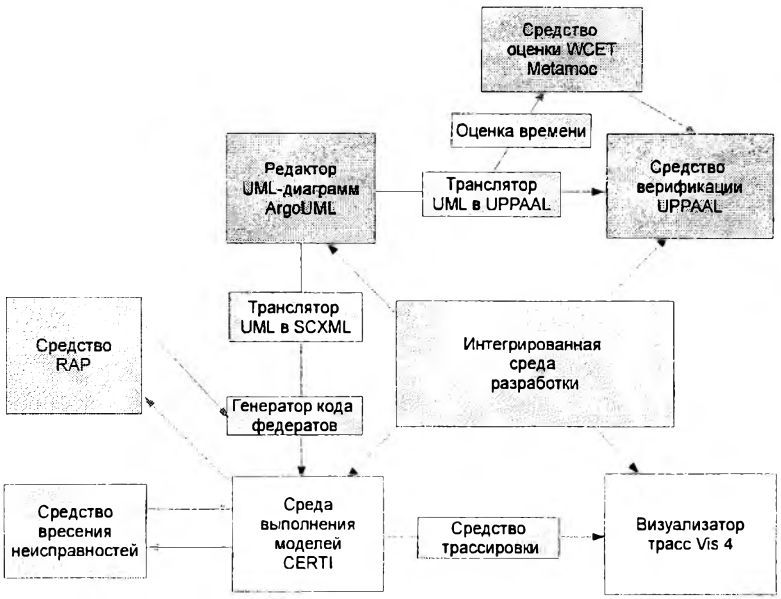


Рис. 2. Архитектура среды моделирования ДИАНА-2012

В состав среды моделирования входят следующие подсистемы:

- Редактор диаграмм состояний UML; в качестве редактора используется средство ArgoUML [13].

- Транслятор диаграмм UML в исполняемые модели, совместимые со стандартом HLA, состоит из двух подсистем: транслятора UML в SCXML и генератора кода федератов [14].
- Среда выполнения моделей на основе системы CERTI [15], дополненная средствами для поддержки моделирования РВС РВ, интеграции с натурными каналами и интеграции с библиотекой времени компиляции Proto-X, кодирующей данные с использованием встроенных типов языка C++. [16]
- Средство внесения неисправностей [17] в модель в стандарте HLA, взаимодействующее со средой выполнения моделей и позволяющее оперативно изменять параметры модели.
- Средство трассировки, представляющее собой специальный федерат HLA; оно позволяет записывать в трассу в формате OTF [18] события, возникающие в ходе моделирования.
- Средство анализа и визуализации трасс, созданное на основе средства vis3, входящего в Стенд ПНМ, и обеспечивающее интеграцию с форматом описания трасс OTF [19].
- Средство верификации UPPAAL [20], предназначенное для проверки свойства поведения систем реального времени, представленных в виде сетей временных автоматов. Оно снабжено транслятором, преобразующим диаграммы состояний UML в сети временных автоматов [21] с учетом оценок наихудшего времени выполнения программы, полученных с помощью средства Metamoc [22]. Средство трансляции было опробовано на различных моделях РВС РВ [21].
- Интегрированная среда разработки, объединяющая все указанные выше средства и запускающая процедуру моделирование. Также среда разработки интегрирована со средством для решения задачи выбора оптимального набора механизмов обеспечения отказоустойчивости (RAP) [23].

В рамках разработки СМ было написано более 700 килобайт исходного кода, преимущественно на языке Python (средства трансляции UML в различные форматы, среда разработки) и C++ (средства моделирования и визуализации). Весь исходный код документирован с помощью систем doxygen (для C++) и sphinx (для Python).

5. Методика использования системы

При разработке моделей РВС РВ с помощью созданных методов и инструментальных средств, рекомендуется придерживаться типичной последовательности действий, приведенной ниже.

Основным средством, реализующим взаимодействие всех созданных инструментов с пользователем, является интегрированная среда разработки моделей. Предполагается, что все остальные средства

запускаются через ее интерфейс, если явно не указано противное.

На первом этапе разработки модели обычно проводится её описание в виде диаграммы состояния UML. Для этого необходимо воспользоваться редактором ArgoUML [13]. На данном этапе необходимо определить цель моделирования и построить диаграммы состояний UML так, чтобы конструируемая модель была корректна (то есть исследуемые характеристики объекта в модели должны быть эквивалентны своим прообразам в исходном объекте) и адекватна (то есть ей присущи лишь характеристики, существенные для цели моделирования). В диаграмму могут быть включены фрагменты кода на C++. После завершения редактирования модели необходимо средствами ArgoUML экспортировать полученные диаграммы в формат XMI.

На следующем этапе проводится проверка того, что построенная модель удовлетворяет некоторым заданным свойствам. Этот этап необходим при исследовании сложных моделей РВС РВ, корректность которых должна быть строго доказана. Для проведения верификации необходимо преобразовать UML-диаграммы, представленные в формате XMI, в сети временные автоматы UPPAAL с помощью разработанного транслятора [21]. Транслятор может обнаруживать несколько видов ошибок в диаграммах, генерировать файлы, позволяющие использовать средство оценки наихудшего времени выполнения кода Metamoc [22], уменьшать количество получаемых временных автоматов. В случае использования средства Metamoc необходимо также задать характеристики целевой архитектуры.

Если транслятор не обнаружил ошибок, то можно приступать к верификации модели с помощью средства UPPAAL [20]. Для этого проверяемое свойство поведения диаграмм UML необходимо задать в виде формулы темпоральной логики TCTL [20] и запустить процедуру верификации. Верификатор проверяет выполнимость заданной формулы в дереве трасс вычислений сети временных автоматов, соответствующих диаграммам состояний UML. Если свойство не выполнено, то верификатор строит контрпример – одну из трасс, на которых проверяемое свойство нарушается. Этот контрпример можно конвертировать в трассу переходов UML с конкретными значениями параметров и таймеров. С временными автоматами можно работать и непосредственно в графическом интерфейсе средства UPPAAL.

Затем можно приступить к трансляции модели из UML-представления в код на C++ [14]. Процесс трансляции проходит в несколько этапов. Сначала необходимо по XMI-файлу сгенерировать диаграмму состояний в формате SCXML [24]. Данный формат гораздо проще XMI и не содержит информации о расположении объектов на диаграмме состояний. Простые модели можно строить сразу в формате SCXML, что особенно удобно в случае автоматического построения

моделей. Для построения/изменения SCXML-файлов существует редактор, интегрированный в среду разработки моделей. Кроме того, SCXML-диаграмму можно преобразовать в сети временных автоматы UPPAAL и провести её верификацию. На следующем этапе SCXML-диаграмма преобразуется в исходный код федератов на C++, генерируются служебные файлы, необходимые для запуска процесса моделирования.

После того, как код федератов получен, можно запустить имитационный эксперимент в среде CERTI [15].

Результатом моделирования является файл, содержащий трассу эксперимента в формате OTF [18], а также несколько служебных файлов, просмотр которых может быть полезен для отладки моделей. Для просмотра и исследования трассы эксперимента используется разработанное нами средство анализа и визуализации трасс Vis4 [19]. Оно позволяет получать информацию о событиях модели и информационных обменах между ее компонентами, осуществлять поиск события, навигацию по трассе, масштабирование трассы. В зависимости от результатов анализа трассы пользователь может вернуться на один из предыдущих этапов, изменить модель и повторить эксперимент.

На рисунке 3 показаны преобразования, которые происходят с форматами файлов в процессе работы с СМ.

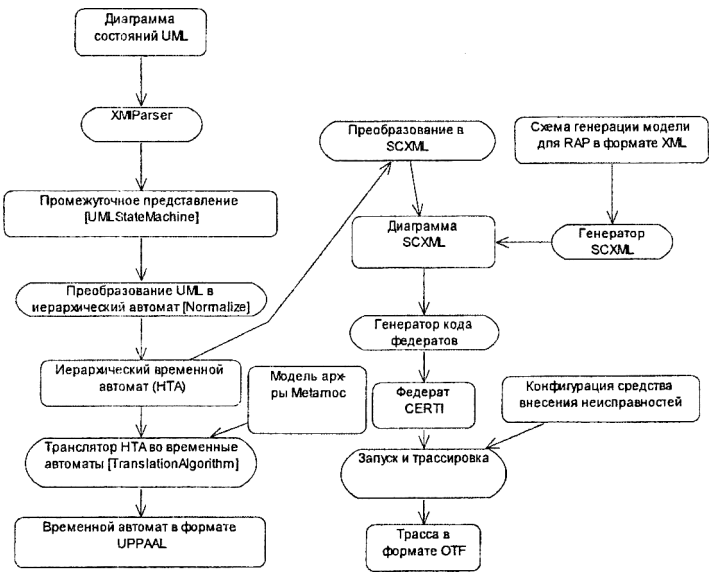


Рис. 3. Форматы файлов, используемые в СМ и их преобразования

Одним из полезных свойств разработанной среды моделирования является возможность ее интеграции со средствами планирования расписаний и синтеза архитектур для оценки времени выполнения работ [22]. Разработчику средств планирования (синтеза) предоставляется скрипт, который по расписанию заданного формата строит имитационную модель, проводит эксперимент, анализирует результаты и генерирует файл, содержащий время выполнения всех работ расписания. Таким образом, средство планирования должно лишь в определенные моменты генерировать файл с расписанием и вызывать скрипт. Примером может служить средство решения задачи выбора механизмов обеспечения отказоустойчивости РВС РВ [22], позволяющее строить РВС РВ высокой надежности при ограничениях на стоимость и время выполнения программ.

6. Заключение

В данной работе описана архитектура и методика использования системы имитационного моделирования для решения целого комплекса задач, возникающих при проектировании РВС РВ. Система удовлетворяет приведенным в разделе 2 требованиям и готова для эксплуатации. Для ее дальнейшего совершенствования мы планируем решить следующие задачи:

- Оптимизировать алгоритм трансляции диаграмм состояний UML в сети временных автоматов для преодоления эффекта «комбинаторного взрыва» в пространстве состояний автоматов.
- Разработать гибридный консервативно-оптимистический алгоритм синхронизации времени в среде выполнения моделей.
- Автоматизировать обработку результатов моделирования.

Литература

1. Stankovic J. A. Real-time Computing. // Byte Magazine – 1992. – v. 17, N 8. – p. 155-160.
2. Лоу А.М., Кельтон А.Д. Имитационное моделирование. Пер. с англ. (3-е изд.) – СПб.: BHV, 2004, 848 с.
3. Балашов В.В., Бахмуров А.Г., Волканов Д.Ю., Смелянский Р.Л., Чистолинов М.В., Ющенко Н.В. Стенд полунатурного моделирования для разработки встроенных вычислительных систем реального времени // Имитационное моделирование. Теория и Практика. 4-я Всероссийская научно-практическая конференция. Сборник докладов. Т.2. СПб: ОАО "ЦТСС", 2009. с. 215-219.
4. Смелянский Р.Л., Бахмуров А.Г. Применение метода полунатурного моделирования для отработки комплекса бортового оборудования

летательного аппарата // Труды Международной научно-технической конференции "СуперЭВМ и многопроцессорные вычислительные системы (МВС'2002)". Таганрог: Изд-во ТРТУ, 2002. - С.135-140.
5. Балашов В.В., Бахмуров А.Г., Волканов Д.Ю., Смелянский Р.Л., Чистолинов М.В., Ющенко Н.В. Стенд полунатурного моделирования для разработки встроенных вычислительных систем // Методы и средства обработки информации: Третья Всероссийская научная конференция.уды конференции. - М.: Издательский отдел факультета ВМиК МГУимени М.В. Ломоносова; МАКС Пресс, 2009. – с.16-25.
6. Замятина Е.Б.. Современные теории имитационного моделирования: специальный курс. ПГУ, Пермь, 2007.
7. Савенков К.О., Смелянский Р.Л., Масштабирование дискретно-событийных имитационных моделей // Программирование – 2006. – N 6 – с. 14-26.
8. Balashov V., Bakhmurov A., Chistolinov M., Smeliansky R., Volkanov D., Youshchenko N. A Hardware-in-the-Loop Simulation Environment for Real-Time Systems Development and Architecture Evaluation // Proceedings of the 3-rd International Conference on Dependability of Computer Systems DepCoS-RELCOMEX 2008, Szklarska Poreba, Poland, June 26-28 2008.
9. Bakhmurov A., Kapitonova A., Smeliansky R. DYANA: An Environment for Embedded System Design and Analysis // Proceedings of the 5-th International Conference TACAS'99, Lecture Notes in Computer Science – 1999. – v. 1579 – p. 390-404.
10. Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules // IEEE, 2010 – c. 26.
11. Simulation Interoperability Standards Committee of the IEEE Computer Society IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Federate Interface Specification. 2000.
12. Гома Х. UML. Проектирование систем реального времени, распределенных и параллельных приложений. М.: ДМК – 2011. – 704 с.
13. ArgoUML Homepage [HTTP] (http://argouml.tigris.org/)
14. Антоненко В.А., Волканов Д.Ю., Средство трансляции внутренней логики федерата HLA для построения моделей РВС РВ на языке UML // Труды конференции Моделирование-2012, 16-18 мая 2012 , Киев.
15. Noulard E., Rousselot J.-Y., CERTI, an Open Source RTI, why and how // Spring Simulation Interoperability Workshop. San Diego, USA, 2009.
16. Чемерицкий Е.В., Волканов Д.Ю., Смелянский Р.Л. Среда полунатурного моделирования на основе стандарта HLA // Труды международной научной конференции Моделирование-2012, Киев,

Украина, 16-18 мая 2012, стр. 454-457, К.: Три К (ISBN 978-966-7690-11-3).

17. Волканов Д.Ю., Шаров А.А. Программное средство автоматического внесения неисправностей для оценки надежности вычислительных систем реального времени с использованием имитационного моделирования // Методы и средства обработки информации. Труды второй Всероссийской научной конференции. – М.: Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова, 2005. – С.457-464.
18. Пашков В.Н., Волканов Д.Ю. О подходах к трассировке распределенных вычислительных систем реального времени. // Материалы 17-ой международной конференции по вычислительной механике и современным прикладным программным системам (ВМСППС'2011), 25-31 мая 2011 г., Алушта. - М.: Изд-во МАИ-ПРИНТ, 2011.
19. Пашков В.Н., Волканов Д.Ю. Разработка средства анализа и визуализации трасс распределенных вычислительных систем реального времени // Труды международной научной конференции «Моделирование-2012», Киев, Украина, 16-18 мая 2012, К.: Три К, с. 330-333.
20. Bengtsson J., Larsen K. G., Larsson F., Pettersson P., Yi W. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems // Lecture Notes in Computer Science. – 1996. – v. 1066. – p. 232-243.
21. Konnov I., Podymov V., Volkanov D., Zakharov V., Zorin D. On the designing of model checkers for real-time distributed systems // 3-rd Workshop "Program Semantics, Specification, and Verification: Theory and Applications", Nizhni Novgorod, Russia, July 1-2, 2012. – p. 72-81.
22. Dalsgaard A.E., Olesen M.C., Toft M., Hansen R.R., Larsen K. G., METAMOC: modular execution time analysis using model checking // Proceedings of the 10th International Workshop on Worst-Case Execution-Time Analysis (WCET2010). – p.113-123.
23. Bakhmurov A. G., Balashov V. V., Pashkov V.N., Smeliansky R.L., Volkanov D. Yu. Method For Choosing An Effective Set Of Fault Tolerance Mechanisms For Real-Time Embedded Systems, Based OnSimulation Modeling // Problems of dependability and modelling /eds. Jacek Mazurkiewicz [i in.]. Wrocław // Oficyna Wydawnicza Politechniki Wrocławskiej, 2011. p. 13-26.
24. State Chart XML (SCXML): State Machine Notation for Control Abstraction, W3C Working Draft 26 April 2011 [HTML] (http://www.w3.org/TR/scxml/)

Зорин Д. А.

Инструментальная система структурного синтеза вычислительных систем реального времени и построения расписаний

1. Введение

В работе[1] описана задача структурного синтеза вычислительных систем реального времени при ограничениях на надежность. Множество заданий или программа, для которых строится расписание, задано графом потока данных. Из-за зависимостей некоторые задания должны выполняться в строго определенном порядке, поэтому не любое расписание будет выполнимым (корректным). Требуется построить расписание, выполняющееся на наименьшем возможном числе процессоров. При этом также заданы ограничения на время выполнения расписания и требования к надежности системы [2].

Расписание определяется привязкой заданий к процессорам и их порядком выполнения на процессоре. Конкретные моменты начала и окончания выполнения задания не задаются, таким образом, расписание не зависит от конкретной физической системы и среды передачи данных между процессорами. Это позволяет сделать алгоритм построения расписания универсальным и платформо-независимым.

В работах [3,4] предложен алгоритм решения данной задачи, основанный на методе имитации отжига. Алгоритм допускает настройку на решение частных задач (заданы ограничения на возможные значения входных данных задачи) путем задания соответствующих значений параметров алгоритма. Также алгоритм допускает использование различных методик расчета надежности ВС РВ и имитационных моделей ВС РВ с различным уровнем детализации для оценки времени выполнения расписания. Это позволяет использовать алгоритм на различных этапах проектирования ВС РВ и делает возможным использования алгоритма в различных инструментальных средствах проектирования и отладки программного обеспечения ВС РВ.

Для практического применения алгоритма необходимо разработать программное средство, позволяющее решать задачу структурного синтеза в режиме диалога с пользователем. Приведем основные требования, которым должна удовлетворять такая система:

- Наличие понятного графического интерфейса пользователя.
- Возможность ввода исходных данных (графов) в диалоговом режиме.
- Возможность запускать алгоритм, менять его настройки,

визуализировать на экране результаты.

- Возможность править результаты алгоритма в ручном режиме. При изменении решения автоматически должна проверяться его корректность.
- Возможность задавать различные модели вычислительной системы для оценки времени.
- Возможность генерации кода процедур обмена.

2. Описание инструментальной системы

Общее описание

Инструментальная система написана на языке программирования Python [5] (версии 3.1) с использованием кросплатформенной библиотеки Qt, и может работать в операционных системах Windows и Linux.

ГПИ, построенный на основе программной библиотеки, представляет собой следующий набор взаимосвязанных окон:

– основное рабочее окно ГПИ;

– окно для просмотра и редактирования исходных данных;

– окно для просмотра и редактирования построенных расписаний;

Основное окно

Основное окно ГПИ служит для выполнения всех действий, необходимых в процессе работы с программой. Действия представлены пунктами в меню, кнопками на панели инструментов и кнопками в самом окне. Программа работает с «проектами». Каждый проект содержит исходные данные и результаты работы алгоритма. Файлы проектов – бинарные, сжаты с помощью библиотеки pickle.

В главном окне содержатся следующие элементы:

- Имя проекта. Нажатие на кнопку справа от имени позволяет его редактировать;
- Статистика. Представлены размеры исходных данных: число вершин и ребер в графе программы, ограничения на время и надежность и длина трассы операций, совершенных алгоритмом. Ограничения на время и надежность можно редактировать аналогично имени проекта. Кнопка справа от числа вершин вызывает редактор исходных данных;
- Кнопка запуска алгоритма, список доступных алгоритмов и полоска, показывающая прогресс работы алгоритма;
- Список ошибок в исходных данных. Если среди ошибок есть критические, запуск алгоритма становится недоступным. Кнопки справа вверху списка позволяют сворачивать и разворачивать его;

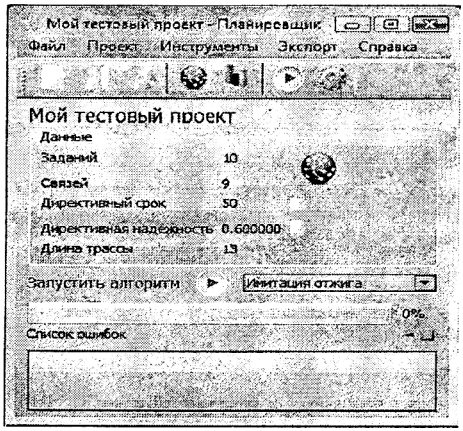


Рис. 1. Основное окно программы

Меню держит все действия, необходимые для работы в программе. Для большинства пунктов также добавлены горячие клавиши (по возможности стандартные, например, для сохранения – Ctrl+S) и иконки на панели инструментов. В меню доступны следующие действия:

- Создание проекта;
- Открытие проекта;
- Сохранение проекта под текущим или новым именем;
- Открытие окна для ввода исходных данных;
- Выбор метода оценки времени и его параметров;
- Выбор алгоритма структурного синтеза и его настройка: доступны алгоритм имитации отжига, описанный в [3,4] и экспериментальная реализация генетического алгоритма;
- Запуск и перезапуск алгоритма;
- Открытие окна для визуализации и ручной корректировки результатов алгоритма;
- Настройка программы: язык интерфейса (русский или английский), визуальное оформление окон;
- Экспорт результатов в различные форматы.

Редактор исходных данных

Окно редактора исходных данных позволяет редактировать граф потока данных программы, для которой строится расписание.

Исходные данные могут быть сохранены отдельно от проекта в виде XML-файла, который можно редактировать вручную. Меню редактора позволяет загружать исходные данные из XML и сохранять под заданным именем.

Большая часть окна представляет собой поле для рисования графа расписания. Редактор может находиться в одном из трех состояний, каждому из которых соответствует кнопка на панели инструментов:

- Кнопка с иконкой курсора – редактирование графа.
- Кнопка с иконкой страницы – добавление заданий.
- Кнопка с двумя зелеными стрелками – добавление связей.

При добавлении заданий новое задание появляется на поле при каждом клике левой кнопкой мыши.

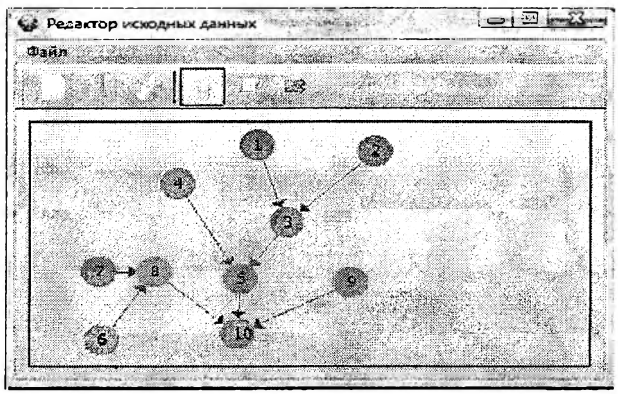


Рис. 2. Окно редактора исходных данных

При добавлении связей нужно провести стрелку между нужными заданиями, удерживая нажатой левую кнопку мыши. При редактировании графа клик левой кнопкой позволяет выделять вершины и ребра другим цветом. Вершины можно перетаскивать по полю, удерживая левую кнопку мыши. Клик вне вершин и ребер приводит к снятию выделения. Нажатие на кнопку Delete приводит к удалению выделенного ребра или выделенной вершины. Нажатие на кнопку Enter или двойной клик вызывает диалоговое окно редактирования вершины либо ребра. У ребер можно редактировать имя и объем передаваемых данных. У вершин можно задавать имя, время выполнения соответствующего задания, а также список версий. Версии добавляются и удаляются кнопками с плюсом и крестиком. Значение надежности в таблице редактируемое.

Редактор расписаний

В редакторе расписаний визуализируются расписания, найденные алгоритмом. До запуска алгоритма там доступно только начальное приближение, после появляется возможность просмотреть всю трассу.

Ось времени изображена и размечена внизу основного поля; она направлена вправо, директивный срок обозначен красной линией. Горизонтальные линии соответствуют процессорам, прямоугольники – заданиям, стрелки – передачам данных. Координаты прямоугольников и стрелок точно соответствуют моментам времени, когда соответствующие задания и операции передачи данных происходили. Прямоугольники подписаны номерами заданий, версия с номером 1 не обозначается, остальные помечены v2, v3 и так далее.

Справа от поля расписания указаны его характеристики: текущие значения времени выполнения, надежности и число процессоров. Кнопки внизу служат для навигации по трассе построенных расписаний: можно двигаться к следующему и предыдущему расписанию (для этого также есть пункты в меню и клавиши PgUp, PgDn), к началу и к концу. Можно переместиться к произвольному месту трассы, введя номер в редактируемое поле. Клавиша Home и соответствующий пункт меню перемещают к лучшему из найденных расписаний. Ползунок в правом нижнем углу позволяет изменять масштаб изображения расписания. Также в данном окне есть возможность ручного редактирования расписания. При применении операций вручную они добавляются в трассу так, как будто бы они были сделаны алгоритмом при автоматическом поиске, в частности, обновляется значение лучшего найденного расписания.

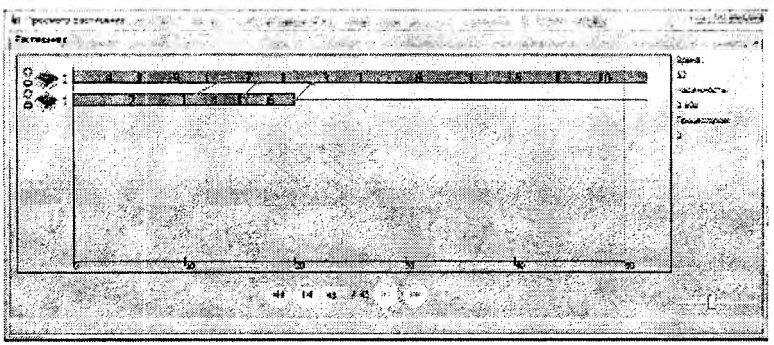


Рис. 3. Окно редактора расписаний

Если применить вручную операции к расписанию, находящемуся не в конце трассы, то все последующие шаги удаляются.

Операции добавления и удаления процессоров можно выполнять с помощью иконок с плюсом и минусом возле значка процессора у левого края окна. Число справа от значка процессора есть количество резервных копий.

Клик левой кнопкой мыши по прямоугольнику выделяет его другим цветом, на оси времени появляются моменты начала и

окончания выполнения этого задания. Если к выделенному заданию можно добавлять или удалять версии, на краях прямоугольника появляются соответствующие значки с плюсом и минусом, клик по ним приводит к выполнению операции добавления/удаления версий.

Удерживая левую кнопку мыши, задание можно перетащить на другую позицию в расписании: соответствующая позиция подсвечивается. При отпускании мыши делается операция переноса задания. Если операция невозможна из-за нарушения условий корректности, сообщение об этом появляется в строке статуса, а операция не выполняется.

Плагины для оценки времени

Для пользователей доступна возможность создавать произвольные методы оценки времени выполнения расписания и подключать их как плагины. Плагины хранятся в папке plugins в виде исполняемых файлов на языке python. Для корректной работы код плагина должен иметь следующий вид:

```
classFibreChannelInterpreter:
idletimes = []
    delays = []
    endtimes = []
    executionTimes = {}
    deliveryTimes = []

    dummy = 0

    def __init__(self):
        pass

    def GetName():
        return "Fibre Channel"

    def Interpret(self, schedule):
        return 0
    def GetSettings(self):
        from PyQt4.QtCore import QObject
        class Translator(QObject):
            def __init__(self, parent):
                QObject.__init__(self)
                self.parent = parent
            def getTranslatedSettings(self):
                return [
       [self.tr("Channel speed"),self.parent.dummy]
                        ]
        t = Translator(self)
        return t.getTranslatedSettings()

    def UpdateSettings(self, dict):
        self.dummy = dict[0][1]
```

```
def pluginMain():
    return FibreChannelInterpreter
```

ФункцияpluginMainдолжнавозвращатьобъект-класс, оценивающийвремядлязаданногорасписания.

Класс плагина должен содержать как минимум следующие элементы:

– Конструктор;

– Функция GetName, возвращающая имя плагина, которое отображается в списке плагинов в главном окне;

– Функция Interpret, принимающая на вход объект класса Schedule и возвращающая вычисленное время выполнения. Во время интерпретации должны заполняться пять списков, соответствующих простоям, задержкам, времени выполнения, времени передачи и времени окончания работы (подробнее о назначении каждого из этих списков сказано в описании алгоритма в [4]);

– Функции GetSettings и UpdateSettings для установки параметров плагина. Функция GetSettings должна возвращать список настроек. Каждый элемент данного списка – это пара «имя - объект». В окне настроек все объекты отображаются последовательно, сначала имя, затем редактор для соответствующего объекта. Если объект имеет числовой тип, то для него отображается строковый редактор. Если объект имеет тип список, то он должен состоять из двух элементов [L, n], где L – список строк, а n – номер элемента в первом списке. Данный объект отображается как раскрывающийся список из строк L, в котором выбрана строка n. Если объект имеет тип словарь, то вложенные в него объекты (числа и списки строк) группируются в список, озаглавленный ключом словаря. В примере плагина выше настройки состоят из одного числового параметра dummy, также приведен образец реализации корректного перевода имен настроек;

По умолчанию всегда присутствует метод оценки времени для полносвязной архитектуры (Default). В поставку включены также плагины, реализующие оценку времени для архитектуры с общей шиной (Bus) и с коммутатором Fibre Channel. Для всех трех реализованных методов оценки времени возможна настройка пропускной способности канала и задержки передачи, для Fibre Channel также можно задавать ограничение на число доступных каналов;

Сохранение результатов

Результаты работы алгоритма могут быть экспортированы в двух форматах. Первый формат представляет собой XML-файл, содержащий информацию обо всех используемых процессорах и порядке выполнения заданий на них. Формат совместим со средством трансляции в файлы среды выполнения моделей вычислительных систем реального времени CERTI[6]. Второй вариант экспорта –

генерация кода процедур обмена между процессорами на языке C++. Сгенерированный код может быть использован в сочетании с собственно кодом заданий для параллельного запуска в соответствии с построенным расписанием. Для обмена между процессорами используется библиотека MPI [7].

Заключение

В работе описана инструментальная система для решения задачи структурного синтеза вычислительных систем реального времени. Система удовлетворяет приведенным во введении требованиям и готова для эксплуатации.

Благодаря возможности подключения моделей вычислительных систем с разным уровнем детализации для оценки времени выполнения и надежности, систему можно использовать на различных этапах проектирования вычислительной системы. Расписание может быть перестроено на более позднем этапе по мере уточнения данных о заданиях, подлежащих планированию.

Литература

1. Зорин Д.А. Способ представления и преобразования расписаний в итерационных алгоритмах структурного синтеза вычислительных систем реального времени // Программные системы и инструменты. Тематический сборник № 12, М.: Изд-во факультета ВМиК МГУ, 2011. С. 163-171.
2. Wattanapongsakorn N., Levitan S.P. Reliability optimization models for embedded systems with multiple applications// Reliability, IEEE Transactions on. 2004., 53,P. 406-416.
3. D. A. Zorin and V. A. Kostenko Algorithm for Synthesis of Real-Time Systems under Reliability Constraints // Journal of Computer and Systems Sciences International. 2012. Vol. 51. No. 3. P. 410–417.
4. Зорин Д.А. Сравнение различных стратегий применения операций в алгоритме имитации отжига для задачи построения расписаний для многопроцессорных систем // "Параллельные вычисления и задачи управления" PACO'2012. Шестая международная конференция, Москва, 24-26 окт. 2012 г., Труды: в 3 т. М.: ИПУ РАН, 2012. Том 1. С. 278-291.
5. Python v3.2 documentation [HTML] (http://docs.python.org/py3k/).
6. Eugene Chemeritskiy Towards a HLA-based Hardware-In-the-Loop simulation runtime // Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE). Perm, Russia, 2012. P. 144-150.
7. Pacheco, Peter S. Parallel Programming with MPI // Morgan Kaufmann, 1997.500 pp.

Плакунов А. В.

Способы сведения задачи построения расписания обменов по каналу с централизованным управлением к задаче нахождения на графе пути

1. Введение

В бортовых системах реального времени широко используется архитектура, основанная на каналах с централизованным управлением. Примерами каналов с централизованным управлением являются MIL STD-1553B (МКИО ГОСТ Р52070-2003) [1], STANAG 3910 [2], FC-AE-1553 [3].

Канал обеспечивает обмен данными между устройствами, присоединёнными к каналу (далее – оконечные устройства). Обмен представляет собой последовательность передач прикладных и служебных данных между оконечными устройствами. Время начала каждого обмена определяет расписание, которое строят заранее, и которое не меняется в ходе функционирования бортовой системы. Расписание исполняется контроллером, который является одним из оконечных устройств. Для решения задачи построения расписания в данной работе будут применяться муравьиные алгоритмы [4,5]. Муравьиные алгоритмы впервые были использованы для решения задачи коммивояжера [6]. Для этого строится граф, вершины которого соответствуют городам в задаче. Вершины соединяются ребром, если коммивояжер может перейти из одного города, соответствующего вершине, в другой. Путь в таком графе определяет последовательность городов, и задачей является найти путь минимальной длины, где длина пути равна сумме длин ребер, входящих в путь.

Каждому ребру построенного графа соответствуют два значения – эвристическая функция, в простейшем случае равная расстоянию между городами, и количество феромона. Используя эти значения, искусственные муравьи по вероятностному правилу строят пути в графе. В зависимости от длины полученного пути, изменяется количество феромона на ребрах графа, входящих в путь. Количество феромона используется муравьями на следующих итерациях. Таким образом, строится дополнительная разметка исходных данных, которая используется для построения решения на каждой итерации алгоритма и уточняется по мере увеличения числа итераций. При использовании муравьиных алгоритмов не возникает проблема «четкого» выделения частной задачи.

Алгоритмы, основанные на использовании схемы муравьиных колоний, были также успешно использованы для решения таких

комбинаторных задач, как квадратичная задача о назначениях [7], задача упаковки в контейнеры [8], задачи построения расписаний [9-11]. Аналогично задаче коммивояжера, для использования муравьиных алгоритмов для решения задач построения расписания требуется свести задачу построения расписания к задаче нахождения на графе пути, обладающего определенными свойствами.

В данной работе предлагается два подхода к сведению задачи построения расписания обменов по каналу с централизованным управлением к задаче нахождения пути на графе, и приводятся результаты их экспериментального сравнения.

2. Задача построения расписания обменов по каналу с централизованным управлением

Набор сообщений, экземпляры которых требуется разместить в расписание, задается как $SM = \{M_i =< v_i, r_i, p_i^L, p_i^R > | i \in [1..n]\}$, где v_i – время передачи сообщения, r_i – частота передачи сообщения, p_i^L и p_i^R – фазовые сдвиги. Фазовым сдвигом будем называть отступ от границы периода сообщения, который будет учитываться при определении директивного интервала соответствующих сообщению работ.

Множество экземпляров сообщений (множество работ) вычисляется по множеству *SM* следующим образом: для каждого сообщения формируется множество соответствующих ему работ: $W^i = \{A_j =< t_j, s_j, f_j > | j \in [1..n_i]\}$, где $n_i = \left\lceil \frac{r_{scl}}{\tau_i} \right\rceil$ – количество экземпляров сообщения, которые можно разместить в интервале планирования, $\tau_i = \frac{1}{r_i}$ - период сообщения, r_{scl} – длительность интервала планирования (обычно задается как наименьшее общее кратное периодов сообщений), $s_j = (j-1)\cdot\tau_i + p_i^L$; $f_j = (j-1)\cdot\tau_i + p_i^R$ – соответственно левая и правая границы директивного интервала (далее будем полагать, что для всех $j \in [1..n]$ выполнено $0 < t_j \le f_j - s_j$). Множество работ определяется как $SW = \bigcup_{i=1}^{n} W^i$.

Расписанием является набор подциклов с указанием упорядоченного списка работ выполняемых в каждом подцикле: $SP = \{C_i =< S_i, SW_i > | i \in [1..N]\}$, где $N = |SP|$ - число подциклов, S_i – время старта цепочки подцикла, $SW_i = \{A_j^i | j \in [1..L_i]\}$ – список работ, включенных в *i*-й подцикл, L_i – число работ в *i*-м подцикле.

Исходными данными для задачи построения расписания являются:

1) Множество сообщений $SM = \{M_i =< v_i, r_i, p_i^L, p_i^R > | i \in [1..n]\}$.

2) Длительность подцикла T, минимальный отступ от начала подцикла r_{mco}, резерв времени в конце подцикла r_{rf}, длительность интервала планирования r_{scl}, максимально допустимое отклонение от периода r_{mpe}, максимальное число работ в одной цепочке r_{mcc}, максимальная длительность выполнения работ в одной цепочке r_{mcl}, резерв для сдвига расписания r_{rfs}.

Требуется построить расписание $SP = \{C_i =< S_i, SW_i > | i \in [1..N]\}$. При этом для всех $i \in [1..N]$ должны соблюдаться следующие ограничения:

1) $\sum_{A_j \in SW_i} t_j \le r_{mcl}$ – длительность цепочки не должна превышать максимальной длительности.

2) $|SW_i| < r_{mcc}$ – длина цепочки не должна превышать заданной максимальной длины.

3) $S_i \ge r_{mco}$ – отступ от начала подцикла не должен быть меньше минимально возможного.

4) Расписание $SP = \{C_i =< S_i + r_{rfs}, SW_i > | i \in [1..N]\}$ должно быть корректным. Это означает, что сдвиг каждой работы вправо на r_{rfs} не должен приводить к нарушению любого из приведенных ограничений.

5) $S_i + T \cdot r_{rf} + \sum_{A_j \in SW_i} t_j \le T$ – суммарная длительность выполнения всех работ не должна превышать длины подцикла, с учетом отступа от начала и резерва времени в конце подцикла.

6) $\forall m \in [1..n], i \in [1..n_m] \Rightarrow (1 - r_{mpe}) \cdot \tau_m \le s_i^m - s_{i+1}^m \le (1 + r_{mpe}) \cdot \tau_m$, где s_i^m - время старта i-й работы, соответствующей сообщению m, - промежуток между временами старта соседних работ, соответствующих одному и тому же сообщению должен удовлетворять ограничению на отклонение от периода.

7) Для всех $i = [1..N]$ рассмотрим отношения $\prec_i$: $s_{ij} < s_{ik} \Leftrightarrow M_p \prec_i M_q$, где $A_j \in W_p$, $A_k \in W_q$ – два сообщения связаны отношением $\prec_i$ тогда и только тогда, когда соответствующие им работы присутствуют в i-м подцикле. Тогда отношение $\prec$, представляющее собой замыкание объединения всех отношений $\prec_i$, должно являться

отношением частичного порядка, т.е. $M_p \prec M_q \Rightarrow \overline{M_q \prec M_p}$.

8) Для всех $i = [1..N]$, $j \in [1..L_i]$ выполнено: $s_{ij} \geq s_j, s_{ij} + t_j < f_j$, где $s_{ij} = (i-1) \cdot T + S_i + \sum_{k=1}^{j-1} t_k$ – время начала выполнения j-й работы в i-м подцикле, - каждая работа должна выполняться в рамках своего директивного интервала.

9) Для всех $i = [1..N]$, $j \in [1..L_i]$ выполнено: $f_{ij} - s_{ij} = t_j$, где f_{ij} - время окончания j-й работы в i-м подцикле, - работы должны выполняться без прерываний.

10) Для всех $i, p = [1..N]$, $j, q \in [1..L_i]$ выполнено: $((s_{ij} < s_{pq}) \rightarrow (f_{ij} < s_{pq})) \vee ((s_{ij} \geq f_{pq}) \rightarrow (f_{ij} > f_{pq}))$ - интервалы выполнения работ не пересекаются.

Целевая функция $TRG(SP) = \frac{\sum_{k=1}^{N} |SW_k|}{n}$ задается как отношение суммарного числа размещённых работ по всем цепочкам к общему количеству работ. Если существует полное и корректное расписание SP_{full}, то $TRG(SP_{full}) = 1$ является оптимумом целевой функции.

3. Первый подход к сведению задачи построения расписания к задаче поиска кратчайшего пути в графе

Первый способ построения графа основан на соответствии вершин графа работам в задаче. Сопоставим каждой работе A_i из SW вершину r_i. Также введем специальную вершину O, с которой будет начинаться маршрут каждого муравья. Получим граф $G = < V, E >$, где $V = \{O, r_1, r_2, ..., r_n\}$ – множество вершин, $E = \{(u, v) \mid u, v \in V, u \neq v\}$ – множество ориентированных ребер (между любыми двумя различными вершинами есть два разнонаправленных ориентированных ребра), и таким образом, G – полносвязный ориентированный граф.

Первый способ построения графа позволяет рассматривать все возможные последовательности работ, однако не учитывает связи соответствующих одному сообщению работ между собой.

4. Второй способ сведения задачи построения расписания к задаче поиска кратчайшего пути в графе

Второй способ построения графа основан на соответствии вершин графа сообщениям. Сопоставим каждому сообщению M_i из SM вершину v_i. Аналогично, введем специальную вершину O, с которой будет начинаться маршрут каждого муравья. Получим граф

$G =< V, E >$, где $V = \{O, v_1, v_2, ..., v_n\}$ – множество вершин, $E = \{(u,v) \mid u,v \in V, u \neq v\}$ – множество ориентированных ребер (между любыми двумя вершинами есть два разнонаправленных ориентированных ребра), и аналогично получим, что G – полносвязный ориентированный граф.

Второй способ построения графа позволяет учитывать связь соответствующих одному сообщению работ между собой и значительно снижает количество вершин графа.

5. Алгоритм построения расписания

Схема работы гибридного алгоритма следующая:

1) Построение муравьями путей $P = \{P_i, i = [1..N_{ant}]\}$ в графе G с учетом феромона и эвристической функции на ребрах графа: $P =$ ***select(G)***.

2) Применение алгоритма упаковки [12,13], на вход которому подаются последовательности вершин, построенные муравьиным алгоритмом: SP_i=***build(P$_i$)***.

3) Вычисление целевой функции: $T_i = TRG(SP_i)$.

4) Обновление количества феромона на ребрах графа в зависимости от качества решений. Качество решения определяется значением целевой функции: ***update(G,P,T)***.

5) Если условие останова не выполнено, переход к п.1;

5.1. Построение пути в графе

Для построения пути в графе используется вероятностное правило: вероятность перехода *k-го* муравья из *i*-й вершины в *j*-ю на итерации *t* зависит от тех вершин, которые муравей уже посетил, количества феромона и значения эвристической функции на ребре:

$$P_{ij,k}(t) = \begin{cases} \dfrac{(\tau_{ij}(t))^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum_{l \in J_k} (\tau_{il}(t))^{\alpha} \cdot (\eta_{il})^{\beta}}, j \notin L_k \\ 0, j \in L_k \end{cases}$$

Здесь $\tau_{ij}(t)$ - количество феромона на ребре (i,j), η_{ij} - значение эвристической функции на ребре (i,j), $\alpha \geq 0$ и $\beta \geq 0$ - параметры алгоритма, определяющие важность феромонного следа и локальной целевой функции, L_k – список посещённых вершин муравья k.

5.2. Алгоритм размещения работ в расписание

Расписание строится алгоритмом упаковки, который последовательно добавляет работы в расписание в порядке,

определяемом маршрутом P в графе G.

Во время построения расписания снимается ограничение на количество цепочек в подцикле. В конце работы алгоритма осуществляется попытка сдвига всех цепочек к самой правой цепочке, при этом работы, у которых оказывается нарушен директивный срок выполнения, не включаются в расписание.

Пусть $P = \{p_i \mid i \in [1..n_{jobs}]\}$, A_j – работа, соответствующая вершине p_i. Маршрут определяет последовательность работ $A = \{A_j \mid j \in [1..n_{jobs}]\}$. При размещении в расписании работа попадает либо в конец или середину одной из существующих цепочек, либо инициирует создание новой цепочки, содержащей только размещаемую работу:

1) Подциклы сортируются по критерию заполненности, т.е. в порядке увеличения суммарной длительности выполнения всех работ, включенных в подцикл. При одинаковом значении критерия, сортировка происходит по увеличению времени старта подцикла.

2) Очередной подцикл выбирается по порядку из отсортированного в п.1 списка, если директивный интервал работы позволяет разместить её в выбранный подцикл, и если ограничения 1, 2 не будут нарушены:

2.1) В выбранном подцикле просматриваются все цепочки в порядке увеличения их времен старта.

2.2) Для очередной цепочки производится попытка разместить работу в её конец (если это не нарушает ограничений 5, 7) со сдвигом других цепочек подцикла вправо по временной оси, если это необходимо и не нарушает ограничение 5.

2.3) Если работу нельзя разместить в конец ни одной из цепочек, цепочки просматриваются заново в том же порядке.

2.4) Для очередной цепочки производится попытка разместить работу между двумя соседними уже размещенными работами (если это не нарушает ограничение 7) со сдвигом работ цепочки вправо по временной оси, если это не нарушает ограничение 5.

3) Если операция в п.3 также невыполнима, и время старта работы находится в промежутке между двумя цепочками или превышает время завершения последней цепочки подцикла, создается новая цепочка, содержащая только размещаемую работу, если это не нарушает ограничений 3, 5, 7.

4) Если создание новой цепочки невозможно, пункты 2-3 выполняются для следующего по порядку подцикла. Работа размещается в первый подходящий для этого подцикл. Если работу не удается разместить ни в один подцикл, она помещается в список неразмещенных работ.

При использовании второго способа построения графа описанный выше алгоритм используется для размещения в расписание

работ, соответствующие выбранному сообщению в порядке возрастания их левой границы директивного интервала.

После проведения вышеописанных действий для каждой размещенной работы и для каждого подцикла происходит попытка сдвига всех цепочек к самой правой цепочке этого подцикла, при этом работы, у которых оказывается нарушен директивный срок выполнения или которые нарушают ограничение 6, не включаются в расписание.

После сдвига в каждом подцикле остается не более одной цепочки, что позволяет получить корректное расписание – завершение работы контейнерного алгоритма.

5.3. Обновление количества феромона

После того, как вычислены целевые функции T_i, выполняется завершающий этап итерации - операция ***update(G,P,T)***. На каждое ребро графа, входящее в путь, откладывается феромон, количество которого зависит от качества решения, которому соответствует этот путь:

$$\Delta \tau_{ij,k}(t) = \begin{cases} T_k, (i,j) \in P_k(t) \\ 0, (i,j) \notin P_k(t) \end{cases}$$

Здесь $P_k(t)$ – путь, построенный k-м муравьем, а целевая функция $T_i = TRG(SP_i)$ задается как отношение количества размещённых в расписание работ к общему числу работ.

В качестве параметра алгоритма задается коэффициент испарения феромона $p \in [0;1]$, определяющий, какая часть феромона останется с предыдущих итераций. Тогда суммарное количество феромона на ребре (i, j) после итерации t вычисляется по формуле:

$$\tau_{ij}(t+1) = (1-p) \cdot \tau_{ij}(t) + \sum_{k=1}^{m} \Delta \tau_{ij,k}(t).$$

Испарение феромона вводится для избегания попадания алгоритма в локальный оптимум, когда решение с относительно хорошим значением целевой функции остается единственно значимым, а также для отбрасывания решений, которые с большой вероятностью являются плохими.

6. Исследования динамики работы алгоритмов

Для исследования динамики работы алгоритмов была разработана метрика в пространстве решений, которая позволяет вычислить расстояние между двумя любыми маршрутами в графе.

Рассмотрим отображение $M : P \times P \to \mathrm{N} \cup \{0\}$, которое для любой пары маршрутов P_1 и P_2 возвращает целое неотрицательное число. Пусть даны маршруты $P_1 = \{p_i^1, i = [1..n]\}, P_2 = \{p_i^2, i = [1..n]\}$. Множество дуг Arc_i маршрута P_i определяется так:

$Arc_i = \{(p'_k, p'_{k+1}), k = [1..n]\}$. Тогда значение $M(P_1,P_2)$ вычисляется так: $M(P_1, P_2) = |Arc| - |Arc_1 \cap Arc_2|$, где $|Arc| = |Arc_1| = |Arc_2|$.

Исследование динамики алгоритма происходит путем разбиения путей на группы:

1. Фиксируется число r.
2. Для каждого пути P и для каждой группы G вычисляется расстояние $M(P,C_g)$ между путем P и центром группы C_g.
3. Путь P добавляется в состав группы, к центру которой он ближе всего, с условием, что расстояние до центра не превышает числа r.

Если путь не был добавлен ни в одну группу, создается новая группа G_P, и путь P становится центром этой группы. Результаты исследования представлены в таблицах 1 и 2. В таблице 1 показан алгоритм на основе соответствия вершин работам, в таблице 2 – на основе соответствия вершин сообщениям.

Итерация	Общее число групп	Число «живых» групп	Макс. Размер
20	1600	0	0
40	2336	6	64
60	2721	10	179
80	3011	11	3534
100	3287	11	3534
120	3620	11	3534
140	3993	12	3534
160	4386	12	3534
180	4781	14	3534
200	5171	17	3534

Табл. 1. Сравнение подходов к построению графа с помощью метрики.

Итерация	Общее число групп	Число «живых» групп	Макс. Размер
20	1531	2	150
40	1729	8	449
60	1852	13	449
80	1995	23	449
100	2110	25	698
120	2196	30	698
140	2376	40	698
160	2487	47	698
180	2585	51	698
200	2659	54	698

Табл. 2. Сравнение подходов к построению графа с помощью метрики.

«Живой» группой в таблицах называется группа, в которую хотя бы раз добавлялись новые пути, и в которой находится не менее 20 путей. Число r в экспериментах равнялось 25, при количестве вершин 950 для первого способа построения графа и 250 – для второго способа построения графа.

Как видно из таблиц, алгоритм, использовавший способ построения графа на основе соответствия вершин сообщениям, образовал больше «живых» групп, а значит, исследование пространства путей было более широким. Кроме того, алгоритм, использовавший способ построения графа на основе соответствия вершин работам, долгое время находился в локальном оптимуме.

Исследование в данном разделе проводилось путем однократного запуска каждого алгоритма на двух различных классах исходных данных. Каждый класс данных состоял из 120 различных наборов сообщений.

7. Сравнение точности разработанных алгоритмов

Отличие классов исходных данных друг от друга заключалось в структуре фазовых сдвигов. Сообщения, претендующие на размещение работ в подцикл, делились на несколько групп, работы, соответствующие сообщениям одной группы, претендовали на один и тот же отрезок подцикла. Класс исходных данных EXT имел два таких отрезка разной длительности, которые частично включались друг в друга. Класс исходных данных SPLIT имел три отрезка разной длительности, первый из которых включал в себя второй, и частично включал третий, притом второй и третий отрезки не пересекались. Загрузка шины выбиралась исходя из загрузки для реальных систем [14].

Алгоритмы выполняли 200 итераций на каждом из наборов сообщений, если не было найдено оптимального решения. Для построения графика результат сортировался по возрастанию значения целевой функции. Одна точка на графике соответствует усреднению пяти результатов запуска. Во всех случаях для алгоритма, основанного на схеме муравьиных колоний, в качестве эвристической функции использовалась взвешенная сумма $\mu_i^2 = \frac{w_i}{p_i^R - p_i^L}$ и $\mu_i^3 = \frac{1}{\tau_i}$.

На рисунках 1 и 2 представлено сравнение алгоритмов на основе предложенных подходов по критерию «значение целевой функции». Как видно из графиков, на обоих классах исходных данных алгоритм на основе соответствия вершин сообщениям показывает лучший результат, при этом на втором наборе данных результаты отличались значительно.

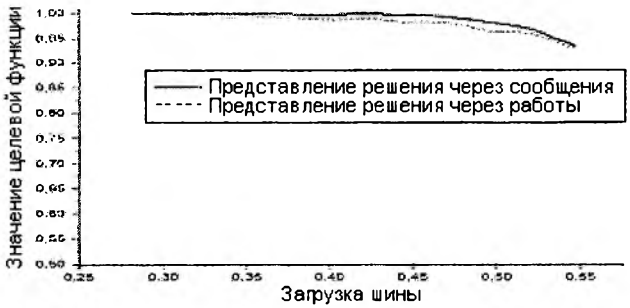


Рис. 1. Сравнение алгоритмов, разработанных на основе предложенных подходов на первом наборе данных.

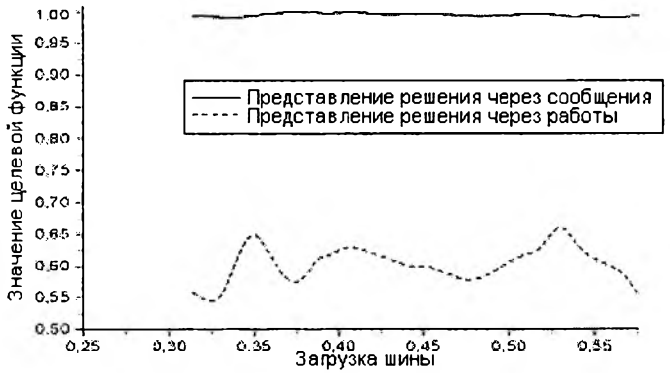


Рис. 2. Сравнение алгоритмов, разработанных на основе предложенных подходов на втором наборе данных.

8. Заключение

В работе описаны два подхода к сведению задачи построения расписания обменов по каналу с централизованным управлением к задаче нахождения пути на графе. Подход на основе соответствия вершин графа работам более полно представляет возможные последовательности работ, тогда как подход на основе соответствия вершин графа сообщениям позволяет лучше учесть ограничения на корректность расписания. По результатам исследований, алгоритм, основанный на втором подходе, получает более качественные решения.

Литература

1. ГОСТ Р 52070–2003. Интерфейс магистральный последовательный системы электронных модулей. – Введ. 01.01.2004 – М.: Изд-во стандартов, 2001. – 23с.

2. Guide to Digital Interface Standards for Military Avionic Applications: Technical report ASSC/110/6/2-ISSUE 3 // Avionics Systems Standardization Committee (ASSC). – UK, 2006. – 249 p.
3. Information technology – Fibre Channel – Part 312: Avionics environment upper layer protocol MIL-STD-1553B Notice 2 (FC-AE-1553): Technical Report TR 14165-312:2009 // International Organization for Standardization (ISO). – 2009. – 84 p.
4. Dorigo M. Optimization, Learning and Natural Algorithms// PhD Thesis. Dipartimento di Elettronica, Politechnico Di Milano, Milano. 1992.
5. Штовба С.Д. Муравьиные алгоритмы: теория и применение // Программирование. 2005. №4.
6. Marco Dorigo. Ant Colonies for the Traveling Salesman Problem. IRIDIA, Université Libre de Bruxelles. IEEE Transactions on Evolutionary Computation, 1(1):53–66. 1997.
7. Stuzle T., Dorigo M. ACO Algorithms for the Quadratic Assignment Problem. // New Ideas in Optimization, McGraw-Hills, 1999. P. 33-50.
8. Levine J., Ducatelle F. Ant Colony Optimization and Local Search for Bin Packing and Cutting Stock Problems. // Journal of the Operational Research Society, 2003.
9. Ritchie G. Static Multi-processor Scheduling with Ant Colony Optimization and Local Search. // Master's Thesis. University of Edinburgh, Edinburgh. 2003.
10. Blum C., Sampels M. Ant Colony Optimization for FOP Shop Scheduling: A case study on different pheromone representation // In Proceedings of the 2002 Congress on Evolutionary Computations, Honolulu. 2002.
11. Гафаров Е.Р. Гибридный алгоритм решения задачи минимизации суммарного запаздывания для одного прибора // Москва, ВЦ РАН. 2006.
12. Лущиков С. Алгоритмы планирования обменов по шине с централизованным управлением в вычислительных системах реального времени // Дипломная работа, Москва, 2008., 63 стр.
13. Костенко В. А. Алгоритмы построения расписаний для одноприборных систем, входящих в состав систем реального времени // Методы и средства обработки информации: Третья Всероссийская научная конференция. Труды конференции. - М.: Издательский отдел факультета ВМК МГУ имени М.В. Ломоносова; МАКС Пресс, 2009. - С.245-258.
14. Balashov V.V., Balakhanov V.A., Kostenko V.A., Smeliansky R.L., Kokarev V.A., Shestov P.E. A technology for scheduling of data exchange over bus with centralized control in onboard avionics systems // Proc. Institute of Mechanical Engineering, Part G: Journal of Aerospace Engineering. – 2010. – Vol. 224, No. 9. – P. 993–1004.

Щербинин В. В.

Инструментальная система построения алгоритмов распознавания нештатного поведения динамических систем

1. Введение

В работе [1] описана задача построения алгоритмов распознавания нештатного поведения динамических систем. В этой задаче рассматривается динамическая система, окруженная набором датчиков. Система может демонстрировать три типа поведения:

- Нормальное поведение.
- Нештатное поведение – поведение, при котором система в скором времени гарантированно перестанет выполнять заложенные в нее функции. Возможно несколько классов неисправностей, вызывающих нештатное поведение.
- Аварийное поведение – поведение, при котором система не выполняет заложенных в нее функций.

Данные о поведении системы представлены в виде фазовых траекторий в пространстве показаний датчиков, окружающих систему. Каждому классу нештатного поведения соответствует характерная фазовая траектория, называемая эталонной траекторией нештатного поведения.

В задаче распознавания нештатного поведения требуется построить алгоритм распознавания участков нештатного поведения в наблюдаемой траектории системы по набору прецедентов нормального и нештатного поведения системы. Рассматривается два вида наборов прецедентов:

- набор прецедентов с указанием участков нештатного поведения – в таком наборе прецедентов известны точные положения участков нештатного поведения в траекториях системы.
- набор прецедентов с указанием точек аварий – в таком наборе прецедентов известны только точки, в которых начинается аварийное поведение системы.

В работе [1] предложен аксиоматический подход к построению распознавателей нештатного поведения динамических систем, основанный на идеях алгебраического подхода к решению задачи выделения трендов [2]. В аксиоматическом подходе используются бинарные функции, называемые аксиомами. Аксиома ставит в соответствие точке траектории одно из значений "истина" или "ложь". Формируется набор аксиом, называемый системой аксиом. Каждой точке траектории ставится в соответствие аксиома, которая на ней

выполняется, т.е. принимает значение "истина". Полученная строка аксиом называется разметкой траектории. Распознавание нештатного поведения системы происходит путем поиска разметок эталонных траекторий нештатного поведения в траектории, в которой требуется провести распознавание.

Таким образом, при аксиоматическом подходе алгоритм распознавания включает в себя:

- алгоритм предобработки исходных данных,
- систему аксиом,
- алгоритм поиска разметок.

Для предобработки используются следующие алгоритмы: сглаживания, сжатия и интерполяции траектории на произвольный коэффициент, быстрое преобразование Фурье. Для поиска разметок используются алгоритмы на основе метрики Минковского и DTW (Dynamic Time Warping) [3]. Основную сложность при построении алгоритма распознавания на основе аксиоматического подхода представляет построение системы аксиом. Аксиома строится как булева формула над некоторым заранее заданным множеством элементарных условий [1].

На основе аксиоматического подхода был разработан ряд алгоритмов построения системы аксиом по набору прецедентов нормального и нештатного поведения системы:

- Для случая набора прецедентов с указанием участков нештатного поведения был разработан генетический алгоритм построения системы аксиом, описанный в работе [4].
- Для случая набора прецедентов с указанием точек аварий был разработан алгоритм построения системы аксиом, основанный на направленном переборе. Этот алгоритм описан в работе [5].

На основе алгоритма построения системы аксиом, использующего направленный перебор, был предложен автоматизированный метод построения системы аксиом по выборке с указанием точек аварий. Предложенный метод подразумевает взаимодействие с пользователем при построении системы аксиом, что позволяет добиться выигрыша (до 40 раз) по времени построения системы аксиом без потери точности распознавания. Этот метод описан в работе [6]. Для практического применения описанных алгоритмов и методов необходимо разработать программное средство, позволяющее решать задачу построения алгоритма распознавания нештатного поведения по набору прецедентов нормального и нештатного поведения динамической системы. К системе предъявляются следующие требования. Система должна:

- По заданному набору прецедентов с указанием участков нештатного поведения или с указанием точек аварий строить алгоритмы

распознавания нештатного поведения системы.

- Выделять участки нештатного поведения в предъявляемой фазовой траектории при помощи построенного ранее алгоритма распознавания.
- Предоставлять интерфейс для добавления новых элементарных условий и новых алгоритмов поиска разметок к уже реализованным в системе элементарным условиям и алгоритмам.
- Обладать кросс-платформенностью, т. е. возможностью работать на вычислительных системах под управлением различных операционных систем.
- Допускать параллельное выполнение алгоритмов обучения на кластерных вычислительных системах и обладать масштабируемостью по числу используемых SMP-узлов в кластере и по числу используемых вычислительных ядер в каждом узле.

Предоставлять графический интерфейс пользователя (ГПИ), позволяющий строить алгоритмы распознавания нештатного поведения в режиме диалога с пользователем в соответствии с автоматизированным методом построения алгоритмов распознавания нештатного поведения, описанным в [6].

2. Описание инструментальной системы

В данной работе приведено описание программной библиотеки AxiomLib, в рамках которой реализованы с учетом выдвинутых требований к инструментальной системе все описанные алгоритмы. Библиотека AxiomLib уже была описана в работе [1], однако на время составления этого описания не был разработан автоматизированный метод построения системы аксиом по набору прецедентов с указанием точек аварий и не был реализован ГПИ для построения системы аксиом по этому методу.

Общее описание

Инструментальная система написана на языке программирования C++ с использованием библиотеки boost [7], интерфейсов MPI [8] и OpenMP [9]. ГПИ для работы с библиотекой AxiomLib написано на языке программирования C++ с использованием библиотек Qt [10] и Qwt [11]. В качестве реализации интерфейса MPI используется библиотека MPICH2 [12]. Поддержка средств OpenMP обеспечивается за счет использования для сборки разработанного средства компиляторов, поддерживающих директивы OpenMP.

Использование интерфейсов MPI и OpenMP позволяет осуществить распараллеливание алгоритмов построения распознавателей, что, в свою очередь, позволяет достичь выигрыша по времени при запуске распараллеленных алгоритмов на кластерных

вычислительных системах. Подробно схема распараллелилвания алгоритмов построения распознавателей описана в [1]. Там же приведено исследование, показывающее, что реализованные алгоритмы построения распознавателей обладают масштабируемостью по числу SMP-узлов в кластере и по числу вычислительных ядер в каждом узле.

Структура разработанной библиотеки

Библиотека AxiomLib состоит из следующих модулей:

- Модуль основ распознавания и разметки. Данный модуль содержит классы системы аксиом, аксиомы и элементарного условия с основными функциями работы с ними.
- Модуль алгоритмов распознавания. Он содержит классы, соответствующие элементам алгоритма распознавания – алгоритмам предобработки исходных данных, алгоритму разметки и алгоритмам поиска разметок. Каждый из алгоритмов предобработки и алгортимов поиска разметок реализован в отдельном классе, отвечающем единому интерфейсу.
- Модуль генетического алгоритма построения распознавателя по выборке с указанием участков нештатного поведения. Этот модуль, в частности, содержит классы операций мутации, скрещивания и селекции, используемых в генетическом алгоритме.
- Модуль алгоритма построения распознавателя по выборке с указанием точек аварий. Этот модуль содержит классы, реализующие алгоритм построения распознавателя на основе направленного перебора.
- Модуль работы с окружением. Данный модуль содержит классы, позволяющие работать с наборами данных и параметрами алгоритмов обучения и распознавания.
- ГПИ для работы с библиотекой AxiomLib, позволяющий запускать и останавливать все реализованные в AxiomLib алгоритмы, а также просматривать их входные и выходные данные. Это ГПИ, за исключением компонента для полуавтоматического построения системы аксиом, описано в [1]. Компонент ГПИ, позволяющий осуществлять полуавтоматическое построение системы аксиом, описан ниже в данной работе.

ГПИ для полуавтоматического построения системы аксиом в режиме диалога с пользователем

Для запуска данного компонента ГПИ пользователю необходимо выбрать ManagedFuzzyMultiData из списка доступных алгоритмов в главном окне ГПИ для библиотеки AxiomLib, выбрать конфигурационных файл и набор данных и нажать кнопку «Старт».

После нажатия кнопки «Старт» появляется окно построения системы аксиом (см. рисунок 1), в котором отображаются стадии

построения системы аксиом. В меню окна построения системы аксиом есть пункты, позволяющие сохранять текущую стадию в файл или загружать ее из файла. Данное окно содержит четыре вкладки, каждая из которых соответствует шагу автоматизированного метода построения системы аксиом [6]:

1. Параметры 1-го этапа (см. рисунок 1).
2. Результаты 1-го этапа.
3. Результаты 2-го этапа.
4. Результаты 3-го этапа.

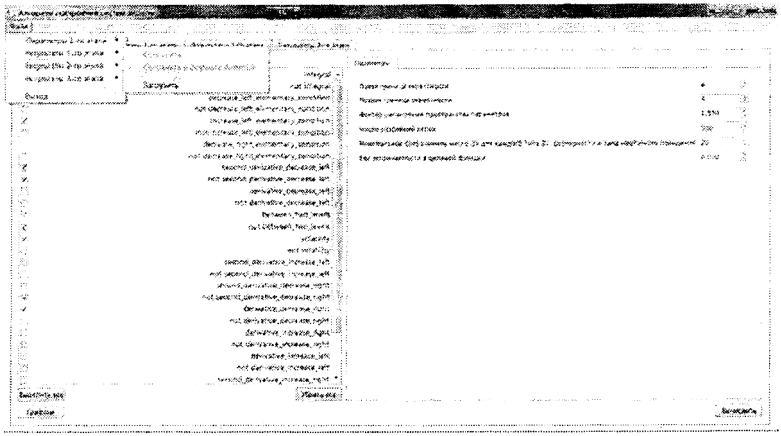


Рис. 1. Окно построения системы аксиом

Вкладки 2 – 4 активны, когда есть результаты соответствующего этапа алгоритма, которые либо вычислены (после того, как пользователь нажал кнопку «Вычислить» на предыдущей вкладке), либо загружены из файла (это можно сделать при помощи меню).

На вкладке «Параметры 1-го этапа» (см. рисунок 1) пользователь может произвести следующие действия:

- Выбрать типы элементарных условий, которые будут использоваться на дальнейших стадиях метода (список в левой части окна).
- Задать параметры первого этапа алгоритма (правая часть окна).
- Просмотреть графики обучающей выборки и указать более точное положение участков нештатного поведения в траекториях обучающей выборки (кнопка «Графики»).
- Запустить вычисление первого этапа алгоритма (кнопка «Вычислить»).

При нажатии кнопки «Графики» появляется окно просмотра графиков (см. рисунок 5). В этом окне пользователь может выбирать

траектории обучающей выборки, просматривать их и производить ограничение обучающей выборки. Об окне просмотра графиков будет подробнее рассказано ниже.

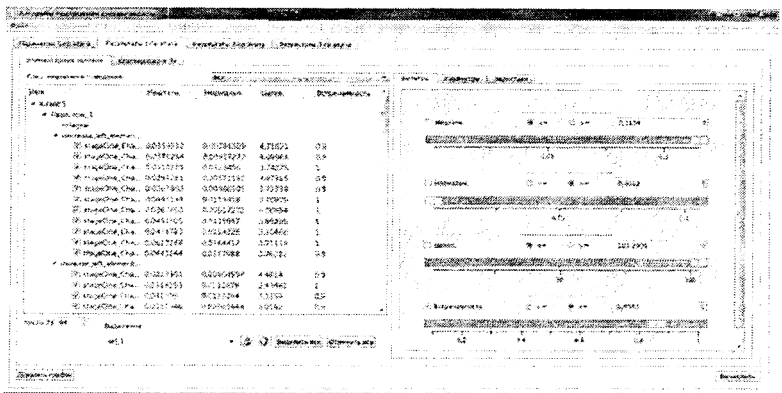


Рис. 2. Вкладка «Результаты 1-го этапа»

На вкладке «Результаты 1-го этапа» отображаются характеристики элементарных условий, полученных в результате работы 1-го этапа алгоритма построения системы аксиом (частота срабатывания на траекториях, соответствующих нештатному поведению; частота срабатывания на траекториях нормального поведения; целевая функция первого этапа алгоритма; значение встречаемости).

Работая с данной вкладкой, пользователь имеет возможность:

- Выбрать элементарные условия в соответствии с их характеристиками при помощи фильтров (вкладка «Фильтры»).
- Задать параметры следующего этапа алгоритма (вкладка «Параметры»).
- Задать параметры эвристик уменьшения числа элементарных условий и запустить эвристики (вкладка «Эвристики»).
- Посмотреть, как выполняются выбранные элементарные условия на траекториях обучающей выборки, используя окно просмотра графиков (кнопка «Графики»). Об окне просмотра графиков будет подробнее рассказано ниже.
- Провести кластеризацию выбранных элементарных условий и получить дополнительные характеристики элементарных условий, на основе которых можно сократить число элементарных условий (вкладка «Кластеризация»). О вкладке «Кластеризация» будет сказано ниже.
- Запустить вычисление 2-го этапа алгоритма с использованием выбранных элементарных условий, нажав кнопку «Вычислить».

На вкладке «Результаты 2-го этапа» (см. рисунок 3) отображаются характеристики аксиом, полученных в результате работы 2-го этапа алгоритма построения аксиом (частота срабатывания на траекториях, соответствующих нештатному поведению; частота срабатывания на траекториях нормального поведения; целевая функция второго этапа алгоритма; значение встречаемости).

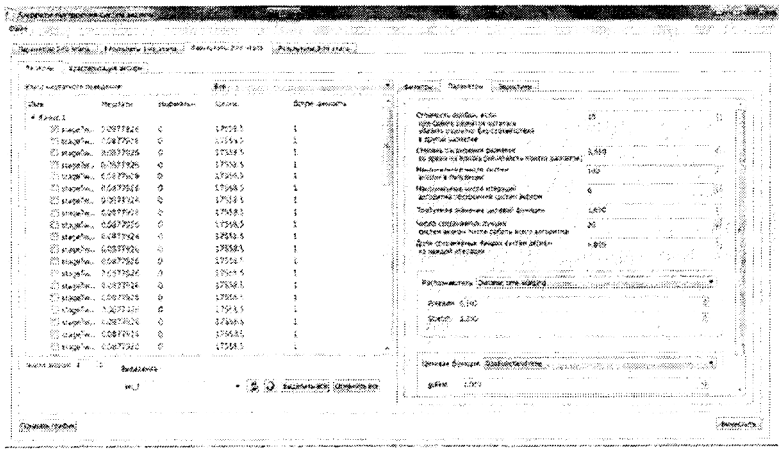


Рис. 3. Вкладка «Результаты 2-го этапа»

Работая с этой вкладкой, пользователь может:

- Выбрать аксиомы в соответствии с их характеристиками при помощи фильтров.
- Задать параметры эвристик уменьшения числа аксиом и запустить эвристики (вкладка «Эвристики»).
- Задать параметры следующего этапа алгоритма (вкладка «Параметры»).
- Посмотреть, как выполняются выбранные аксиомы на траекториях обучающей выборки, используя окно просмотра графиков (кнопка «Графики»). Об окне просмотра графиков будет подробнее рассказано далее.
- Провести кластеризацию выбранных аксиом и получить дополнительные характеристики аксиом, на основе которых можно провести сокращение числа аксиом (вкладка «Кластеризация»). О вкладке «Кластеризация» будет сказано ниже.
- Запустить вычисление 3-го этапа алгоритма с использованием выбранных аксиом (кнопка «Вычислить»).

На вкладке «Результаты 3-го этапа» (см. рисунок 4) показаны получившиеся в результате вычисления 3-го этапа алгоритма системы

аксиом и соответствующие им разметки эталонных траекторий нештатного поведения.

Рис. 4. Вкладка «Результаты 3-го этапа»

Пользователь имеет возможность:

- Просмотреть характеристики систем аксиом – число ошибок первого рода, процент ошибок второго рода, значение целевой функции.
- Просмотреть состав систем аксиом.
- Просмотреть и отредактировать по двойному щелчку указателя мыши разметки эталонных траекторий. При редактировании разметок пересчитываются значения характеристик системы аксиом.

При помощи окна просмотра графиков (см. рисунок 5) пользователь может получить наглядное представление о том, как выполняются элементарные условия или аксиомы на траекториях обучающей выборки.

В правой части данного окна находится список аксиом, соответствующих выбранному классу нештатного поведения (либо список всех выбранных для просмотра аксиом, если выбрано нормальное поведение). В правом верхнем углу – инструмент для выбора класса поведения и траектории. Слева находятся графики выбранной траектории с отмеченными на них аксиомами в точках их выполнения. Над графиками размерностей находится инструмент для ограничения траектории – с помощью него, двигая ползунки, можно указывать более точное положение участков нештатного поведения в траекториях обучающей выборки. Границы траектории отмечены на графиках размерностей красными вертикальными линиями.

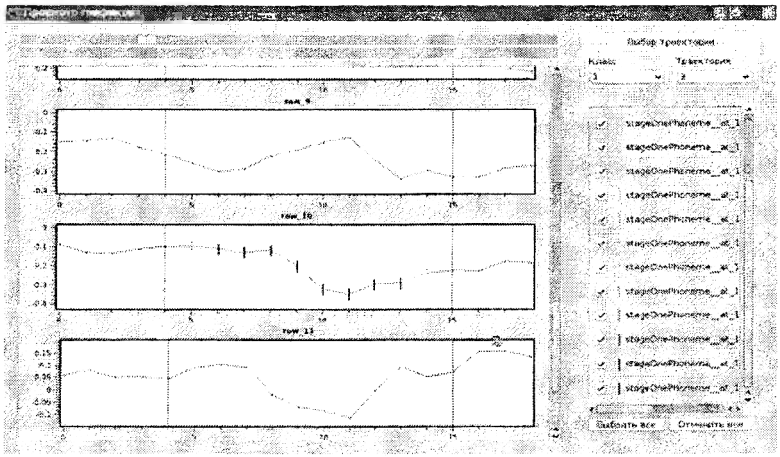


Рис. 5. Окно просмотра графиков

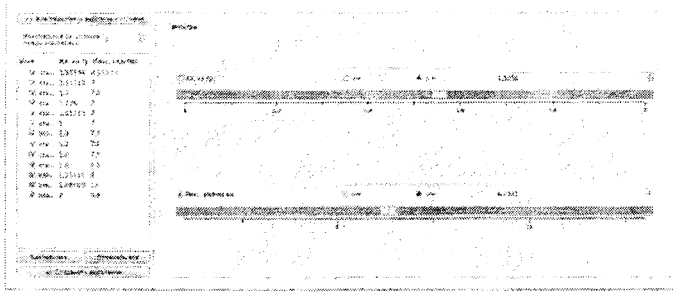


Рис. 6. Вкладка «Кластеризация»

Вкладка «Кластеризация» присутствует на вкладках «Результаты 1-го этапа» и «Результаты 2-го этапа» главного окна. На этой вкладке (см. рисунок 6) пользователь может производить кластеризацию выбранных аксиом, фильтровать их по параметрам, получаемым при кластеризации, и сохранять их выбор.

3. Заключение

В работе описана инструментальная система для решения задачи построения алгоритма распознавания нештатного поведения динамических систем по набору прецедентов нормального и нештатного поведения системы. Система удовлетворяет приведенным во введении требованиям и готова для эксплуатации.

Реализованная система позволяет строить распознаватели нештатного поведения динамических как в автоматическом режиме, так

и в режиме диалога с пользователем, что позволяет значительно сократить время построения алгоритма распознавания за счет использования экспертных знаний пользователя о системе.

Литература

1. Коваленко Д.С. Методы и программные средства обучения алгоритмов распознавания участков фазовых траекторий: дис. ... канд. физ.-мат. наук: 05.13.11 / МГУ им. М. В. Ломоносова. – М., 2001. – 168 с.
2. Рудаков К.В., Чехович Ю.В. О проблеме синтеза обучающих алгоритмов выделения трендов (алгебраический подход). // Прикладная математика и информатика N 8, М. Издательство факультета ВМиК МГУ, 2001. С. 97-114.
3. Keogh E.J., Michael J. Pazzani. Derivative Dynamic Time Warping. // Proceedings of the First SIAM International Conference on Data Mining (SDM'2001), Chicago, USA. 2001.
4. Kovalenko D., Kostenko V. A Genetic Algorithm for Construction of Recognizers of Anomalies in Behaviour of Dynamical Systems. // Proceedings of the IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications, IEEEPress, China, 2010. P. 258-263.
5. Коваленко Д.С. Метод автоматического построения алгоритмов распознавания участков фазовых траекторий. // Моделирование и анализ информационных систем, Т. 16, №4, 2009. Изд.: ЯрГУ. С. 6–21.
6. Коваленко Д.С., Щербинин В.В., Костенко В.А. Алгоритм и автоматизированный метод построения алгоритмов распознавания участков фазовых траекторий. Труды 15ой Всероссийской конференции Математические методы распознавания образов (ММРО-15), С196-200, М.:Макс Пресс, 2011.
7. Boost C++ Libraries [HTML] (http://www.boost.org/)
8. The Message Passing Interface (MPI) standard. // [HTML] http://www.mcs.anl.gov/research/projects/mpi/standard.html
9. Official OpenMP API specification for parallel programming. // [PDF] http://www.openmp.org/mp-documents/spec30.pdf
10. Qt Reference Documentation [HTML] (http://doc.trolltech.com/4.7/).
11. Qwt – Qt Widgets for Technical Applications [HTML] (http://qwt.sourceforge.net/).
12. Gropp W., Lusk E., Ashton D., Balaji P., Buntinas D., Butler R., Chan A., Goodell D., Krishna J., Mercier G., Ross R., Thakur R., Toonen B. MPICH2. // [PDF] http://www.mcs.anl.gov/research/projects/mpich2/documentation/files/mpich2-1.5-userguide.pdf

Раздел V
Сообщения

Брусенцов Н. П.

О выразимости отношения следования методом индексов Кэррола

В современной булевой алгебре, базирующейся на отношениях конъюнкции, дизъюнкции, отрицания, невыразимо отношение следования, обозначаемое общеутвердительным суждением Axy – "Все x суть y", "Сущность y целиком содержится в сущности x", "Из x необходимо следует y".

Льюис Кэррол в "Символической логике" [1, стр. 256] выразил следование $x \Rightarrow y$ методом индексов посредством конъюнкции $x_1 \wedge xy'_0$, в которой упущен член y'_1, а вернее сказать, нет необходимости в членах с индексом "1" и в самом этом индексе, означающем "существование", в противоположность символу "несуществования" – индексу "0". Ведь все термины в логике сосуществуют с их противоположностями, отмечаемыми штрихом, а несуществовать могут только их сочетания, т. е. индекс "0" означает несовместимость.

Например, сочетание xy'_0 выражает несовместимость сущности y' с сущностью x, обусловленную тем, что в каждой из них содержится противоположность другой: $x = xy$, $y' = x'y'$, причем обе они сосуществуют с их противоположностями. Поэтому несовместимость (кэрролово nullity) xy'_0 равнозначна следованию $(x \Rightarrow y)(y' \Rightarrow x')$, тогда как булево отрицание $\neg(xy') = x' \vee y$ составляет "материальную импликацию" – отношение, несоблюденное при исключенности y независимо от x, а также при существовании x независимо от y.

В ДНФ отношения следования $(x \Rightarrow y) \equiv xy \vee xy'_0 \vee x'y'$ содержатся конъюнкции xy и $x'y'$, конъюнкция xy' исключена индексом "0", а конъюнкция $x'y$ умалчивается как несущественная. Исключением ее следование превращается в эквивалентность:

$$(x = y) \equiv xy \vee xy'_0 \vee x'y_0 \vee x'y',$$

а включением – в материальную импликацию:

$$(x \supset y) \equiv xy \vee xy'_0 \vee x'y \vee x'y' \equiv x' \vee xy'_0 \vee y .$$

Очевидна трехзначность статуса членов ДНФ – включенность/исключенность/несущественность.

Литература

1. Кэррол Л. Символическая логика // Льюис Кэррол. История с узелками. – М.: "Мир", 1973.

Леонов М. В., Пенкин С. А., Егоренкова М. А.

Информационная система «Студенты Московского университета 1901-1902 учебного года»

Дорога в тысячу ли начинается с одного шага.
Китайская поговорка.

Введение и актуальность задачи

Задача преобразования в электронную форму информации, накопленной предыдущими поколениями, имеет комплексный характер и становится все более актуальной. Достаточно упомянуть проект Google Books, благодаря которому отсканированы миллионы книг, в том числе и на русском языке. Но для быстрого поиска и анализа данных простого сканирования недостаточно: необходима трудоемкая работа по структурированию текстов, например, с помощью технологии баз данных. Это относится и к старинным справочникам, не потерявшим своего значения и до сих пор. Среди таких книг «Алфавитные списки студентов И.М.У», которые Московский Университет выпускал в течение нескольких десятилетий XIX и XX века. Аналогичные книги выпускались и в других известных университетах мира. Эти книги сейчас стали библиографической редкостью, а потребность в хранящихся в них данных возрастает. К сожалению, в Московском университете в настоящее время отсутствует доступный источник о наших студентах XIX и начала XX века, так как дореволюционный Архив университета передан в ЦИАМ[1], да и там поиск обеспечивается средствами середины прошлого века. В лаборатории вычислительного практикума и информационных систем факультета ВМК предпринята попытка частично ликвидировать этот пробел. В рамках экспериментального проекта «Электронный имматрикуляционный[2] архив студентов Московского университета до 1917 года» разработана и заполнена данными информационная система по 1901-1902 учебному году, представленная в данном сообщении.

1. Особенности справочника, требования к системе, структура базы данных

Компьютеризуемый справочник [1] имеет табличную структуру со следующими строками (см. рис.1): фамилия, имя и

[1] Центральный Исторический Архив Москвы
[2] Имматрикуляция – внесение в студенческие матрикулы, зачисление в студенты

отчество студента с его порядковым номером, факультет и номер семестра, вероисповедание, звание, год рождения, место рождения, место предварительного образования, год поступления. Строки «вероисповедание», «звание», «место рождения», «место предварительного образования» содержат сокращения, причем не всегда стандартизованные. Например, «сын купца» иногда сокращался до «с. куп.», а иногда «с. купца». В графе «звание» чаще всего отражался социальный статус, например, «дворянин», «крестьянин», «сын статского советника», но иногда и другие признаки, например, «сын турецко-подданного», «житель города Тифлис». Поле «вероисповедание» также заполнено в справочнике не всегда стандартно. Например, наряду со значением «магометанин» встречается «мусульманин», кроме «иудей» есть и «еврей», и т.д. Аналогичные проблемы есть и с графой «место рождения». Иногда канцелярист записывал место рождения с точностью до губернии, иногда до города, иногда и до более мелкого населенного пункта.

Среди требований, предъявленных к разрабатываемой системе, можно выделить две основных. Первое: сохранить в электронной форме практически все сведения из соответствующего печатного антикварного издания. Второе: предоставить историкам (и другим заинтересованным исследователям) инструмент для научной с точки зрения историков обработки статистических данных.

Поэтому, согласно второму требованию, в структуру базы данных мы добавили поля для так называемых обобщенных атрибутов. Например, в результате консультаций с историком[3], выяснилось, что несколько десятков значений для поля «звание» можно свести до 8: дворянство потомственное, дворянство личное и служащее, духовенство, городское сословие, сельское сословие, военное сословие, иностранцы. Добавленное поле «обобщенной религии» имеет сейчас 4 значения: ислам, иудаизм, христианство, сектанты. (Буддистов среди студентов 1901-1902 года не оказалось). Для графы «место рождения» в качестве обобщенного поля мы ввели поле «Губерния», что позволит получать списки студентов по заданному названию губернии.

Заметим, что в поле «Место предварительного образования» записывалось не только сокращенное название гимназии, но иногда и специальные пометки типа «св. зр.» или «зол. мед.», означающие соответственно, что студент сдал испытания на «Свидетельство зрелости», а не учился в указанной гимназии, или удостоин золотой медали и т.п. Эти пометки хранятся у нас в соответствующих полях таблицы students.

Итак, из приведенных выше примеров ясно, что для того, чтобы сохранить в базе данных всю информацию печатного

[3] Дмитрий Алексеевич Гутнов, старший научный сотрудник Музея истории МГУ

справочника, а также обеспечить исследователя возможностями статистического анализа, необходимо было предусмотреть не только таблицы-словари для хранения атрибутов сущностей «СТУДЕНТ», «ФАКУЛЬТЕТ», «ВЕРОИСПОВЕДАНИЕ», «МЕСТО РОЖДЕНИЯ», «МЕСТО ПРЕДВАРИТЕЛЬНОГО ВОСПИТАНИЯ», но и таблицы для обобщенных значений для вероисповедания, места рождения, сословия (статуса). В итоге получилась структура, представленная на рис.2.

Имя, отчество и фамилия.	Факультет.	Вероисповедание.	Звание.	Год рождения.	Место рождения.	Место предварительного воспитания.	Год окончания.
957. Гринберг Моисей Гиршевич.	Юр. 1.	Иуд.	с. купца.	1881	Москва.	Москва. г.	1900
958. Гриценко Николай Прокофьевич.	Мат. 7.	Прав.	г. кун.	1874	Сумск.	Сумск. г.	1896
959. Гринцарь Александр Николаевич.	Юр. 3.	Прав.	с. кун.	1880	Москва.	Моск. 4 г.	1900
960. Гринцарь Николай Николаевич.	Мед. 1.	Прав.	с. кун.	1881	Москва.	Моск. 1 г.	1901

Рис. 1. Фрагмент страницы справочника

2. Архитектура, реализация, интерфейс

Информационная система «СТУДЕНТЫ МОСКОВСКОГО УНИВЕРСИТЕТА 1901-02 года» – это Web-приложение, написанное на PHP. При его работе используются СУБД MySQL, интерпретатор PHP и Web-серверный пакет MoWes Portable[3], а также Интернет-браузер. Благодаря выбранным средствам реализации оно обладает свойством мобильности, то есть его можно использовать непосредственно на флэш-носителе или на CD, без инсталляции на жестком диске компьютера. Это свойство полезно при работе в читальных залах, для участия в конференциях и т.д. Существенным аргументом в пользу архитектуры Web-приложения был планируемый в перспективе доступ к базе данных через Интернет. Такой подход уже был нами успешно опробован в нескольких проектах, в частности в работе «Информационная биобиблиографическая система по содержанию Журнала Министерства Народного Просвещения» [2].

Пункты основного меню (см. рис. 3) обеспечивают просмотр, добавление и редактирование любого элемента, для которого существует "словарь", то есть «Студент», «Факультет», «Вероисповедание», «Звание», «Место обучения», «Место рождения», а также "обобщенные элементы" ("base_religions", "base_classes", "base_locations") для вероисповедания, социального статуса, места рождения соответственно. Кроме того, возможно назначение реальных значений поля соответствующим значениям обобщенного поля. Например, значения «с.свящ.», «с.протоиерея», «с. диак.» значению «духовенство» поля "base_religions".

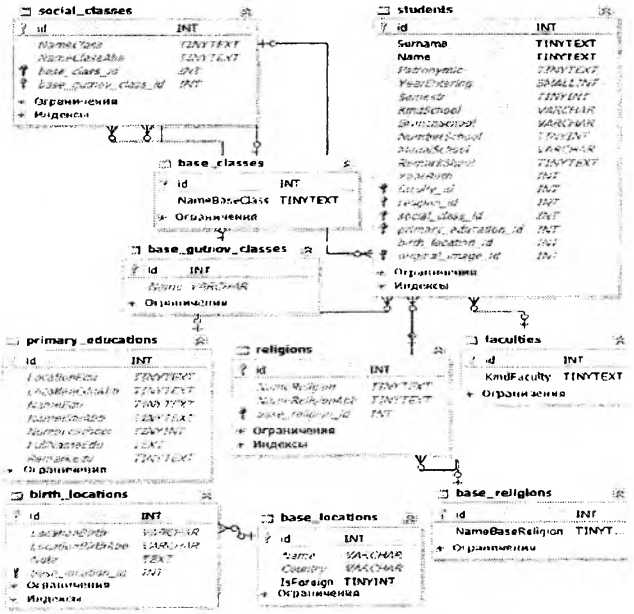


Рис.2. Схема базы данных

Учитывая, что у историков может быть разное представление об «обобщенных атрибутах» в зависимости от целей исследования, допускается изменять число и значения этих атрибутов. Запросы в актуальной версии системы реализуются в пункте "ПОИСК", где можно задать некоторый набор атрибутов, например, год рождения, фамилию студента (либо ее часть), год поступления, факультет, и получить список записей, удовлетворяющих указанным критериям.

Заключение

Представленная информационная система является первым шагом к созданию интегрированной БД в рамках проекта «Электронный имматрикуляционный архив по студентам Московского университета до 1917 года». База данных описанной системы содержит сведения по 4362 студентам четырех факультетов по состоянию на 25 декабря 1901 года. Из них 360 студентов числилось на историко-филологическом, 530 – на математическом, 682 – на естественном отделении физико-математического факультета, 1670 – на юридическом и 1120 – на медицинском факультете. Выбранный нами поэтапный подход компьютеризации архивных данных отдельно по академическим годам объясняется несколькими причинами. Во-первых, справочники по различным академическим годам содержат различный «репертуар»

атрибутов по каждому студенту. Во-вторых, необходимо было предусмотреть «распараллеливание» работ по проекту, учитывая огромный объем как технической работы, так и работы специалиста-архивиста, и деление работы по академическим годам здесь вполне оправдано. В-третьих, важно было относительно быстро получить реальный результат, имеющий самостоятельное значение. В-четвертых, уже давно в технологии баз данных существует подход, заключающийся в создании так называемых федеративных (или интегрированных) БД, именно его предполагается использовать на последующих стадиях проекта. В-пятых, принятое разделение на годичные БД дает возможность в порядке эксперимента безболезненно для других компонетов проекта расширять некоторые годичные БД дополнительными сведениями, например источниками из личных дел студентов, хранящихся в ЦИАМ и т.д.

В настоящее время система проходит опытную эксплуатацию с участием историков. Выяснилось, что нашу систему можно рассматривать и как своеобразное электронное пособие по социальной структуре России конца XIX века и истории Московского университета.

Авторы считают своим приятным долгом выразить искреннюю благодарность члену-корреспонденту РАН профессору Льву Николаевичу Королеву за помощь и поддержку проекта.

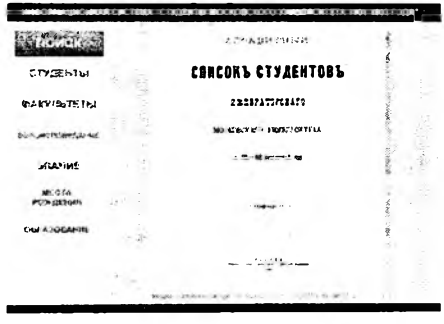


Рис. 3. Головная страница информационной системы

Литература

1. Алфавитный список студентов Императорского Московского Университета за 1901-902 академический год. М.: Университетская типография, 1901.
2. Гутнов Д.А., Леонов М.В., Пенкин С.А. Информационная биобиблиографическая система по содержанию «Журнала Министерства Народного Просвещения 1843-1917гг.» // Историческая информатика, № 1, 1012, стр. 444-444.
3. http://www.chsoftware.net/de/mowes/mowesportable/mowes.htm -Сайт компании CH Software.

Маслов С. П.

Троичная схемотехника

Схемотехника определяется как научно-техническое направление, охватывающего проблемы проектирования и исследования схем электронных устройств радиотехники и связи, вычислительной техники, автоматики и др.

Содержание понятия схемотехника зависит от уровня, на котором оно используется. В нынешней цифровой технике большинство разработчиков оперируют большими интегральными схемами и у многих может никогда не возникнуть необходимости реализовать какое-либо логическое устройство на уровне логических элементов. Разработчик располагает мощными средствами проектирования и разнообразными "полуфабрикатами". Для достижения результата достаточно уметь обоснованно выбрать необходимые компоненты и грамотно их сопрячь.

Технология интегральных схем за счет больших вложений и многолетних усилий доведена до совершенства. Намерения поменять ее элементную базу не получат поддержки – нет оснований проходить пройденное. Прорыв в этой сфере возможен лишь при появлении кардинально новых идей, сулящих многообещающие перспективы.

В недвоичной же технике вообще, и в троичной в частности, дело обстоит иначе. Никакого задела и никаких канонов здесь нет и чтобы довести дело до практического результата нужно пройти весь путь от логических элементов, через типовые узлы и устройства к сложным структурам.

Троичная Схемотехника (ТС), о которой говорится в статье, имеет в основе осмысление опыта, полученного при создании более 50-и лет тому назад троичных ЭВМ "Сетунь" [5]. Элементная база этих машин создавалась интуитивно - из желания воплотить на аппаратном уровне такие функции как, например, "Совпадение" или "Запрет". На более позднем этапе возникло понимание троичности, присущей элементам с индуктивными связями, и того как на их основе реализуется Троичная пороговая логика [4].

Автор имел цель: придумать функциональный аналог элемента "Сетуни", который можно осуществить на современной технической базе и попытаться осуществить на нем троичные логические структуры, используя наработки "Сетуней".

Запатентованы Пороговый Элемент Троичной Логики (ПЭТЛ) [1] и многоцелевой узел, состоящий из трех ПЭТЛ - Устройство Троичной схемотехники (УТС) [2]. ПЭТЛ является функциональным аналогом элементов "Сетуней", а его прототипом послужил элемент

ЭСЛ, однако использовать схемные решения "Сетуней" и приемы из арсенала ЭСЛ-схемотехники при создании троичных устройств не удается. Для этого потребовалась специальная Троичная Схемотехника (ТС).

Статья посвящена элементам, подходам, приемам и изобразительным средствам ТС. Устройство самих ПЭТЛ и УТС не обсуждается. Они описываются в терминах "черного ящика" .

Как ПЭТЛ, так и УТС имеют единственный вход X и несколько выходов. На вход в параллель поступают n дискретных сигналов, каждый из которых имеет одно из трех значений: +1,0 или –1 (на физическом уровне это дискретные токи $+I_ф$, 0 и $–I_ф$). На входе формируются значение, описываемые функцией $TS(X_1,.. X_{n-1},X_n)$:

$$TS(X_1,.. X_{n-1},X_n) = \begin{cases} +1 & \text{если } (n_{+1} - n_{-1}) > 0 \\ 0 & \text{если } (n_{+1} - n_{-1}) = 0 \\ -1 & \text{если } (n_{+1} - n_{-1}) < 0 \end{cases} \quad (1)$$

где n_{+1} - число сигналов, текущие значения которых +1

n_{-1} - число сигналов, текущие значения которых –1

Троичным значениям TS соответствуют двоичные значения (двузначные компоненты TS) на 4-х выходах ПЭТЛ (+R,+L,–R,–L) и 8-и выходах УТС (M1,M2,M3,M4,M5,M6,M7,M8). Соответствие значений на входах ПЭТЛ и УТС значениям на их выходах приведено в Таблице 1.

TS	+1	0	–1	TS	+1	0	–1	TS	+1	0	–1
+R	+1	0	0	M1	+1	0	0	M5	0	0	+1
+L	0	+1	+1	M2	0	+1	+1	M6	+1	+1	0
–L	–1	–1	0	M3	–1	–1	0	M7	0	–1	–1
–R	0	0	–1	M4	0	0	–1	M8	–1	0	0

Табл. 1. ПЭТЛ и УТС

Отметим, что хотя в Таблице 1 фигурируют троичные значения –1,0,+1, значения на выходах ПЭТЛ и УТС - двоичные: +1 или 0 на выходах +R,+L,M1,M2,M5,M6; –1 или 0 на выходах –R,–L,M3, M4,M7,M8. Как ПЭТЛ, так и УТС могут иметь несколько групп выходов. Текущие значения на одноименных выходах тождественны.

Преобразование (1) и объединение (сборка) выходов ПЭТЛ и УТС (двузначных компонент троичных значений) являются средствами ТС. Например, объединяя +R и –R, получим повторение входного троичного значения, объединяя +L и –L – его нециклическую инверсию, объединяя +R и +L или –R и –L - постоянные значения +1 или –1 для любого значения на входе. Допустимо и объединение выходов разных ПЭТЛ и УТС.

На схемах ТС сборкам соответствуют вертикальные линии, нижние концы которых помечены буквами, а верхние – наименованием

элемента, к входу которого данная сборка подключена, либо названием сигнала. Слева от сборок располагаются элементы, выходы которых к ней подключены, справа – единственный элемент, со входом которого она соединена.

Проиллюстрируем приемы и средства ТС на примере Троичного полусумматора [3]. Хотя рисунок является привычной формой изображения, в ТС более удобно использовать "табличную" форму. На Рис.1 полусумматор показан в привычном, а на Рис.2 – в табличном представлениях.

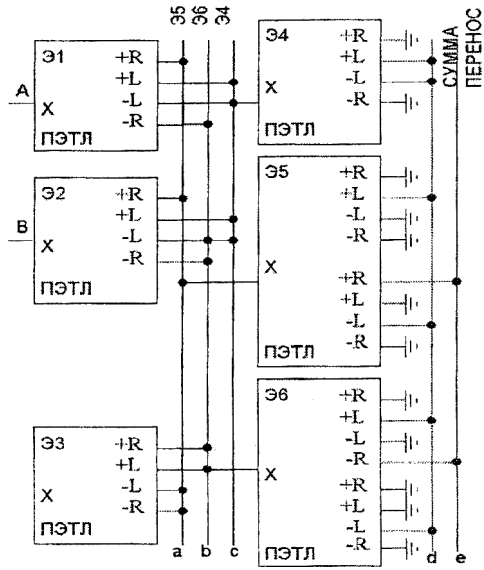


Рис. 1. Троичный полусумматор

В табличной форме сборки изображаются не вертикальными линиями, а столбцами таблицы, строки которой соответствуют выходам ПЭТЛ или УТС. На подсоединение выхода к сборке указывает черный кружок в ячейке, соответствующей этой сборке. Если в строке более одного кружка - подключены несколько тождественных выходов.

Табличная форма представления, не уступая рисунку в наглядности, упрощает заполнение таблиц истинности. Из нее легче перейти к цифровому описанию устройства. Снимается проблема присущая использованию рисунка: ПЭТЛ и УТС могут иметь более одной группы тождественных выходов и разработчик вправе брать выход из любой группы. Рисунок пришлось бы постоянно переделывать; в табличном же представлении всегда изображается

единственная группа, а о наличии других говорят несколько кружков в строке таблицы.

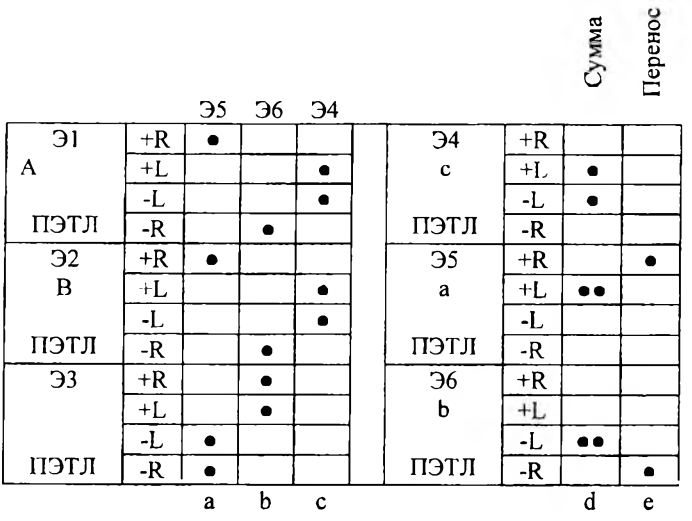


Рис. 2. Троичный полусумматор

Троичный полусумматор содержит 6 ПЭТЛ (Э1-Э6) и 5 сборок a,b,c,d,e. Слагаемые A и B поступают на входы ПЭТЛ Э1 и Э2, СУММА и ПЕРЕНОС снимаются с сборок d,e. Его функционирование описывается Таблицей 2.

A	+1	0	–1	+1	0	–1	+1	0	–1
B	+1	+1	+1	0	0	0	–1	–1	–1
a	+1	0	0	0	–1	–1	0	–1	–1
b	+1	+1	0	+1	+1	0	0	0	–1
c	–1	–1	0	–1	0	–1	0	+1	+1
d	+1	+1	0	+1	0	–1	0	–1	+1
e	+1	0	0	0	0	0	0	0	–1

Табл. 2. Троичный полусумматор

В ТС имеется процедура заполнения таблицы истинности. Хотя для такого простого устройства как полусумматор заполнить таблицу можно "вручную", в качестве примера опишем эту процедуру детально:

1. Фиксируем значения на входах: (например: A=–1, B=0);
2. По Таблице 1 определим для этих значений набор значений на подключенных к сборке выходах: (сборка a; ПЭТЛ Э1, выход

+R=0; ПЭТЛ Э2, выход +R= 0; ПЭТЛ Э3, выходы –L= –1, –R= 0).

3. К набору применим преобразование (1); результат поместим в ячейку таблицы, соответствующую сборке: (a = –1);
4. Перейдем к п.1; зададим другие значения A и B; выполним пп.2 и 3 для другой ячейки этой сборки. По завершению перейдем к другим сборкам.

Приведем другие примеры использования ТС. Троичные дешифратор и демультиплексор.

Троичный дешифратор 1x3 изображен на Рис.5. Дешифратор реализован на одном УТС, имеет управляющий вход C и три пары выходов: a,b c,d и e,f.

	M1				•		
	M2		•				
	M3					•	
C	M4			•			
	M5				•		
	M6						•
	M7	•					
УТС	M8			•			
		a	b	c	d	e	f

Рис.3 Дешифратор 1x3

Функционирование дешифратора описывается Таблицей 4.

C	+1	0	–1
a	0	–1	–1
b	0	+1	+1
c	–1	0	–1
d	+1	0	+1
e	–1	–1	0
f	+1	+1	0

Табл. 4. Дешифратор 1x3

Дешифратор 1x3 является составной частью многих устройств. В частности троичного демультиплексора, изображенного на Рис.4. Демультиплексор имеет один вход A, три выхода A(+1),A(0),A(-1) и управляется троичным дешифратором 1x3. Пары сборок a,b c,d и e,f дешифратора и демультиплексора – общие.

В зависимости от значения на входе С дешифратора на одном из выходов демультиплексора A(+1),A(0),A(-1) повторяется значение, имеющееся на входе A.

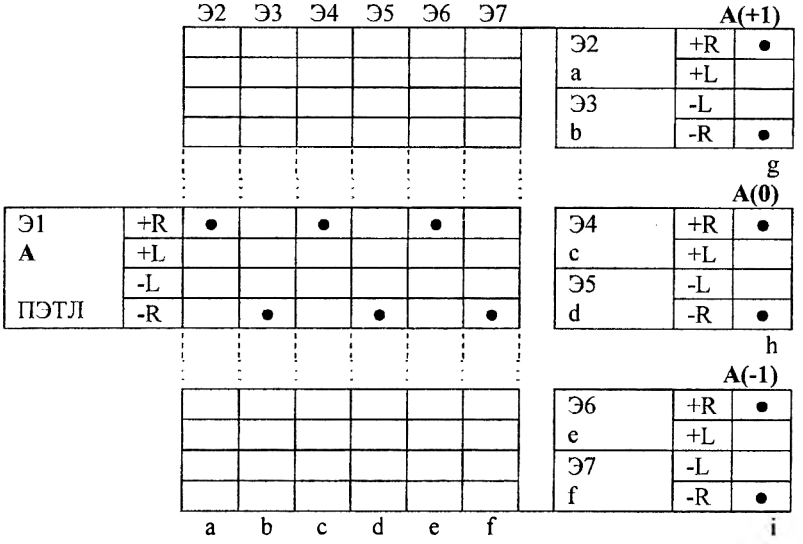


Рис. 4. Троичный демультиплексор 1x3

ПЭТЛ Э1 имеет 3 группы выходов. В ТС допускается частичное использование ПЭТЛ – либо только выходы +R,+L, либо только –L, –R. На Рис.4 частично используются ПЭТЛ Э2 – Э7.

Таблица истинности для узла дешифратор-демультиплексор имеет большие размеры. В рамках статьи ограничимся содержательным описанием его работы.

Для каждого значения С имеется единственная пара сборок, на которую от дешифратора поступают нули, благодаря чему на входы единственной из трех пар ПЭТЛ: Э4,Э5 Э6,Э7 Э8,Э9 поступают положительная (+A) и отрицательная (–A) компоненты входного значения. На входах других сборках будут либо 0, если +A=+1 или –A=–1, либо +1, если +A=0, либо –1, если –A=0. Выходы +R и –R пар ПЭТЛ подключены соответственно к сборкам g,h,i, являющихся выходами демультиплексора. На одной из них (той, на входы которой от дешифратора поступили нули) формируется текущее значение A.

Помимо рассмотренных здесь в качестве примеров использования ТС устройств, в патентах [1,2] описаны: "Троичная схема совпадений на 2 входа", "Генератор троичных констант", "Троичные дешифратор и демультиплексор 2x9". На момент подачи статьи на

стадии формальной экспертизы находится заявка на изобретение: "Троичные D-триггеры".

Литература

1. Маслов С.П. Пороговый элемент троичной логики и устройства на его основе. Патент РФ на изобретение: RU № 2278469 C1.
2. Маслов С.П. Узел троичной схемотехники и дешифраторы-переключатели на его основе. Патент РФ на изобретение: RU № 2461122 C1.
3. Маслов С.П. Об одной возможности реализации троичных цифровых устройств. // Программные системы и инструменты. Тематический сборник № 12. М.; Изд-во факультета ВМиК МГУ, 2011, С.222-227.
4. Брусенцов Н.П. Пороговая реализация трехзначной логики электромагнитными средствами. // Вычислительная техника и вопросы кибернетики. Вып.9. – М.: Изд-во Моск. ун-та, 1972. С.3-35.
5. Брусенцов Н.П., Жоголев Е.А., Маслов С.П., Рамиль Альварес Х. Опыт создания троичных цифровых машин. // Компьютеры в Европе. Прошлое, настоящее и будущее. – Киев: Феникс, 1998. С. 67-71.

Царёв Д. В.

Исследование и разработка системы мониторинга потоков корпоративной электронной текстовой информации[1]

Введение

Сейчас по разным оценкам [1, 2] неструктурированные данные составляют 80-85 процентов от всего объёма данных организаций, причём в подавляющем большинстве это текстовая информация, например, содержащаяся в деловых документах, отчётах, контрактах, электронной почте и т.п. Обычно выделяют следующие специфики функционирования корпоративной электронной текстовой информации:

1. Текстовая информация представляется в виде различных типов электронных документов, к ним можно отнести текстовые файлы (например, файлы форматов *doc*, *docx*, *pdf*), электронные текстовые сообщения (*e-mail*, *icq*).
2. Электронные документы, содержащие текстовую информацию, функционируют во множестве разнородных систем организации: компьютеры сети, внешние носители, серверы электронной почты.
3. Модификация типа представления текстовой информации, например, текстовый документ впоследствии может быть прикреплён к электронному письму, или изменён формат представления документа с *doc* на *pdf*.
4. Текстовая информация постоянно изменяется, и объём её растёт: например, создаются или принимаются по электронной почте новые документы, копируются и редактируются старые.

Таким образом, текстовая информация является разнородной (как по типу представления, так и по среде функционирования) и её объём постоянно растёт, поэтому организации вынуждены тратить огромные ресурсы на её управление для ведения своей эффективной работы [3-7]. Организациям требуются средства управления текстовой информацией, включающие обеспечение её безопасности, хранения, доступа к ней и её соответствия нормативным требованиям [5-7]. Понимание деловой ценности информации и процессов, происходящих с ней, позволит разрабатывать свои политики и применять их к различным типам информации, а также управлять рисками, связанными с использованием конфиденциальных данных и наличием неизвестной

[1] Работа выполнена при поддержке государственного контракта №14.514.11.4016 и грантов РФФИ 11-07-00616, 12-07-00585

или неконтролируемой информации [5]. Поэтому актуальным является разработка средств:

- предоставления данных о том, где и какие электронные документы организации функционируют, кто и как их использует;
- анализа содержимого электронных текстовых документов, которые позволят осуществлять быстрое ознакомление с текстовым содержимым документов (например, предоставляя аннотации документов), производить рубрикацию документов, а также выделять группы схожих документов для анализа большой выборки.

Для описания процессов происходящих с текстовыми документами в данной работе предлагается использовать понятие *потока текстовой информации* — последовательность изменений состояния электронного документа и описание операций вызвавших данные изменения. Для каждого типа электронного документа и среды его функционирования определён свой набор операций изменяющий его состояние. Например, для текстовых файлов на компьютерах сети и подключаемых внешних носителях это операции: создание, изменение, перемещение, удаление; для электронных сообщений, проходящих через почтовые сервера: получение, отправка, создание ответа, пересылка. Изменения электронного документа могут быть двух типов:

- *контентное* — изменение содержимого документа (например, редактирование текстового файла);
- *контекстное* — изменение атрибутов документа (например, изменение имени или пути текстового файла, адреса получателя при пересылки письма).

При контентном изменении поток включает в себя данные о содержимом документа до и после его изменения и об операции вызвавшей данное изменение, при контекстном изменении — только информацию об операции.

Консолидация потоков текстовой информации из различных источников, таких как компьютеры сети, внешние носители, серверы электронной почты, в единой базе данных предоставит пользователям информацию об интересующих их документах, включающую данные об операциях с документами и их содержимом. Кроме того, использование потоков позволит легко переходить от описаний операций с документом к его содержимому (и наоборот) в каждый момент жизненного пути документа.

Описания операций с документами являются структурированными данными и хорошо представляются в виде реляционной модели. Используя средства СУБД пользователь может строить различные отчёты, касающиеся операций с документами, а также просматривать содержимое документов в различные периоды их жизненных циклов. Например, можно строить отчёты о работе

сотрудников с документами, о действиях с интересующими документами, о передаваемых документах и т.п.

Вместе с информацией об операциях над документами также потоки включают и текстовое содержимое документов. Таким образом, остаётся большой класс собираемых данных, в котором скрыто огромное количество информации и из которого можно извлекать знания — *текстовое содержимое документов*.

Выявление знаний в тексте — это нетривиальный процесс обнаружения скрытой, ранее неизвестной, потенциально полезной информации из неструктурированных текстовых данных [8]. Выявлять знания можно как в текстовых документах по отдельности, так и в целом наборе документов. Примерами обнаруживаемой информации для текстового документа являются ключевые слова или его аннотация, которая, как правило, представляет набор его наиболее значимых предложений [9]. Для набора текстовых документов — результаты классификации (рубрикации) и кластеризации [10]. В общем случае текстовые данные являются неструктурированными и для выявления в них знаний используются методы *интеллектуального анализа текстовых данных* (англ. *text mining*). На практике в различных системах управления контентной информацией организаций широко используются средства информационного поиска и классификации документов [6, 7].

В данной работе представляется система, осуществляющая мониторинг потоков текстовой информации. Потоки в рассматриваемой системе собираются для текстовых документов на компьютерах под управлением ОС Windows и подключаемых к ним внешних носителях. Представляемая система является основой для дальнейшего выявления знаний в информационных потоках из различных источников данных, в том числе и почтовых серверов.

1. Мониторинг и консолидация информационных потоков

Целью мониторинга и консолидации информационных потоков корпоративной сети является построение централизованной базы данных, которая будет содержать информацию о контентных данных и их использовании. Более того, операции связанные с одним электронным документом и изменения его контента объединяются в один *«поток»*. Например, для текстовых файлов на компьютерах сети ведутся их полные жизненные пути, а электронные сообщения, проходящие через корпоративный почтовый сервер, группируются с ответами на них, формируя цепочки переписок.

Для достижения поставленной цели необходимо решение ряда задач:

1. *Мониторинг и сохранение данных.* Необходимо осуществлять мониторинг и сохранение потоков электронных документов для различных источников информации, таких как компьютеры сети, почтовые серверы, и т.п.
2. *Фильтрация данных.* Система должна собирать данные из разнородных источников информации, однако не вся информация, проходящая через источники, важна для организаций. Для разных организаций разные документы и операции над ними могут представлять интерес (например, операции связанные с действиями определённых пользователей).
3. *Консолидация данных.* Данные из различных источников информации необходимо объединять (консолидировать) в едином хранилище для составления целостной картины информационных потоков организации. Также консолидация информационных потоков необходима для последующего выявления в них знаний путём агрегированного анализа как операций с электронными документами, так и их текстового содержимого.
4. *Обработка содержимого электронных текстовых документов.* Вспомогательные задачи, например, выделение текста из файлов различных форматов, выделение прикреплённых документов из электронных писем и т.п. Такие операции обработки необходимы для последующего применения методов интеллектуального анализа текстовых данных.

Для решения подобного класса задач управления информацией из множества различных источников обычно применяется мультиагентный подход [11], заключающийся в установки *программных агентов* на каждый источник информации. В нашем случае агенты, устанавливаемые на соответствующие источники данных, как минимум должны выполнять следующие задачи: мониторинг операций с электронными документами, фильтрацию электронных документов, сохранение операций и содержимого электронных документов в виде потоков, передачу собранных данных в центральное хранилище (которое также можно рассматривать в качестве специального агента — *агент консолидации*). Кроме того, задачи обработка содержимого электронных текстовых документов также можно выполнять на агентах, обеспечивая тем самым высокое распределение нагрузки внутри системы. По существу данные агенты выполняют сбор, обработку и передачу данных, такие агенты в [11] отнесены к типу *информационных* — управление информацией из множества различных источников, в том числе и физически разных. В дальнейшем мы будем называть данные агенты *агентами мониторинга.*

Агенты мониторинга должны передавать собираемые данные об информационных потоках в единое хранилище — *агент консолидации*. Поэтому агент консолидации должен обеспечивать принятие данных от множества агентов мониторинга и помещать полученные данные в хранилище. Основной особенностью работы агента консолидации является необходимость параллельного получения данных от большого количества агентов (тысячи и даже десятки тысяч), что объясняется масштабами современных сетей. Поэтому возникает необходимость реализации:

1. Эффективного представления данных на агентах мониторинга для их последующей передачи и хранения на агенте консолидации;
2. Механизмов распределения нагрузки на сеть (стратегий передачи данных).

Далее в данном разделе приводится описание агента мониторинга информационных потоков электронных текстовых документов на компьютерах корпоративной сети под управлением ОС Windows и подключаемых к ним внешних носителях.

1.1 Мониторинг текстовых файлов на компьютерах и подключаемых внешних носителях

Поток текстовой информации — последовательность изменений состояния электронного документа и описание операций вызвавших данные изменения. Для текстовых файлов на компьютере и подключаемых внешних носителях это операции: создание, изменение, перемещение, удаление. В случае создания файла или его первой регистрации в системе агенту мониторинга достаточно сохранять его содержимое и путь, в случае перемещения сохранять новый путь, в случае изменения содержимого — новое содержимое, или, что лучше — только изменения содержимого. В случае удаления — просто пометить его как удаленный. После чего всю собранную информацию необходимо передавать в центральное хранилище.

На каждом компьютере, для которого производится мониторинг, необходимо решение агентом следующих задач:

1. *Мониторинг файловой системы*. Получение данных об изменениях в файловой системе локального компьютера для мониторинга операций с файлами и подключения внешних носителей;
2. *Фильтрация файлов и операций*. Для разных организаций разные файлы и операции над ними представляют интерес, поэтому необходимо иметь средство задания правил, по которым будут определяться требуемые файлы, которые мы будем называть «документами», и требуемые операций над ними;
3. *Сохранение операций с документами* и *сохранение содержимого документов*;

4. *Обработка содержимого электронных текстовых документов.* Выполнение набора необязательных задач по обработки содержимого копий документов;
5. *Передача данных об информационных потоках в центральное хранилище (агент консолидации).*

1.1.1 Мониторинг файловой системы

В ОС Windows используются специальные структуры данных ядра, называемые IRP пакетами (англ. *I/O Request Packet* — пакет запроса ввода/вывода), для обеспечения обмена данными между приложениями и драйвером, а также между драйвером и драйвером. Таким образом, обращение к файлам — это фактически формирование соответствующих IRP и посылка их драйверам файловой системы [12, 13].

Фильтрация IRP — это общий и универсальный механизм, его используют антивирусные мониторы, файловые компрессоры/декомпрессоры, файловые крипторы/декрипторы и т.д. Для реализации фильтрации IRP есть документированные возможности — написание драйвера и присоединение его к стеку драйверов файловой системы. Начиная с Windows XP SP2, возможно написание драйверов — *минифильтров* ФС [13], предназначенных специально для мониторинга и фильтрации IRP-ов ФС.

Для ведения потоков файлов, достаточно перехватывать операции открытия и закрытия, соответственно, IRP_MJ_CREATE и IRP_MJ_CLEANUP.

1.1.2 Фильтрация файлов и операций

Под электронным *документом* на локальном компьютере и подключаемых к нему внешних носителей будем понимать любой файл, удовлетворяющий заранее заданным свойствам (например: имя файла, путь к файлу), а также свойствам, касающимся операций над ним (например: имя процесса, имя пользователя, тип операции).

Задача определения, является ли файл *документом*, или нет, затруднена тем, что в различных организациях могут быть заданы разные свойства для определения *документов*. Соответственно, необходимо предоставить экспертам возможность задания правил, по которым будет производиться классификация. Для задания правил удобны так называемые скриптовые языки (англ. *scripting languages*), разработанные для записи «сценариев». В силу наибольшей распространённости был выбран язык Python [14], кроме того для Python имеется множество дополнительных свободных библиотек, в частности, для работы с различными форматами текстовых файлов, работы с кодировками и т.п. Python является интерпретируемым языком, но для него существует JIT-компилятор Psyco, позволяющий

транслировать исходный код в машинный во время первого запуска, что позволяет существенно увеличить производительность.

1.1.3 Сохранение операций с документами

Поступающую от драйвера информацию о работе с ФС, после прохождения через соответствующие фильтры, необходимо где-то сохранять. Кроме описаний самих операций нужно хранить связанные с ними данные о процессах и пользователях. Также необходимо осуществлять эффективную выборку данных для их последующей передачи в центральное хранилище (агент консолидации). Поэтому логично хранить *журнал операций* в легкой РСУБД, например, MS Access, тем более, библиотеки для работы с ней предустановленны практически на любой рабочей станции с ОС Windows. Использование клиент-серверной/иерархической СУБД не рассматривалось, т.к. установка дополнительного (тем более «тяжелого») ПО на локальный компьютер нежелательна.

Одной из проблем ведения потоков документов является отсутствие глобального идентификатора у файла в Windows, ассоциированного с ним от создания до удаления. Введение такого идентификатора на уровне драйверов ФС является достаточно сложным решением. Учитывая, что операция перемещения документов и директорий является редкой, более простым решением является ведение словаря *[полный путь к файлу]-[идентификатор]* в режиме пользователя и обновление его при перемещениях. Аналогично *журналу операций* представляется логичным реализовать словарь в легковесной встраиваемой реляционной СУБД.

Предлагается использовать одну из наиболее распространенных легковесных СУБД (MS Access, SQLite или MS SQL Compact Edition) с таблицей с тремя столбцами: *id*, *name*, *parent_id*, где *id* — первичный ключ, *parent_id* — внешний ключ к *id*, *name* — имя директории/файла. В этом случае для получения идентификатора по пути приходится делать до N запросов к базе, где N — уровень вложенности пути (решением этой проблемы может стать кэширование запросов к словарю). Такая схема позволяет эффективно обновлять словарь в случае перемещения директории, чего не скажешь о более простой с виду таблице *[полный путь к файлу]-[идентификатор]*.

Серия экспериментов, заключающихся в добавлении в словарь путей к 20.000 файлам и выполнением 20.000 запросов их идентификаторов, с рассмотренными подходами по реализации словаря *[полный путь к файлу]-[идентификатор]* показала, что наилучшие результаты получаются при использовании MS Access через интерфейс ODBC. Так же в подразделе 1.3 приводятся данные о скорости работы системы при реализации таблицы словаря *[id, name, parent_id]* в оперативной памяти.

1.1.4 Сохранение содержимого документов

В отличие от описания операций с документами содержимое самих документов, как правило, представляет гораздо больший объём данных, поэтому необходима эффективная организация его хранения. Содержимое документов сохраняется в случае операции создания документа или любого обращения к существующему документу, но который ранее не был зарегистрирован в системе, а также в случае последующих операций изменения содержимого документов. Однако сохранять каждый раз новое состояние документа нецелесообразно, т.к. достаточно сохранять лишь информацию о самих изменениях его содержимого.

Подход заключается в построении дельта-файлов апостериори — то есть в начале новое состояние полностью копируется в хранилище, и лишь затем применяется алгоритм построения «дельты» файла, после чего новое состояние удаляется. Так как дельта-файл, как правило, меньше целостного содержимого, таким образом достигается уменьшение размеров хранилища. Одним из таких алгоритмов является алгоритм *rsync*, разработанный австралийским программистом Эндрю Триджеллом [15]. Изначально реализованный в одноимённой утилите *rsync* для эффективной передачи структур (например, файлов) по коммуникационным соединениям в том случае, когда принимающий компьютер уже имеет отличающуюся версию этой структуры. Данный алгоритм позволяет эффективно строить дельта-файлы и в контексте рассматриваемой задачи. Для построения дельта-файла алгоритму не нужно хранить все предыдущее состояние целиком, достаточно сигнатуры.

В представляемой системе используется комбинированный подход, заключающийся в использовании инкрементального копирования с помощью утилиты *rsync* и сжатия накопленных данных архиватором *gzip* при передаче. Архиватор *gzip* [16] был выбран, т.к. предоставляет хорошую степень сжатия при высокой скорости работы, кроме того он имеет свободные реализации на всех популярных платформах. Организация эффективного хранения накопленных контентных данных также важна и с точки зрения уменьшения объёма передаваемых данных, т.к. все собираемые данные об информационных потоках с компьютеров должны, в конечном счете, передаться в центральное хранилище.

1.1.5 Передача данных об информационных потоках в центральное хранилище

Консолидация данных подразумевает передачу больших объемов собранной информации по сети. Так как существует ряд сетей, где скорость передачи невелика, а объем трафика имеет значение (например, при передаче данных из регионального филиала в

центральный офис), актуальным является наличие методов *сжатия передаваемых данных* и *планирования передачи данных*. Несомненно, требуются и механизмы *защиты данных*, передаваемых по сети, с целью гарантирования достоверности собранных и обрабатываемых данных, а так же минимизации рисков утечки информации.

От агента мониторинга требуется передача следующих собранных данных:

1. *Описание операций над документами.* Информация об операциях хранится на агенте в виде таблиц в СУБД MS Access (пункт 1.1.3), поэтому был реализован экспорт данных из СУБД средствами самой СУБД.
2. *Содержимое документов.* Содержимое документов хранится на агенте в виде файлов, при этом для организации эффективного хранения используется инкрементальное копирование с помощью утилиты *rsync* (пункт 1.1.4). Список файлов, требующийся для передачи, определятся на основе данных об экспортируемых операциях.

Пакет файлов, содержащий данные об операциях с документами и файлы содержимого документов, перед отправкой сжимается архиватором *gzip*. Для обеспечения распределения нагрузки на сеть и балансировки нагрузки на наблюдаемые компьютеры предложено организовывать передачу собранных данных по одной или нескольким из следующих стратегий:

1. *Фиксированными объемами данных.* Агент накапливает определенный объем информации или фиксированное количество записей в базе данных и затем передает их на сервер консолидации.
2. *Через равные промежутки времени.* Агент через равные промежутки времени передает все имеющиеся у него в локальном хранилище данные независимо от их объема.
3. *Немедленная передача.* Агент сразу же передает полученные данные при появлении каждой новой записи в журнале. Данная стратегия наиболее требовательная к ресурсам компьютера.

Указанный механизм реализован заданием параметров, описывающих максимально возможный объем не переданных данных и максимально возможный интервал времени, в течение которого агент может не передавать данные. Как только одно из максимально возможных значений достигнуто, все собранные данные помещаются в очередь на отправку.

Для обеспечения безопасности все передаваемые по сети данные шифруются с помощью криптографического протокола SSL. Применяется двусторонняя авторизация для невозможности подмены принимающей или передающей стороны. Протокол обеспечивает конфиденциальность обмена данными между клиентом и сервером,

использующими TCP/IP, причём для шифрования используется асимметричный алгоритм с открытым ключом [17].

1.2 Архитектура агента мониторинга

Агент мониторинга потоков электронных документов на компьютере и подключаемых к нему внешних носителях состоит из следующих основных программных компонент:

1. *Драйвер-минифильтр.* Получение данных об изменениях в файловой системе локального компьютера для мониторинга операций с файлами и подключения внешних носителей.
2. *Служба Windows мониторинга информационных потоков.* Служба состоит из двух параллельно выполняющихся нитей: *ведение журнала операций*, *теневое копирование документов.* Модуль ведения журнала операций обеспечивает: фильтрацию файлов и операций, сохранение операций с документами. Модуль теневого копирования документов выполняет первоначальное копирование документов в локальное хранилище агента. Для взаимодействия данных нитей используется *очередь на копирование*.
3. *Служба Windows обработки содержимого электронных документов.* Выполнение набора задач по обработки содержимого копий документов. Примерами подобных задач могут являться: извлечение текста из документов различных форматов, преобразование кодировки и т.п. Данные задачи не являются обязательными и выполняются в режиме ожидания с низким приоритетом (т.е. когда наблюдаемый компьютер простаивает). Для взаимодействия со *службой мониторинга информационных потоков* используется *очередь на обработку.*
4. *Служба Windows оптимизации хранения содержимого электронных документов.* Для собранных первоначальных копий документов служба строит сигнатуры и, в случае существования в хранилище предыдущих состояний, выполняет построение дельта-файла и удаление исходного документа. Выполнение оптимизации хранения содержимого электронных документов не является обязательной задачей агента. Для взаимодействия со *службой обработки содержимого электронных документов* используется *очередь на оптимизацию.*
5. *Служба Windows передачи данных.* Служба реализует планирование и передачу данных (пункт 1.1.5) на агент консолидации.

Связь между перечисленными компонентами агента мониторинга изображена на рисунке 1.

1.3 Характеристики работы агента мониторинга

Основными функциями агента мониторинга являются сохранение операций с документами и копирование содержимого

документов. Остальные функции, таки как обработка документов, оптимизация хранения не являются «критичными» и, в крайнем случае, могут вообще не выполняться на агенте. Функция передачи данных в данном контексте не рассматривается, т.к. она связана непосредственно с уже собранными и, возможно, обработанными данными и имеет независимую настройку параметров планирования передачи. Поэтому основными целями при реализации агента были быстрота и надёжность выполнения функций сохранения операций и копирования содержимого. Данные функции выполняет *Служба Windows мониторинга информационных потоков* (Рисунок 1).

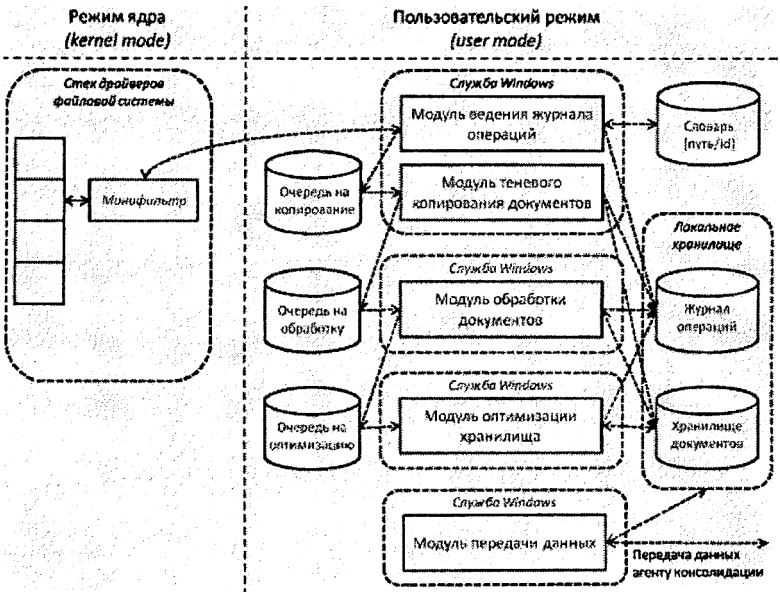


Рис. 1. Архитектура агента мониторинга

Информация, которая поступает в *Журнал операций*, должна сохраняться на жёстком диске компьютера, т.к. в случае сбоя или аварийного завершения работы по ней можно восстановить все необходимые данные: *Словарь [полный путь к файлу]-[идентификатор]* и данные для всех очередей (на копирование, на обработку, на оптимизацию хранения). Аналогично и с сохранением копий документов — их нужно сохранять на жёстком диске. Именно поэтому *Журнал операций* реализован в виде базы данных MS Access, а

копии документов хранятся в виде файлов в специальной директории ФС (*Хранилище документов*).

Таким образом, для ускорения обработки сообщений поступающих от *драйвера-минифильтра* к *Службе Windows мониторинга информационных потоков* можно реализовать *Очередь на копирование* и словарь *[полный путь к файлу]-[идентификатор]* в оперативной памяти. Для выбора оптимального решения была проведена серия экспериментов, заключающаяся в копировании большого количества новых (не зарегистрированных ранее агентом) документов на наблюдаемый компьютер с внешнего носителя, при этом *Локальное хранилище* агента и директория копирования документов физически находятся на одном диске.

В качестве тестового набора был выбран набор 20_newsgroups [18], который содержит примерно 20.000 документов. Размер данного набора составляет 43.9 Мб, но на диске он занимает 90.4 Мб, что обусловлено небольшим размером самих документов. Эксперименты проводились на виртуальной машине с ОС Windows Vista, 2 процессора, 2ГБ ОЗУ, жесткий диск 50ГБ, ФС NTFS (основная машина: Intel Core i7 2.2ГГцz, 8ГБ ОЗУ, жесткий диск 750ГБ, Windows 7 Pro).

Во время экспериментов замерялись: загрузка оперативной памяти, время операции копирования и время до появления всех документов в локальном хранилище агента. Полученные данные были усреднены за несколько итераций и приведены в таблице 1.

Словарь [полный путь к файлу]-[идентификатор]	*Очередь на копирование*	*Загрузка оперативной памяти (Мбайт)*	*Время процесса копирования (мин:сек)*	*Время до появления всех документов в хранилище (мин:сек)*
Деактивированный агент		-	2:58	-
Только сохранение операций		9-10	3:17	-
БД	БД	9-11	6:39	6:41
БД	Оперативная память	9-12	5:44	5:44
Оперативная память	БД	15-15	6:26	6:26
Оперативная память	Оперативная память	14-16	5:35	5:35

Табл. 1. Тестирование оптимизации агента мониторинга

Из приведённых в таблице 1 результатов тестирования различных реализаций агента, можно сделать следующие выводы:

1. Хранение словаря *[полный путь к файлу]-[идентификатор]* в памяти не даёт существенного прироста производительности (~3%). Кроме того, в случае не сохранения его состояния, его придётся восстанавливать путём просмотра всех зарегистрированных операции со всеми документами в *Журнале операций*;
2. Хранение *Очереди на копирование* в памяти увеличивает производительность примерно на 16%, при этом в случае сбоя состояние *Очереди на копирование* легко восстанавливается путём простого запроса о незавершённых операциях копирования к *Журналу операций*.

На основе полученных выводов было решено реализовать словарь *[полный путь к файлу]-[идентификатор]* в виде таблицы MS Access, а *Очередь на копирование* в оперативной памяти. Реализация *Очереди на обработку* и *Очереди на оптимизацию* в памяти не имеет смысла, т.к. сами операции обработки и оптимизации не являются обязательными и выполняются в режиме ожидания с низким приоритетом (т.е. когда наблюдаемый компьютер простаивает).

Стоит также отметить, что агент не занимает много оперативной памяти, в среднем около 12 МБ, но время копирования документов при работе агента увеличивается, что является допустимым, учитывая увеличение объёма копируемых файлов вдвое.

Защищенность собираемых данных обеспечивается средствами разграничения прав доступа ОС Windows: запрет доступа пользователей к *Локальному хранилищу* агента (изменение прав на доступ к соответствующей директории) и системным процессам.

Заключение

Статья посвящена мониторингу корпоративной электронной текстовой информации. Текстовая информация, функционирующая внутри организации, является разнородной (как по типу представления, так и по среде функционирования) и её объём постоянно растёт, поэтому организации вынуждены тратить огромные ресурсы на её управление. Для описания процессов происходящих с текстовыми документами в данной работе вводится понятие *потока текстовой информации* — последовательность изменений состояния электронного документа и описание операций вызвавших данные изменения. Консолидация потоков текстовой информации из различных источников, таких как компьютеры сети, внешние носители, серверы электронной почты, в единой базе данных предоставит пользователям информацию об интересующих их документах, включающую данные об операциях с документами и их содержимом. Кроме того, использование потоков позволит легко переходить от описаний операций с документом к его

содержимому (и наоборот) в каждый момент жизненного пути документа.

В данной работе была описана концепция системы выявления знаний в потоках корпоративной электронной текстовой информации. Кроме того, в рамках данной концепции была представлена мультиагентная система мониторинга информационных потоков электронных текстовых документов на компьютерах корпоративной сети под управлением ОС Windows и подключаемых к ним внешних носителях. Представляемая система является основой для дальнейшего развития её функционала:

1. добавление новых источников данных для сбора информационных потоков (например, корпоративные почтовые серверы);
2. добавление средств построения отчётов, касающихся операций с документами, и средств интеллектуального анализа текстовых данных для выявления знаний в содержимом документов.

Полезный эффект от разработки подобного класса систем будет заключаться в предоставлении организациям средств для решения ряда задач:

1. Выявление знаний о содержимом документов, перемещающихся в корпоративной сети и передаваемых за её пределы, и об операциях над ними;
2. Выявление знаний о действиях пользователей с документами в корпоративной сети;
3. Расследование инцидентов внутренней безопасности: мониторинг доступа пользователей к информации различного рода (в том числе конфиденциальной), снижение рисков потери или искажения информации за счёт сохранения копий документов.

Литература

1. Seth G. Unstructured Data and the 80 Percent Rule. - Clarabridge Bridgepoints, 3rd quarter 2008.
2. Inmon B. Structured and Unstructured Data, Bridging the gap. - B-eye-network, 2007. http://www.b-eye-network.com/view/4955
3. Gantz J.F. The Diverse and Exploding Digital Universe: An Updated Forecast of Worldwide Information Growth Through 2011. - IDC (sponsored by EMC), 2008.
4. Gantz J., Reinsel D. Extracting Value from Chaos. - IDC (sponsored by EMC), June 2011.
5. DuBois L., Tero V. Practical Information Governance: Balancing Cost, Risk, and Productivity. - IDC (sponsored by EMC), August 2010.
6. Childs Sh., Chin K. Magic Quadrant for Enterprise Information Archiving. - Gartner, 29 October 2010.
7. Панасенко А. Анализ рынка систем защиты от утечек

конфиденциальных данных (DLP) в России 2009-2011, http://www.anti-malware.ru/russian_dlp_market_2009_2011
8. Fayyad U., Piatetsky-Shapiro G., Smyth P. From Data Mining to Knowledge Discovery: An Overview. - Advances in Knowledge Discovery and Data Mining, AAAI Press : The MIT Press, Menlo Park, CA, 1996. - pp.1-34.
9. Mashechkin I., Petrovskiy M., Popov D., Tsarev D. Automatic text summarization using latent semantic analysis. - Programming and Computer Software. 2011. 299-305.
10. Tsarev D., Petrovskiy M., Mashechkin I. Supervised and Unsupervised Text Classification via Generic Summarization. - International Journal of Computer Information Systems and Industrial Management Applications, Volume 5 : MIR Labs, 2013. - pp. 509-515.
11. М. ван Стеен, Таненбаум Э. Распределенные системы. Принципы и парадигмы. - Питер, 2003.
12. Filtering IRPs and Fast I/O, http://msdn.microsoft.com/en-us/library/windows/hardware/ff540511(v=vs.85).aspx
13. Advantages of the Filter Manager Model, http://msdn.microsoft.com/en-us/library/ff538896.aspx
14. Python Programming Language, http://www.python.org
15. Rsync algorithm, http://rsync.samba.org/tech_report
16. GNU zip compression utility, http://www.gzip.org/
17. OpenSSL Project, http://www.openssl.org/
18. The 20 Newsgroups data set, http://people.csail.mit.edu/jrennie/20Newsgroups/

Аннотации

Антоненко В. А., Вдовин П. М., Волканов Д. Ю., Глонина А. Б., Захаров В. А., Зорин Д. А., Коннов И. В., Пашков В. Н., Подымов В. В., Савенков К. О., Смелянский Р. Л., Чемерицкий Е. В. Система имитационного моделирования РВС РВ, основанная на стандарте HLA, и методика ее использования // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 105-116.

В статье описывается архитектура системы имитационного моделирования РВС РВ, основанная на стандарте HLA, разработанной в лаборатории вычислительных комплексов факультета ВМиК МГУ. Приводится методика применения системы моделирования и перспективы её развития.

Ил.: 3 рис. Библиогр.: 24 назв.

Баранов М. И. Алгоритмы локального поиска для задач удовлетворения ограничений // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 16-26.

В статье изучается возможность применения метаэвристики локального поиска для решения задач удовлетворения ограничений. Кратко описываются сущность задач удовлетворения ограничений, особенности алгоритмов локального поиска и их применения к конкретным задачам. Предлагается эвристика, позволяющая ускорить процесс поиска решения. Более подробно рассматривается применение метаэвристики локального поиска для двух задач удовлетворения ограничений: задачи об n ферзях и задачи составления школьного расписания, оценивается эффективность разработанных и реализованных алгоритмов.

Ил.: 1 рис. Библиогр.: 4 назв.

Брусенцов Н. П. О выразимости отношения следования методом индексов Кэррола // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 146-146.

Рассматривается выразимость отношения следования методом индексов Кэрролла в виде ДНФ, полученной троичным обобщением:

$$(x \Rightarrow y) \equiv xy \vee xy'_0 \vee x'y'.$$

Библиогр.: 1 назв.

Бурцев А. А., Рамиль Альварес Х. Реализация средств объектно-ориентированного программирования в кросс-компиляторе языка ДССП-Т // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 27-36.

В НИЛ троичной информатики ВМиК МГУ создан программный имитатор ТВМ (Троичная Виртуальная Машина) и кросс-система ДССП-ТВМ для разработки программ для ТВМ на языке ДССП-Т.

В настоящей статье рассмотрены средства построения новых типов данных в ДССП, эквивалентные возможностям объектно-ориентированного программирования (ООП). Раскрыты секреты их реализации в ДССП-интерпретаторе. Обозначены проблемы реализации таких ООП-средств кросс-компилятором ДССП-ТВМ и предложены пути их преодоления.

В результате осуществлённой модификации кросс-компилятора в языке ДССП-Т обеспечены возможности построения новых типов данных, эквивалентные средствам объектно-ориентированного программирования.

Ил.: 1 рис. Табл.: 1 табл. Библиогр.: 9 назв.

Бурцев А. П., Курин Е. А. Исследование различных подходов к разработке параллельных алгоритмов моделирования распространения акустических волн для задач сейсморазведки // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 55-60.

Одним из методов поиска месторождений полезных ископаемых является сейсморазведка. Сейчас геофизики собирают сотни гигабайт данных при обследовании даже небольших площадей, а требуется исследовать сотни и сотни квадратных километров. Обработка такого объема полученных данных требует применения мощных компьютеров и больших затрат машинного времени, что часто невозможно в полевых условиях. Решением проблемы обработки больших объёмов данных может быть использование супер-вычислителей: многоядерных машин, кластеров, грид-систем.

В данной статье рассматриваются три основных комплекса программно-аппаратных средств для разработки параллельных решений: OpenMP для SMP архитектуры, MPI для кластеров и CUDA для графических ускорителей компании NVIDIA. Был разработан параллельный алгоритм моделирования распространения акустических волн в однородной среде. Цель данной работы — исследование эффективности различных подходов к разработке параллельного алгоритма решения данной задачи.

Табл.: 3 табл. Библиогр.: 5 назв.

Гришанин А. А. Численное моделирование задач для квантовых точек на суперкомпьютере // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 61-68.

Квантовые точки – это составляющие гетероструктуру наноразмерные кристаллы полупроводника. Свободные носители заряда удерживаются в области кристалла за счет потенциальных барьеров, окружающих область квантовой точки по всем трем направлениям. Если размер этой области меньше, чем длина волны электрона, то электронные состояния становятся квантовыми, с дискретными уровнями энергии, как у простого атома. Необходимость численного решения обусловлена потребностью нахождения энергетических уровней и волновых функций электронов в квантовой точке с заданной формой и размером, что позволило бы моделировать квантовую точку с заданными оптическими свойствами. В трехмерном случае, это задача, требующая объемных вычислений, что вызывает необходимость использования суперкомпьютера. Исследование показывает, что имеется возможность численно моделировать квантовые точки, используя супер-ЭВМ.

Ил.: 6 рис. Библиогр.: 7 назв.

Ершов Н. М. Неоднородные клеточные генетические алгоритмы // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 37-43.

В работе вводится в рассмотрение понятие неоднородного клеточного генетического алгоритма, в котором ряд параметров, влияющих на работу генетических операторов, делается зависимым от местоположения клеток заданного клеточного пространства. Приводятся результаты численного сравнения неоднородных клеточных генетических алгоритмов со стандартными вариантами генетических алгоритмов, показывающие преимущества предложенного подхода при минимизации мультимодальных функций с большим числом локальных экстремумов.

Ил.: 2 рис. Табл.: 3 табл. Библиогр.: 3 назв.

Жарков А. В. Разработка веб-интерфейса для суперкомпьютерного решения задач оптимизации развития транспортной сети // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 69-73.

Описан программный комплекс с веб-интерфейсом для проведения моделирования развития транспортной сети. Создание веб-интерфейса позволило использовать вычислительные мощности суперкомпьютера широкому кругу специалистов. Программный

комплекс был апробирован специалистами кафедры оптимального управления ВМК МГУ и позволяет решать практические прикладные задачи оптимального управления.

Ил.: 2 рис. Библиогр.: 3 назв.

Захаров В. Б., Махнычев В. С. О решении проблемы шахматных 7-фигурных окончаний на суперкомпьютере «Ломоносов» текстов // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 74-78.

В статье рассказывается о решении на суперкомпьютере задачи нахождения оптимальной стратегии для всех 7-фигурных шахматных позиций. Вычислительная сложность задачи составляет порядка 1020 операций, что потребовало больших усилий по оптимизации алгоритмов и кода программы. Особый интерес представляет параллельный алгоритм ретроанализа и алгоритм балансировки нагрузки между процессорными ядрами. Итогом работы явились таблицы с точными решениями 7-фигурных окончаний, занимающие более 100 ТБ на диске. Работа имеет как теоретическое значение в области параллельных алгоритмов, так и большое практическое значения для шахматных экспертов. Результаты работы уже используются для комментирования шахматных партий на крупнейших турнирах. Разработанные алгоритмы могут быть использованы и для решения других игр двух противников.

Библиогр.: 3 назв.

Зорин Д. А. Инструментальная система структурного синтеза вычислительных систем реального времени и построения расписаний // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 117-124.

В работе описана инструментальная система для решения задачи структурного синтеза вычислительных систем реального времени. Благодаря возможности подключения моделей вычислительных систем с разным уровнем детализации для оценки времени выполнения и надежности, систему можно использовать на различных этапах проектирования вычислительной системы. Расписание может быть перестроено на более позднем этапе по мере уточнения данных о заданиях, подлежащих планированию.

Ил.: 3 рис. Библиогр.: 7 назв.

Корж О. В. Анализ масштабируемости методов системы визуализации для суперкомпьютера «Ломоносов» // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 79-82.

Актуальным является разработка высокоуровневых средств программирования для петафлопсных вычислительных систем, а так же проектирование программного обеспечения для систем экзафлопсного диапазона. Для систем петафлопсного уровня вычислительной мощности проблема анализа полученных результат становится особенно острой, т.к. в существующих коммуникационных сетях не возможна полноценная передача данных на локальную машину пользователя. В данной статье были проанализированы основные особенности вычислительных систем такого уровня с точки зрения разработки для них систем визуализации больших данных. Было проведено тестирование разработанных ранее методов для системы визуализации с точки зрения возможности их масштабируемости при вычислительном эксперименте на нескольких тысячах ядер. Проведенный анализ позволяет сделать вывод, что применение существующих методов построения, компоновки и сжатия изображений в системах петафлопсного уровня не позволяет использовать преимущества вычислителя. Алгоритмы имеют низкую масштабируемость и требуют существенной оптимизации с точки зрения использования архитектурных особенностей построения коммуникационной сети и механизмом ввода вывода данных.

Ил.: 4 рис. Библиогр.: 6 назв.

Лихогруд Н. Н., Микушин Д. Н. KernelGen – прототип распараллеливающего компилятора C/Fortran для GPU NVIDIA на основе технологий LLVM » // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 83-96.

Проект KernelGen имеет цель создать на основе современных открытых технологий компилятор Fortran и C для автоматического портирования приложений на GPU без модификации их исходного кода. Анализ параллелизма в KernelGen основан на инфраструктуре LLVM/Polly и CLooG, модифицированном для генерации GPU-ядер и alias-анализе времени исполнения. PTX-ассемблер для GPU NVIDIA генерируется с помощью бекенда NVPTX. Благодаря интеграции LLVM-части с GCC с помощью плагина DragonEgg и модифицированного линковщика, KernelGen способен при полной совместимости с компилятором GCC генерировать исполняемые модули, содержащие одновременно CPU- и GPU-варианты машинного кода. В сравнительных тестах с компилятором PGI/OpenACC KernelGen демонстрирует большую гибкость по ряду возможностей, обеспечивая при этом сравнимый или незначительно более высокий уровень производительности.

Ил.: 4 рис. Библиогр.: 18 назв.

Леонов М. В., Пенкин С. А., Егоренкова М. А. Информационная система «Студенты Московского университета 1901-1902 учебного года» // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 147-151.

Сообщается о разработке информационной системы на основе содержания антикварного справочника «Алфавитный список студентов И.М.У. за 1901-1902 академический год». Система реализована в виде Web-приложения, функционирующего в среде Web-серверного пакета MoWes Portable с интерпретатором PHP и СУБД MySQL, и является составной частью проекта «Электронный имматрикуляционный архив студентов Московского университета до 1917 года».

Ил.: 3 рис. Библиогр.: 3 назв.

Малышко В. В., Манжосов А. Н. Решение задач инженерии знаний средствами объектно-ориентированной инженерии программного обеспечения // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 44-54.

В работе предлагается подход к решению задач инженерии знаний при помощи языка UML и универсальных инструментов разработки программных систем. Изложен способ представления знаний в виде UML-модели, описаны генераторы, создающие по UML-модели входные данные для планировщиков и спецификации для визуализации и валидации найденных планов. На примере рассмотрена симуляция выполнения плана по сгенерированной спецификации.

Ил.: 4 рис. Библиогр.: 7 назв.

Маслов С. П. Троичная схемотехника // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 152-158.

Обсуждаются возможности и средства Троичной схемотехники (ТС), позволяющей реализовывать троичные цифровые устройства на базе запатентованных Порогового элемента троичной логики (ПЭТЛ) и Устройства троичной схемотехники. (УТС). Описываются троичные полусумматор, дешифратор и демультиплексор.

Ил.: 4 рис. Табл.: 4 табл. Библиогр.: 5 назв.

Плакунов А. В. Способы сведения задачи построения расписания обменов по каналу с централизованным управлением к задаче нахождения на графе пути // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 125-135.

В статье приведено описание двух подходов к сведению задачи построения расписания обменов по каналу с централизованным управлением к задаче нахождения на графе пути. На основе предложенных подходов разработаны муравьиные алгоритмы для решения этой задачи. Проведено экспериментальное сравнение разработанных алгоритмов.

Ил.: 2 рис. Библиогр.: 14 назв.

Полякова И. Н., Крутов В. А. Разрешение неоднозначности в функциональной омонимии // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 97-104.

В настоящей статье рассматриваются прикладные проблемы разрешения функциональной омонимии в русском языке. Исследована задача автоматического разрешения функциональной омонимии двух типов:

- N*/A*, где N*-существительные, A* - полные прилагательные;
- N*/ Vf, где N*-существительные, Vf - личные формы глаголов.

Для решения задачи применен метод, основанный на контекстных правилах. Для данных типов функциональной омонимии приведены и исследуются упорядоченные наборы контекстных правил. Обсуждаются структурные характеристики минимальных контекстов. Контекстные правила реализованы на языке C#.

Библиогр.: 7 назв.

Рябов Г. Г., Серов В. А. Стандартная кубическая решетка R_c^n и биективное (генетическое) кодирование в ней // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 6-15.

В статье развиваются методы представления структур в стандартной кубической решетке R_c^n [1,2] в виде биективного кодирования на конечном алфавите [6]. В конечном итоге эти методы направлены на эффективные компьютерные реализации при хранении и вычислении топологических, метрических и комбинаторных характеристик таких структур для больших размерностей n.

Ил.: 2 рис. Библиогр.: 9 назв.

Царёв Д. В. Исследование и разработка системы мониторинга потоков корпоративной электронной текстовой информации // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 159-173.

Данная статья посвящена мониторингу корпоративной электронной текстовой информации. Для описания процессов

происходящих с текстовыми документами в данной работе вводится понятие потока текстовой информации — последовательность изменений состояния электронного документа и описание операций вызвавших данные изменения. Консолидация потоков текстовой информации из различных источников, таких как компьютеры сети, внешние носители, серверы электронной почты, в единой базе данных предоставит пользователям информацию об интересующих их документах, включающую данные об операциях с документами и их содержимом.

В данной работе приводится концепция системы выявления знаний в потоках корпоративной электронной текстовой информации. В рамках данной концепции представляется мультиагентная система мониторинга информационных потоков электронных текстовых документов на компьютерах корпоративной сети под управлением ОС Windows и подключаемых к ним внешних носителях. Представляемая система является основой для дальнейшего выявления знаний в информационных потоках из различных источников данных.

Ил.: 1 рис. Табл.: 1 табл. Библиогр.: 18 назв.

Щербинин В. В. Инструментальная система построения алгоритмов распознавания нештатного поведения динамических систем // Программные системы и инструменты. Тематический сборник № 13. М.: Изд-во факультета ВМиК МГУ, 2012. С. 136-145.

В статье приведено описание инструментальной системы, предназначенной для решения задачи построения алгоритмов распознавания нештатного поведения динамических систем по набору прецедентов нештатного и нормального поведения системы. Описаны основные интерфейсы и сценарии работы системы.

Ил.: 6 рис. Библиогр.: 12 назв.

Научное издание

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 13

Под общей редакцией
чл.-корр. РАН Л.Н. Королева

Подготовка оригинал-макета:
Глазкова Е.А.

Напечатано с готового оригинал-макета
Подписано в печать 17.12.2012 г.
Формат 60x90 1/16. Усл.печ.л. 11,5. Тираж 100 экз. Заказ 507.

Издательство ООО "МАКС Пресс"
Лицензия ИД N 00510 от 01.12.99 г.

119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус, 527 к.
Тел. 8(495)939-3890/91. Тел./Факс 8(495)939-3891.