© 2012 г. А.В. ПАНЮКОВ, д-р физ.-матем. наук
В.А. ГОЛОДОВ
(Южно-Уральский государственный университет)

ТЕХНИКА ПРОГРАММНОЙ РЕАЛИЗАЦИИ АЛГОРИТМА РЕШЕНИЯ СИСТЕМЫ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ С ИНТЕРВАЛЬНОЙ НЕОПРЕДЕЛЕННОСТЬЮ В ИСХОДНЫХ ДАННЫХ[1]

Рассматривается система линейных алгебраических уравнений $\mathbf{A}x = \mathbf{b}$, с интервальными матрицами $\mathbf{A}$ и $\mathbf{b}$. За множество решений принимается $\Theta_{tol}(\mathbf{A}, \mathbf{b}) = \{x : \mathbf{A}x \in \mathbf{b}\}$. Пусть $\Theta_{tol}(\mathbf{A}, \mathbf{b}(z)) = \{x : \mathbf{A}x = (1 + z)\mathbf{b})\}$, $z^* = \inf\{z : \Theta_{tol}(\mathbf{A}, \mathbf{b}(z)) \neq \emptyset\}$. Элементы множества $\Theta_{tol}(\mathbf{A}, \mathbf{b}(z^*))$ названы псевдорешениями. Доказано существование псевдорешения для любых интервальных систем линейных уравнений, предложен способ поиска псевдорешения с помощью решения соответствующей задачи линейного программирования. Вследствие вырожденности полученной задачи для ее решения необходимо использовать вычисления, обеспечивающие точность, намного превышающую возможности стандартных типов данных языков программирования. Симплекс-метод в сочетании с безошибочными дробно-рациональными вычислениями дает решение задачи. Для реализации используется крупнозернистый параллелизм в распределенных системах на основе MPI. Для реализации безошибочных дробно-рациональных вычислений используется техника программирования GPU с помощью CUDA C.

SOFTWARE ENGENDERING FOR ALGORITHM OF SOLVING A LINEAR EQUATION SET UNDER INTERVAL UNCERTAINTY / A.V. Panyukov, V.A. Golodov (South Ural State University, Lenina Ave., build. 76, Chelyabinsk 454080, Russia, E-mail: anatoly.panyukov@gmail.com). Considered is linear equation set $\mathbf{A}x = \mathbf{b}$ with interval matrixes $\mathbf{A}$, $\mathbf{b}$. Solutions are items of $\Theta_{tol}(\mathbf{A}, \mathbf{b}) = \{x : \tilde{A}x \in \tilde{b}\}$. Let $\Theta_{tol}(\mathbf{A}, \mathbf{b}(z)) = \{x : \mathbf{A}x = (1 + z)\mathbf{b})\}$, $z^* = \inf\{z : \Theta_{tol}(\mathbf{A}, \mathbf{b}(z)) \neq \emptyset\}$ be. Items of set $\Theta_{tol}(\mathbf{A}, \mathbf{b}(z^*))$ is termed as pseudosolutions. Existence of pseudosolution for all interval algebraic linear equation set is proofed in the paper, the way of pseudosolution searching by solving the corresponding linear programming problem is suggested. It is necessary computation guaranteeing sufficient accuracy well above standard data types of programming languages because of result of obtained problem degeneracy. Simplex method coupled with errorless rational-fractional computation gives effective solution of the problem. Coarse-grained parallelism for distributed computer systems with MPI is instrument of realization. CUDA C software engineering is suggested for errorless rational-fractional calculations.

[1]Работа выполнена при поддержке РФФИ, проект № 10-07-96003-р_урал_а

1. Введение

Система линейных алгебраических уравнений - это фундаментальный объект, который встречается при решении многих задач. Часто оказывается, что коэффициенты рассматриваемой системы не могут быть заданы точно, но известны интервалы, которым они принадлежат. В условиях такой интервальной неопределенности коэффициентов необходимо уточнение определения решения. В дальнейшем изложении будем использовать стандартную нотацию, принятую в интервальном анализе [1].

В работах [1] – [5] систематизированы подходы к учету интервальной неопределенности и дана их классификация. В соответствии с данной классификацией, AE-решением системы линейных алгебраических уравнений $\mathbf{A}x = \mathbf{b}$, в которой элементы матриц $\mathbf{A}$ и $\mathbf{b}$ представляют интервалы $\mathbf{a_{ij}} = \left[\underline{a}_{ij}, \overline{a}_{ij}\right]$, $\mathbf{b_j} = \left[\underline{b}_j, \overline{b}_j\right]$, $i, j = 1, 2, \ldots, n$, называют элементы допускового множества

$$\Theta_{tol}(\mathbf{A}, \mathbf{b}) = \left\{ x : \ (\forall i, j = 1, 2, \ldots, n)\, (\forall a_{ij} \in \mathbf{a_{ij}}) \left(\sum_{j=1}^{n} a_{ij} x_j \in \mathbf{b_i} \right) \right\},$$

EE-решением рассматриваемой системы уравнений называют точки объединенного множества

$$\Theta_{uni}(\mathbf{A}, \mathbf{b}) = \left\{ x : \ (\forall i, j = 1, 2, \ldots, n)\, (\exists a_{ij} \in \mathbf{a_{ij}}) \left(\sum_{j=1}^{n} a_{ij} x_j \in \ \mathbf{b_i} \right) \right\},$$

В работах [8, 9] доказано, что поиск EE-решения интервальной системы линейных уравнений является NP-трудной задачей. С другой стороны, в соответствии с теоремой Рона [10] любая точка допустимого множества AE-решений допускает представление в виде $x = x^+ - x^-$, где x^+, x^- являются решением системы неравенств

$$\begin{aligned} \sum_{j=1}^{n} \left[\underline{a}_{ij} x_j^+ - \overline{a}_{ij} x_j^-\right] &\geqslant \underline{b}_i, \ i = 1, 2, \ldots, n., \\ \sum_{j=1}^{n} \left[\overline{a}_{ij} x_j^+ - \underline{a}_{ij} x_j^-\right] &\leqslant \overline{b}_i, \ i = 1, 2, \ldots, n, \\ x^+, x^- &\geqslant 0. \end{aligned}$$

Следовательно, задача поиска AE-решений имеет полиномиальную сложность.

Методы оценки AE-решений для случаев $\Theta_{tol}(\mathbf{A}, \mathbf{b}) \neq \emptyset$ рассмотрены в работах [2] – [5] и др. Основным методом исследования допускового множества решений, развиваемым в Новосибирске [4]–[5], является «метод распознающего функционала». В нём для принятия решения о разрешимости или неразрешимости задачи (т.е. о пустоте/непустоте множества решений) необходимо поработать с некоторым специальным (негладким и вогнутым) функционалом, который назван «распознающим». При этом максимизация распознающего функционала, которую практически можно выполнять, например, с помощью различных методов негладкой оптимизации, разработанных в Институте кибернетики НАН Украины [6], даёт достаточно содержательную информацию для возможной коррекции задачи. Разработанные С.П. Шарым и П.И.Стецюком программы для исследования разрешимости интервальной линейной задачи о допусках (пустоты-непустоты допускового множества решений) имеются в свободном доступе на сайте [7].

Программы реализованы в INTLAB'е – интервальном расширении MATLAB'а, а также в Int4Sci – интервальном расширении Scilab'а.

Во многих практических задачах система неравенств (1) оказывается плохо обусловленными или вообще несовместной. В этом случае по аналогии с работами [11],[12] разумным представляется введение понятия «псевдорешения».

Целью данной работы является изложение анонсированного в работах [13]–[15] понятия «псевдорешение» для систем уравнений с интервальной неопределенностью и способы построения инструментальных программных средств их поиска.

2. Псевдорешение системы интервальных уравнений

Пусть дана система линейных алгебраических уравнений $\mathbf{A}x = \mathbf{b}$, в которой элементы матриц $\mathbf{A}$ и $\mathbf{b}$ представляют интервалы $\mathbf{a_{ij}} = \left[\underline{a}_{ij}, \overline{a}_{ij}\right]$, $\mathbf{b_j} = \left[\underline{b}_j, \overline{b}_j\right]$, $i, j = 1, 2, \ldots, n$.

Для заданной системы уравнений построим параметризованное семейство систем уравнений $\mathbf{A}x = \mathbf{b}(z)$ с модифицированной правой частью $\mathbf{b}(z) = \left[\underline{b} - z\left|\underline{b}\right|, \overline{b} + z\left|\overline{b}\right|\right], z \geqslant 0$.

Пусть $z^* = \inf\{z : \Theta_{tol}(\mathbf{A}, \mathbf{b}(z)) \neq \emptyset\}$. Псевдорешением исходной системы $\mathbf{A}x = \mathbf{b}$ будем называть внутренние точки допустимого множества $\Theta_{tol}(\mathbf{A}, \mathbf{b}(z^*))$.

Корректность введенного определения подтверждает

Теорема 1. Для любой системы интервальных уравнений $\mathbf{A}x = \mathbf{b}$ при всех $z > 1$ множество $\Theta_{tol}(\mathbf{A}, \mathbf{b}(z)) \neq \emptyset$.

Доказательство. В соответствии с теоремой Рона условие $\Theta_{tol}(\mathbf{A}, \mathbf{b}(z)) \neq \emptyset$ эквивалентно совместности системы линейных неравенств

$$\sum_{j=1}^{n} \left[\underline{a}_{ij} x_j^+ - \overline{a}_{ij} x_j^-\right] \geqslant \underline{b}_i - z\left|\underline{b}_i\right|, \quad i = 1, 2, \ldots, n., \tag{1}$$

$$\sum_{j=1}^{n} \left[\overline{a}_{ij} x_j^+ - \underline{a}_{ij} x_j^-\right] \leqslant \overline{b}_i + z\left|\overline{b}_i\right|, \quad i = 1, 2, \ldots, n, \tag{2}$$

$$x^+, x^- \geqslant 0. \tag{3}$$

Полагая в (1) – (3) $x^+ = x^- = 0$, получим

$$0 \geqslant \underline{b}_i - z|\underline{b}_i|, \quad 0 \leqslant \overline{b}_i + z|\overline{b}_i|, \quad i = 1, 2, \ldots, n.$$

Таким образом, для всех $z \geqslant 1$ имеет место включение $0 \in \Theta_{tol}(\mathbf{A}, \mathbf{b}(z))$.

Теорема доказана.

3. Способ поиска псевдорешения

Способ нахождения псевдорешения системы уравнений $\mathbf{A}x = \mathbf{b}$ дает

Теорема 2. Существует решение x^{+^*}, $x^{-^*} \in \mathbf{R}^n$, $z^* \in \mathbf{R}$ задачи линейного про-

граммирования

$$z \rightarrow \min_{x^+,\ x^-,\ z}, \tag{4}$$

$$\sum_{j=1}^{n} (\underline{a}_{ij} x_j^+ - \overline{a}_{ij} x_j^-) \geqslant \underline{b}_i - z|\underline{b}_i|, \quad i = 1, 2, \ldots, n, \tag{5}$$

$$\sum_{j=1}^{n} (\overline{a}_{ij} x_j^+ - \underline{a}_{ij} x_j^-) \leqslant \overline{b}_i + z|\overline{b}_i|, \ i = 1, 2, \ldots, n, \tag{6}$$

$$x_j^+, x_j^-, z \geqslant 0, \quad j = 1, 2, \ldots, n, \tag{7}$$

при этом $x^* = x^{+^*} - x^{-^*}$ *является псевдорешением системы* $\tilde{A}x = \tilde{b}$

Доказательство. Сначала докажем существование оптимального решения $x^{+^*},\ x^{-^*} \in \mathbf{R}^n,\ z^* \in \mathbf{R}$ задачи линейного пограммирования (4)–(7). Из теоремы 1 и теоремы Рона следует, что множество допустимых решений рассматриваемой задачи не пусто. Задача двойственная рассматриваемой имеет вид

$$\sum_{i=1}^{n} \underline{b}_i y_{1i} - \sum_{i=1}^{n} \overline{b}_i y_{2i} \rightarrow \max_{y_{1i}, y_{2i}}, \tag{8}$$

$$\sum_{i=1}^{n} \underline{a}_{ji} y_{1i} - \sum_{i=1}^{n} \overline{a}_{ji} y_{2i} \leqslant 0, \quad j = 1, 2, \ldots, n, \tag{9}$$

$$-\sum_{i=1}^{n} \overline{a}_{ji} y_{1i} + \sum_{i=1}^{n} \underline{a}_{ji} y_{2i} \leqslant 0, \ j = 1, 2, \ldots, n, \tag{10}$$

$$\sum_{i=1}^{n} |\underline{b}_i| y_{1i} + \sum_{i=1}^{n} |\overline{b}_i| y_{2i} \leqslant 1, \quad i = 1, 2, \ldots, n, \tag{11}$$

$$y_{1i}, y_{2i} \geqslant 0, \quad i = 1, 2, \ldots, n, \tag{12}$$

Легко заметить, что решение $y_{1i} = y_{2i} = 0, \quad i = 1, 2, \ldots, n$ является допустимым решением задачи (8)–(12). Таким образом, показано существование допустимых решений как у прямой, так иу и двойственной задач линейного программирования (задачи (4)–(7) и (8)–(12)). Из теоремы двойственности в линейном программировании следует существование у этих задач оптимальных решений.

Пусть $x^{+^*},\ x^{-^*},\ z^*$ оптимальное решение задачи (4)–(7). Из теоремы Рона следует, что $x^* = x^{+^*} - x^{-^*}$ является допустимым решением системы $\tilde{A}x = \tilde{b}(z^*)$. Из оптимальности z^* следует, что x^* является псевдорешением интервальной системы линейных уравнений $\mathbf{A}x = \mathbf{b}$.

Теорема доказана.

Таким образом, введенное понятие «псевдорешение»интервальной системы линейных уравнений является вполне конструктивными и позволяет давать результаты при решении интервальных систем в том случае, когда дпусковое множество $\Theta_{tol}(\mathbf{A}, \mathbf{b})$ допустимых решений пусто.

Однако, следует обратить внимание на высокую степень вырожденност задач (4)–(7) и (8)–(12)), что будет приводить при использовании приближенных вычислений к зацикливанию симплекс-метода. Избежать зацикливания можно за счет использования вычислений без округления [17], [16]. В этом случае на каждой итерации симплекс-метода количество требуемых бит памяти не превосходит величины $4lm^4 + O(lm^3)$, где

m – минимальная из размерностей задачи, l – число бит, достаточных для представления одного элемента матрицы исходных данных, при этом эффективность распараллеливания (т.е. отношение ускорения к числу процессоров) составляет в асимптотике величину близкую к 100 % [18].

4. Техника реализации

Итак, для успешной реализации изложенного способа поиска псевдорешения на компьютере понадобится несколько составляющих. С одной стороны необходимо обеспечить достаточную точность вычислений для преодоления зацикливания симплекс-метода. Подобная техника описана в [19]. Далее, для обеспечения эффективности всего алгоритма в целом, следует обеспечить достаточное быстродействие точных вычислений, а для использования параллельной архитектуры современных процессоров, произвести эффективную декомпозицию задачи линейного программирования на потоки.

4.1. Обеспечение необходимой точности

В рамках предыдущих исследований были созданы классы `overlong` [20] и `rational` [21], реализованные в объектно-ориентированной парадигме на языке C++, а также библиотека классов `Exact Computational` [17]. Данные классы позволяют производить безошибочные дробно-рациональные вычисления и имеют следующие характеристики. Объектами класса `rational` являются обыкновенные дроби p/q, где p, q - объекты класса `overlong`. Класс overlong предназначен для расширения логических возможностей целочисленных вычислений на компьютере. Объем памяти, занимаемый такими объектами, определяется значениями представляемых чисел, их диапазон ограничен только объемом адресуемой памяти. Диапазон чисел, представляемых объектами класса `overlong`, расширен до $(2^{-32*65535}, 2^{32*65535})$, а минимальный шаг дискретизации чисел, представляемых объектами класса `rational`, может достигать $(2^{-2097150})$. Для объектов классов `overlong` и `rational` определены все операторы, операции и бинарные отношения, используемые для стандартных числовых типов данных. Таким образом, классы `overlong` и `rational` дают потенциальную возможность использовать в программах пользователя безошибочное выполнение основных арифметических операций над полем рациональных чисел [22].

На сегодняшний день возможность использования безошибочных вычислений представляет известная библиотека GMP(The GNU Multiple Precision Arithmetic Library) [16]. Библиотека распространяется под лицензией GNU LGPL, актуальная версия библиотеки GMP 5.0.2 доступна для загрузки с официального сайта проекта. Программный код оптимизирован под большинство существующих процессорных архитектур, однако она *не предоставляет* своим объектам возможность их использования в распределенных вычислениях. Для более полноценного использования современных процессорных архитектур классы `overlong` и `rational` хранят и оперируют числами по основанию 2^{32}, код операций оптимизирован для системы счисления по основанию степень двойки. Оптимизации применяются также при работе с памятью, поскольку в C++ нет автоматического сборщика мусора, то лишние перевыделения памяти приводят её фрагментации и снижению быстродействия приложения в целом, краткое описание современных реализаций классов дано в [23]. Для облегчения сопровождения и модификации классов операции с памятью инкапсулированы в отдельный класс `MemHandle`, а выполнение базовых арифметических операций с данными полностью производится в рамках класса `ArifRealization` (см. фрагмент листинга 1).

```
class overlongNM {
 private:
  static ArifRealization realization;
 private:
 MemHandle mhandle;
 ...
 public: inline int32 size() const {return leng;}//leng of number
 public: inline int32 sign() const {return sgn;} //sign of number
 ...
 //addition
 template<typename Type>friend const overlongNM operator+
         (const overlongNM &num,Type v)
          {overlongNM rez(num); return (rez+=v);}
 friend const overlongNM operator+
       (const overlongNM&,const overlongNM&);
 ...
}
```

Листинг 1. Фрагмент класса `overlong`

Тем самым объект класса `overlong` содержит в себе объект типа `MemHandle`, и все действия с памятью происходят через интерфейс `MemHandle`. Все арифметические операции с данными осуществляются вызовом соответствующих методов класса `ArifRealization`. Пример реализации метода `add` класса `overlong` представлен в листинге 2.

```
void overlongNM::add(const overlongNM& alpha,const overlongNM& beta){
 d_t carry;
 const overlongNM& a=(alpha.size()>=beta.size())? alpha:beta;
 const overlongNM& b=(alpha.size()>=beta.size())? beta:alpha;
 int32 LA=a.size(),LB=b.size(),sg=alpha.sgn,newleng;//LA>=LB
 //вызов базовой арифметической операции
 ArifRealization::add(a.mhandle.getptr(),LA,b.mhandle.getptr(),LB,
                      mhandle.providetmpptr(LA,1),newleng,carry);
 mhandle.settmpasptr();
 if(carry) mhandle.safesetvalue(LA,carry);
 leng=newleng;
 sgn=sg;
}
```

Листинг 2. Организация операции сложения

Эта техника позволяет полностью абстрагироваться от реального места хранения данных (за это полностью отвечает реализация класса `MemHandle`) и способа реализации арифметических операций (конкретная методика выполнения операций над разрядами инкапсулирована в `ArifRealization`).

4.2. Обеспечение необходимой производительности

Вышеописанная техника разбиения класса `overlong`, на три части «интерфейс-память-арифметика», позволяет гибко использовать возможности вычислительной системы. Поскольку производительность алгоритма симплекс метода обеспечивается, в том числе, за счет эффективности реализаций классов `overlong` и `rational`, при их написании учтена возможность использования современных гетерогенных вычислитель-

ных сред с GPU ускорителями Nvidia (возможность задействовать GPU от компании AMD и встроенные графических ускорителей, например, Intel HD Graphics дает язык OpenCL, однако эксперименты показали большую трудоемкость процесса кодирования при малом приросте эффективности). Алгоритмы параллельного выполнения базовых арифметических операций, а также некоторые аспекты их реализации в гетерогенной среде описаны в [23, 24]. Хранение операндов организуется с учетом устройства, на котором производятся вычисления. Так, при наличии в системе GPU Nvidia, все данные (разряды) чисел можно хранить непосредственно на GPU, там же выполнять арифметические операции над числами, это снижает до минимума количество пересылок данных по PCI шине, остается лишь скопировать в основную память результаты вычислений. Для систем без GPU, вычисления проводятся на процессоре, данные хранятся непосредственно в оперативной памяти системы.

4.3. Мелкозернистый параллелизм

Параллельные вычисления на GPU требуют переработки алгоритмов базовых арифметических операций с учетом специфической архитектуры устройства. Ниже приводятся листинги некоторых операций для GPU Nvidia на расширениии языка C от Nvidia Corporation (CUDA C).

4.3.1. Сложение

Операция сложения длинных чисел на GPU осуществляется в несколько этапов: параллельное сложение разрядов, синхронизация, параллельное распространение переносов из разрядов. Особенность архитектуры GPU, а именно, выполнение нитей *блоками* и отсутствие синхронизации между блоками требует сохранения «пограничных»переносов во временом массиве (за это отвечает строка номер 16 в листинге 3 и последующего распространения (*DNumAdd_part2*) (листинг 4).

```
1  __global__ void DNumAdd_part1
2 (d_t *A,int32 LA,d_t *B,int32 LB, d_t *C,d_t *bGCarry,int32 *f){
3   int32 gId=blockDim.x*blockIdx.x + threadIdx.x;
4   if (gId >= LA) return; //bound check
5   int64 tmp=0;
6   if(gId>=LB) C[gId]=A[gId];
7   else{
8    tmp=(int64)A[gId]+(int64)B[gId];
9    C[gId]=tmp&MAX_DIGIT;
10  }
11  __syncthreads();//WorkGroup
12  //carry propagation in the block
13  int32 lId=threadIdx.x+1,i=gId+1,gS=blockDim.x;
14  for(tmp>>=BIT_IN_DIGIT;tmp && i<LA;lId++,i++){
15   if(lId==gS){bGCarry[blockIdx.x]=tmp; *f=1; return;}
16   tmp+=(int64)C[i]; C[i]=tmp&MAX_DIGIT;  tmp>>=BIT_IN_DIGIT;
17  }
18  if(i==LA && tmp) {bGCarry[Lcarry]=1;}
19 }
20 }
```

Листинг 3. Поразрядное сложение

```
__global__ void DNumAdd_part2
(d_t *C, d_t *bGCarry, int32 gS, uint32 Lcarry,uint32 LA){
 // //carry propagation between blocks
 int gId = blockDim.x*blockIdx.x + threadIdx.x,i=(gId+1)*gS;
 if (gId >= Lcarry)      return;
 uint64 tmp=(uint64)bGCarry[gId];
 for(;tmp && i<LA;i++){
  tmp+=(uint64)C[i];
  C[i]=tmp&MAX_DIGIT;
  tmp>>=BIT_IN_DIGIT;
 }
 if (i==LA && tmp) {bGCarry[Lcarry]=1;}
}
```

Листинг 4. Параллельное распространение переносов

Установка параметров и запуск кода на графическом ускорителе представлены на листинге 5. Этот код выполняется на стороне CPU, или, так называемой, стороне Host.

```
void ArifRealization::add(const d_t *A,int32 LA,
     const d_t *B,int32 LB,d_t *C, int32 &NL,d_t &Carry){
 int tPerBlock = 128,bPerGrid = (LA + tPerBlock - 1) / tPerBlock;
 d_t *bGCarry_d=NULL,*ansCarry_h=new d_t[1];
 int32 *cF_d=NULL,*cF_h=new int32[1];
 cudaMalloc((void**)&bGCarry_d, sizeof(d_t)*(bPerGrid+1));
 cudaMemset((void*)bGCarry_d, 0, sizeof(d_t)*(bPerGrid+1));
 cudaMalloc((void**)&cF_d, sizeof(int32));
 cudaMemset((void*)cF_d, 0, sizeof(int32));
 DNumAdd_part1 <<< bPerGrid, tPerBlock >>>
  (const_cast<d_t*>(A), LA, const_cast<d_t*>(B), LB, C, bGCarry_d,CF_d);
 cudaMemcpy(cF_h,cF_d,sizeof(int32), cudaMemcpyDeviceToHost);
 if(*cF_h){
  int gS=tPerBlock,LCarry=bPerGrid;
  bPerGrid = (bPerGrid + tPerBlock - 1) / tPerBlock;
  DNumAdd_part2 <<< bPerGrid, tPerBlock>>>
  (d_buffC, bGC_d, gS, LCarry, LA);
 }
 cudaMemcpy(ansCarry_h,&bGCarry_d[bPerGrid],
            sizeof(d_t), cudaMemcpyDeviceToHost);
 NL= (carry = *ansCarry_h)? LA+1: LA;
 delete[] ansCarry_h; cudaFree(bGCarry_d);  cudaFree(cF_d);
}
```

Листинг 5. Вызов функции для исполнения на GPU

4.3.2. Умножение

Операция умножения одна из наиболее затратных по времени. Для выполнения операции параллельного умножения используется быстрая разделяемая между нитями блока *__shared__* память. В реализации существенно используется особенности архитектуры GPU Nvidia, а именно, полностью синхронное выполнение инструкций в рамках одного *warp*. Это избавляет от необходимости выполнять синхронизацию между нитями. Исходный код для выполнения на GPU (*kernel*) представлен на листинге 6.

```
__global__ void DNumMult(d_t *A,int32 LA,d_t *B,int32 LB,d_t *rez)
{
 int32 lId=threadIdx.x,gId=blockDim.x*blockIdx.x + lId;
 if(gId>=LB) return;
 int32 cBS=(LA+blockDim.x-1)/blockDim.x;
 __shared__ uint64 sha[],shrez[];
 for(int i=lId*cBS;i<(lId+1)*cBS && i<LA;i++){
  sha[i]=A[i]; shrez[i]=0;
 }
 shrez[LA+lId]=0;
 uint64 digit=(uint64)B[gId], t=0UL;
 for(int i=0;i<LA;i++){
  t+=sha[i]*digit;
  shrez[i+lId]+=t&MAX_DIGIT;
  t>>=BIT_IN_DIGIT;
 }
 shrez[LA+lId]+=t&MAX_DIGIT;
 cBS=(LA+blockDim.x+blockDim.x-1)/blockDim.x;
 for(int i=lId*cBS;i<(lId+1)*cBS && i<LA+blockDim.x;i++){
  AtomicAdd(rez[i+gId],shrez[i]);
 }
}
```

Листинг 6. Умножение на GPU

Окончательное формирование результата происходит последовательным проходом по массиву **rez[]** и преобразованием 64-битного числа **rez[j]** в непосредственно разряд ответа и разряд переноса. Эти действия также выполняются на GPU, но в однопоточном режиме.

4.3.3. Деление

Поскольку стандартный алгоритм деления столбиком является абсолютно последовательным, для реализации в вычислительной системе с массовым параллелизмом, какой является GPU, эффективным является использование итеративных методов деления, выражающих результат через операцию умножения, подробнее данный подход подход изложен в [24].

4.4. Крупнозернистый параллелизм

Применяемая техника параллельной реализации симплекс метода детально описана в [18, 25]. В данном случае для разбиения задачи на потоки использовался механизм многопоточного програмирования **std::thread**, предоставляемый стандартом C++0x11.

5. Вычислительный эксперимент

Вычислительный эксперимент проводился на компьютере с процессором Intel Core i7-950 3.06 ГГц, 6 Гб ОЗУ, GPU Nvidia 460(1гб GDDR5), под управлением ОС Win 7 x64, в качестве компилятора был выбран 64-разрядный Visual C++ 2011.

В качестве модельной задачи использована система с матрицами:

$$\mathbf{A} = \left[\frac{i*(1-\delta)}{i+j-1}, \frac{i*(1+\delta)}{i+j-1}\right]_{n\times n}; \ \mathbf{b} = [1, 1/2, \ldots, 1/(n-1), 1/n]^T$$

Зависимость минимального расширения правой части (параметр z^*), соответствующего псевдорешению, при фиксированном значении $n = 20$ приведена в таблице (1) В таблице 2 приведены результаты времени затраченного на решение задачи для раз-

Таблица 1. Минимальное расширение правой части системы

δ	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
z^*	0.81	0.389	0.1	0.025	0.0062	0.0017

личных размерностей модельной интервальной системы.

Таблица 2. Время работы

Размерность матрицы (n)	10	20	50	100
Время работы	0.46с	7.73с	7.39м	15.1ч

6. Заключение

В работе полностью рассмотрена задача нахождения псевдорешения интервальной системы линейных алгебраических уравнений, дано определение понятия, показано существование псевдорешения для любой интервальной системы. Дано конструктивное доказательство существования псевдорешения как решения соответствующей задачи линейного программирования. Приведены подробности техники программной реализации симплекс-метода решения задачи линейного программирования, соответствующей псевдорешению интервальной системы, а также некоторые особенности реализации классов `overlong` и `rational`, обеспечивающих необходимую точность вычислений и, тем самым, позволяющих избежать зацикливания симплекс-метода в случае сильной вырожденности сформулированной задачи. Даны результаты численного эксперимента для различных смоделированных исходных данных. Дальнейшая работа направлена на получение более эффективных реализаций за счет оптимизации параллельной версии программы.

СПИСОК ЛИТЕРАТУРЫ

1. *Kearfott, R.B., Nakao M., et. all* Standardized notation in interval analysis. // Print clone is open at `http://www.mat.univie.ac.at/ neum/software/int`

2. *Neumaier A.* Interval Methods for Systems of Equations. – Cambridge: Cambridge University Press, 1990.

3. *Shary S.P.* Solving the linear interval tolerance problem // Mathematics and Computers in Simulation. – 1995. – Vol. 39. - P.53-85.

4. *Shary S.P.* A new technique in systems analysis under interval uncertainty and ambiguity // Reliable Computing. - 2002. - Vol. 8, №5 – P.321–418.

5. *С. П. Шарый.* Решение интервальной линейной задачи о допусках. // Автомат. и телемех., 2004, № 10, 147–162

6. *Стецюк П.И.* Субградиентные методы с преобразованием пространства для минимизации овражных выпуклых функций. // Международная конференция «Современные проблемы прикладной математики и механики: теория, эксперимент и практика», посвященная 90-летию со дня рождения академика Н.Н. Яненко», ISBN 978-5-905569-01-2/ – http://conf.nsc.ru/niknik-90/ru/reportview/37828

7. Интервальный анализ и его приложения – http://www.nsc.ru/interval

8. *Lakeyev A.V., Kreinovich V.* NP-Hard Classes of Linear Algebraic Systems with Uncertainties // Reliable Computing – 1997, №3. – pp. 51–81.

9. Lakeyev A.V., Kreinovich V. Optimal Solution of Interval Linear Systems is Intractable (NP-Hard) // Interval Computations/ – 1993, №1 – P. 6–14.

10. *Rohn J.* Inner solutions of linear interval systems. // Interval Mathematics. – 1985. – New York, Springer Verlag, 1986, pp. 157-158.

11. *Иванов В.К.* О линейных некорректных задачах. // ДАН СССР. – 1962. – Т.145. – №2 – С.270-272.

12. *Тихонов А.Н., Арсенин В.Я.* Методы решения некорректных задач. - М.: Наука, 1979. - 285 с.

13. *Панюков А.В., Чечулина Е.С.* Решения систем линейных алгебраических уравнений при интервальной неопределенности коэффициентов. // Алгоритмический анализ неустойчивых задач: Тез. докладов Всерос. конф., Екатеринбург, 2-6 февр. 2004 г. - Екатеринбург: Изд-во Урал. ун-та, 2004, стр., 291-292.

14. *Панюков, А.В., Чечулина Е.С.* Псевдорешения систем линейных алгебраических уравнений при интервальной неопределенности коэффициентов. / А.В. Панюков, Е.С. Чечулина // Интервальный анализ. Труды XIII Байкальской международной школы-семинара "Методы оптимизации и их приложения Иркутск, Байкал, 2 – 8 июля 2005 г. Том 4. – Иркутск: ИСЭМ СО РАН. – 2005. – С. 62 – 65.

15. *Панюков А.В., Голодов В.А.* Вычисление псевдорешений систем линейных алгебраических уравнений с интервальной неопределенностью коэффициентов // Алгоритмический анализ неустойчивых задач: тез. Докл. Международной конференции, посвященной памяти В.К. Иванова, Екатеринбург, 31 октября – 5 ноября, 2011. – Екатеринбург: ИММ УрО РАН. – С.

16. GNU Multiple Precision Arithmetic Library. – http://swox.com/gmp/

17. *Панюков А.В.,Германенко М.И.,Горбик В.В.* Библиотека классов "Exact Cjmputational"/ Свидетельство о государственной регистрации программы для ЭВМ № 2009612777 от 29 мая 2009г. // Программы для ЭВМ, базы данных, топологии интегральных микросхем. Официальный бюллетень Российского агентства по патентам и товарным знакам № 3. – 2009. – С. 251.

18. *Панюков А.В., Горбик В.В.* Применение массивно-параллельных вычислений для решения задач линейного программирования с абсолютной точностью // Автоматика и телемеханика. – 2012. – №2. – С. 73 – 88.

19. *Схрейвер А.* Теория линейного и целочисленного программирования: В 2-х т. Т. 1: Пер с англ. - М.: Мир, 1991. - 360 с.

20. *Панюков А. В., Силаев М. М.* Класс overlong. // Программы для ЭВМ. Базы данных. Топологии интегральных микросхем. – Официальный бюллетень Российского агентства по патентам и товарным знакам. №4(29), 1999 г. М.: ФИПС. - 1999. - Рег. № 990489. - С. 17.

21. *Панюков А. В., Силаев М. М., Германенко М. И.* Класс rational. // Программы для ЭВМ. Базы данных. Топологии интегральных микросхем. - Официальный бюллетень Российского агентства по патентам и товарным знакам. №4(29), 1999 г. М.: ФИПС. - 1999. - Рег. № 990607. - С. 97.

22. *Голодов В.А.* Адаптация библиотеки "Exact Computational"для гетерогенных вычислительных сред. //Научный поиск [текст]: материалы четвертой научной конференции аспирантов и докторантов. Естественные науки. - Челябинск: Издательский центр ЮУрГУ, 2012.

23. *Голодов В.А.* Распределенные символьные дробно-рациональные вычисления на процессорах x86 и x64. //Параллельные вычислительные технологии (ПаВТ'2012)[Электронный ресурс]: труды международной научной конференции (Новосибирск, 26 марта - 30 марта 2012 г.). – Челябинск: Издательский центр ЮУрГУ, 2012. 774 с., ISBN 978-5-696-04237-4.

24. *А.В. Панюков, С.Ю. Лесовой.* Реализация базовых операций целочисленной арифметики в гетерогенных системах // Параллельные вычислительные технологии (ПаВТ'2012): труды международной научной конференции (Новосибирск, 26 - 30 марта 2012 г.). Челябинск: Издательский центр ЮУрГУ, 2012. – С. 634–637. – ISBN 978-5-696-04237-4 http://pavt.susu.ru/2012/short/217.pdf

25. *Панюков А.В.,Горбик В.В.* Параллельные реализации симплекс-метода для безошибочного решения задач линейного программирования // Вестник Южно-Уральского государственного университета. Серия: «Математическое моделирование и программирование». – №25. – 2011. – С. 107 – 118