ПРОГРАММНО-ТЕХНИЧЕСКИЕ КОМПЛЕКСЫ

Ф.И. АНДОН, А.Е. ДОРОШЕНКО, А.Г. БЕКЕТОВ, В.А. ИОВЧЕВ, Е.А. ЯЦЕНКО

УДК 681.3

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА АВТОМАТИЗАЦИИ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ НА ОСНОВЕ АЛГЕБРЫ АЛГОРИТМОВ

Аннотация. Предложено развитие алгеброалгоритмической методологии и инструментария для автоматизированного проектирования и генерации программ для графических ускорителей. Особенностью предложенного подхода является использование языковых конструкций, близких к естественному языку, а также применение метода, который обеспечивает синтаксическую правильность проектируемых алгоритмов и программ. Подход реализован в инструментальной системе, предназначенной для диалогового конструирования схем алгоритмов и генерации программ. Применение инструментария проиллюстрировано на примере разработки параллельной программы из области метеорологии.

Ключевые слова: алгебра алгоритмов, автоматизация проектирования и генерации программ, графические ускорители, параллельное программирование, схема алгоритма.

ВВЕДЕНИЕ

В связи с широким распространением многоядерных процессоров в современных вычислительных системах актуальной проблемой является создание специальных средств для разработки и реинженерии параллельного программного обеспечения, которые использовались бы на всех этапах жизненного цикла программ.

В Институте программных систем НАН Украины на протяжении длительного периода развиваются теория, методология и инструментарий для автоматизированного проектирования параллельных программ, основанные на средствах алгебры алгоритмики [1]. Последняя представляет собой современное направление компьютерной науки, восходящее к фундаментальным работам академика В.М. Глушкова по теории систем алгоритмических алгебр (САА). Объектом исследования в ней являются высокоуровневые модели алгоритмов, представленные в виде схем. В [1–9] предложены формальные средства для разработки эффективных параллельных программ для многоядерных центральных процессоров, графических ускорителей и систем с распределенной памятью. Данные средства основаны на использовании САА [1, 2], онтологий [3], алгебродинамических моделей и дискретных динамических систем [4, 5], метода параметрически управляемой генерации схем алгоритмов [6], а также техники переписывания термов [7–9]. С учетом разработанной теории и методологии создан интегрированный инструментарий проектирования и синтеза программ (ИПС) [2, 6–8], а также система символьных вычислений TermWare [9].

Система ИПС базируется на представлении спецификаций алгоритмов в САА и выполняет генерацию последовательных и параллельных программ на языках программирования Java и C++. Система TermWare, основанная на парадигме переписывающих правил, дополняет возможности системы ИПС и исполь-

зуется для автоматизации выполнения формальных трансформаций алгоритмов в целях оптимизации их производительности по заданным критериям (память, быстродействие, загрузка оборудования и др.). Разработана также новая версия ИПС — онлайновый диалоговый конструктор синтаксически правильных программ (ОДСП).

В настоящей статье предложено дальнейшее развитие алгеброалгоритмической методологии и инструментария для автоматизации конструирования параллельных программ для графических ускорителей (Graphics Processing Units, GPU), позволяющих значительно повысить производительность вычислений по сравнению с обычными процессорами. Рассмотрена проблема автоматизированного проектирования и генерации параллельных программ для платформы NVIDIA CUDA [10] с использованием средств САА и системы ОДСП.

Предлагаемый подход подобен описанному в работах об алгебраическом программировании [11] и синтезе программ на основе спецификаций [12, 13], а также о проблеме генерации кода для графических процессоров. Существующие подходы к синтезу программ для GPU основаны, в частности, на аннотациях для описания свойств структур данных и областей кода [14], сетях процессов Кана [15], директивах компилятора [16], графах потока данных [17], высокоуровневых абстракциях структур данных, задач и коммуникационных операторов [18].

Отличие приведенного далее подхода состоит в использовании спецификаций алгебры Глушкова, представленных в естественно-лингвистической форме, облегчающей понимание алгоритмов и достижение необходимого качества программ. Преимуществом разработанных средств является применение метода диалогового конструирования синтаксически правильных программ, исключающего возможность возникновения синтаксических ошибок в процессе проектирования схем.

Применение подхода проиллюстрировано на примере проектирования параллельной программы из области метеорологии.

ФОРМАЛИЗОВАННОЕ ПРОЕКТИРОВАНИЕ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ АЛГЕБРЫ АЛГОРИТМОВ И ПЕРЕПИСЫВАЮЩИХ ПРАВИЛ

В основу предлагаемого подхода к проектированию параллельных программ положен аппарат САА и их модификаций [1]. Модифицированные САА (САА-М) предназначены для формализации процессов мультиобработки, которые возникают при конструировании программного обеспечения в мультипроцессорных системах. Описанные САА-М являются двухосновной алгеброй $< U, B; \Omega >$, где U — множество операторов, B — множество логических условий (предикатов), Ω — сигнатура операций. Операторы представляют собой отображения информационного множества (ИМ) в себя, логические условия являются отображениями ИМ во множество значений трехзначной логики $E_3 = \{0, 1, \eta\}$, где 0 — ложь, 1 — истина, η — неопределенность. Сигнатура $\Omega = \Omega_1 \cup \Omega_2$ состоит из системы Ω_1 логических операций, принимающих значения во множестве B, и системы Ω_2 операций, принимающих значения во множестве операторов U. Операторы и предикаты могут быть базисными или составными. Операции, входящие в сигнатуру Ω, рассмотрены далее.

На САА-М базируется алгоритмический язык САА/1 [1], предназначенный для многоуровневого структурного проектирования и документирования последовательных и параллельных алгоритмов и программ. Преимуществом его использования является возможность описания алгоритмов в естественно-лингвистической форме, удобной для пользователя, что облегчает достижение требуемого качества программ. Представления операторов в языке САА/1 называются САА-схемами.

Составные предикаты языка САА/1 строятся из базисных с помощью логических операций САА-М:

— дизъюнкция *predicate*1 OR *predicate*2;

— конъюнкция *predicate*1 AND *predicate*2;

— отрицание NOT *predicate*.

Составные операторы строятся из элементарных на основе использования следующих основных операций:

— последовательное выполнение операторов (композиция) *operator*1; *operator*2;

— оператор ветвления IF (*predicate*) THEN *operator*1 ELSE *operator*2 END IF;

— оператор цикла WHILE (*predicate*) *operator* END OF LOOP.

В работах [2, 7, 8] приведены спецификации дополнительных операций САА-М, предназначенных для формализации многопоточных вычислений для многоядерных центральных процессоров.

Далее в сигнатуру Ω включены новые операции, ориентированные на проектирование параллельных программ для платформы NVIDIA CUDA [10]. Отметим, что технология CUDA представляет собой программно-аппаратный комплекс, позволяющий разрабатывать программы для графических ускорителей GPU. Последний является вычислительным устройством, сопроцессором для центрального процессора, имеющим собственную память и обрабатывающим параллельно определенное количество потоков (threads). Ядром (kernel) называется функция для GPU, выполняемая потоками. Модель программирования в CUDA предполагает группирование потоков в блоки. Каждый блок представляет собой массив потоков, взаимодействующих с помощью разделяемой памяти и точек синхронизации. Блоки, в свою очередь, объединяются в сетку блоков.

Новыми конструкциями САА-М для проектирования программ для графических ускорителей являются следующие:

— определение функции-ядра

$$\text{KERNEL } fname(param\_list) = function\_body,$$

где *fname* — название функции, *param_list* — список формальных параметров, *function_body* — реализация функции, представленная в САА;

— операция вызова функции-ядра

$$fname(N_b, N_{th}, arg\_list),$$

где N_b — количество блоков в сетке, N_{th} — количество потоков в каждом блоке, *arg_list* — список фактических параметров функции;

— операторы, получающие значения глобального (уникального) и в границах блока локального индексов потока, а также присваивающие полученное значение целочисленной переменной *i*:

$$\text{Get global index of the thread}(i),$$

$$\text{Get local index of the thread}(i);$$

— синхронизатор — операция ожидания завершения вычислений всеми потоками

$$\text{Synchronizer (all threads completed work)};$$

— операции выделения и освобождения памяти графического процессора для некоторой переменной

$$\text{Allocate the memory for variable on GPU } (var\_name, size),$$

$$\text{Free the memory for variable on GPU } (var\_name),$$

где var_*name* — название переменной, *size* — объем необходимой памяти (в байтах);

— операция копирования данных из памяти центрального процессора в память графического процессора и в обратном направлении

$$\text{Copy data from CPU to GPU } (dest, src, count),$$

$$\text{Copy data from GPU to CPU } (dest, src, count),$$

где *dest* — переменная копирования, *src* — переменная, значение которой необходимо скопировать, *count* — объем копируемых данных (в байтах).

Пример 1. Проиллюстрируем использование некоторых из описанных выше операций САА-М для построения простого алгоритма, предназначенного для выполнения на GPU. Далее приведена САА-схема алгоритма параллельного заполнения значениями элементов числового массива.

SCHEME Parallel algorithm for GPU, assigning values to array elements;

KERNEL Set array values $(A, N, val) =$
  Get global index of the thread (i);
  Assign value $(A[i], i + val)$;

main function =

  Declare integer array (h_A, N);
  Declare integer array (d_A);
  Allocate the memory for variable on GPU (d_A, A_size);
  Set array values $(N_b, N_{th}, d\_A, N,\ 10)$;
  Copy data from GPU to CPU (h_A, d_A, A_size);
  Free the memory for variable on GPU (d_A);
  Print array (h_A, N);

Схема включает реализацию функции-ядра и основной функции схемы. Функция-ядро Set array values (A, N, val) выполняет присваивание значения $i + val$ элементу $A[i]$ одномерного целочисленного массива A длины N, где i — индекс текущего потока, val — параметр функции. В реализации основной функции (main) приведены операторы определения подлежащего обработке массива h_A, который хранится в памяти центрального процессора, а также операторы определения и резервирования памяти для соответствующего массива d_A в памяти GPU. Далее выполняется вызов описанной ранее функции-ядра, в результате которого осуществляется параллельное заполнение значениями элементов массива d_A. Значения параметров N_b и N_{th} (количество блоков в сетке и потоков в каждом блоке) выбираются таким образом, что $N_b * N_{th} = N$. После этого данные из массива d_A копируются в массив h_A и выполняется оператор освобождения зарезервированной памяти для массива d_A. Последний оператор в функции main выводит на экран значение результирующего массива h_A.

В процессе проектирования схем алгоритмов для выполнения их автоматизированных преобразований совместно с САА-М применяются переписывающие правила с использованием разработанной системы символьных вычислений TermWare [9]. Трансформация основана на применении к алгоритму, представленному в виде терма, систем правил $f(x_1, ..., x_n) \to g(x_1, ..., x_n)$, где f и g — термы, x_i — переменные термов.

Пример 2. Применим технику переписывающих правил для преобразования функции-ядра Set array values из примера 1. Вначале представим данную функцию в виде терма

```
set_array_values(params(A, N, val),
    get_global_thread_index(i),
    assign_value(arr_elem(A, i), summ(i, val))
).
```

Пусть оператор assign_value, приведенный в реализации данной функции, необходимо включить в блок оператора ветвления IF таким образом, чтобы он выполнялся только при условии $i < N$. Для этого к описанному ранее терму применим правило

$$\text{set\_array\_values}(\$x1, \$x2,\ \text{assign\_value}(\$x3, \$x4)) \rightarrow$$
$$\text{set\_array\_values}(\$x1, \$x2, \text{IF}(\text{LESS}(i, N), \text{assign\_value}(\$x3, \$x4))),$$

где $\$x1$, $\$x2$, $\$x3$, $\$x4$ — переменные. В результате, исходный терм преобразуется в следующий:

$$\begin{aligned}&\text{set\_array\_values}(\text{params}(A, N, val),\\ &\quad \text{get\_global\_thread\_index}(i),\\ &\quad \text{IF}\ (\text{LESS}(i, N), \text{assign\_value}(\text{arr\_elem}(A, i), \text{summ}(i, val)))\\ &).\end{aligned}$$

Использование операций САА-М для проектирования параллельного алгоритма для решения задачи из области метеорологии рассмотрено далее.

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА АВТОМАТИЗИРОВАННОЙ РАЗРАБОТКИ ПРОГРАММ

Разработанные инструментальные средства автоматизированного проектирования и генерации программ основаны на использовании САА и методе диалогового конструирования синтаксически правильных программ (ДСП-методе) [1]. В отличие от традиционных синтаксических анализаторов ДСП-метод ориентирован не на поиск и исправление синтаксических ошибок, а на исключение возможности их появления в процессе построения алгоритма. Суть метода заключается в поуровневом конструировании схем сверху вниз путем суперпозиции языковых конструкций САА-М. На каждом шаге конструирования система в диалоге с пользователем предоставляет на выбор лишь те конструкции, подстановка которых в формируемый текст не нарушает синтаксической правильности схемы. На основе построенной схемы алгоритма выполняется автоматическая генерация текста программы. Отображение операций САА-М в текст на языке программирования представлено в виде шаблонов и хранится в базе данных.

Описанный подход реализован в системе ИПС [2, 6–8]. Для автоматизации выполнения трансформаций алгоритмов система ИПС применяется совместно с системой TermWare [9], основанной на парадигме переписывающих правил.

Рассмотрим новую версию системы ИПС — систему ОДСП, предназначенную для диалогового проектирования, генерации и запуска программ. Особенностью инструментария ОДСП по сравнению с ИПС является многопользовательское использование системы через Интернет и распределенная архитектура системы. Отметим, что компоненты инструментария ОДСП автономны, гибко связаны и имеют согласованный протокол обмена данными, т.е. описанная модель по сути является сервисно-ориентированной.

Система ОДСП состоит из следующих компонентов (рис. 1). Клиент — веб-интерфейс для диалогового взаимодействия пользователя с системой и ее ресурсами. Он предоставляет возможность конструировать схемы алгоритмов, выбирая необходимые элементы базы данных, осуществлять генерацию кода на целевом языке программирования (Java или C++) и выполнять запуск программ в вычислительной среде. В основу проектирования алгоритмов в системе положен диалоговый режим с использованием списка конструкций и дерева алгоритма. Спецификации языковых конструкций основаны на

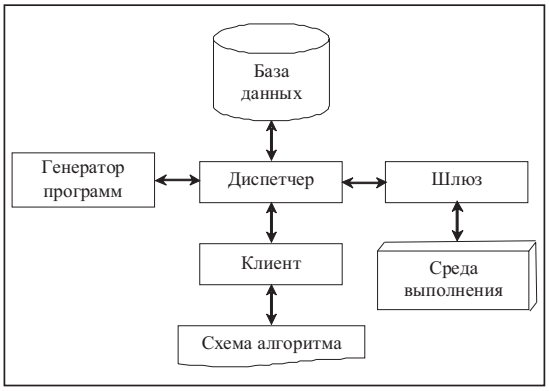

Рис. 1. Архитектура системы ОДСП

англоязычной версии языка CAA/1. Построение схем осуществляется сверху вниз путем детализации алгоритмических конструкций. В процессе конструирования схемы пользователь поочередно выбирает необходимые конструкции из списка и добавляет их в дерево.

Диспетчер является ядром инструментария и организует связь между клиентом, генератором программ, шлюзом и базой данных. Генератор программ выполняет генерацию текста программы на основе схемы алгоритма, построенной с помощью интерфейса (клиента). Шлюз обеспечивает запуск, анализ и получение результатов работы программ в среде выполнения. Последняя представляет собой программную платформу, установленную на стороне сервера системы ОДСП, включает операционную систему и программное обеспечение, необходимое для компиляции и запуска программ.

В базе данных системы ОДСП хранятся разработанные проекты алгоритмов, описание языковых конструкций CAA-M, каркасов проектов на языке программирования, информация о настройках запуска программ. Описание каждой языковой конструкции CAA-M в базе данных включает ее представление в естественно-лингвистической форме, информацию о типе конструкции, реализацию (шаблон) на выбранном языке программирования, названия переменных и список формальных параметров.

Ранее система ОДСП в основном использовалась для разработки параллельных программ сортировки для многоядерных центральных процессоров. В данной работе база данных системы дополнена операциями, ориентированными на проектирование программ для графических ускорителей.

ПРИМЕР ПРИМЕНЕНИЯ АЛГЕБРОАЛГОРИТМИЧЕСКОГО ИНСТРУМЕНТАРИЯ ДЛЯ ПРОЕКТИРОВАНИЯ ПРИКЛАДНОЙ ПАРАЛЛЕЛЬНОЙ ПРОГРАММЫ

Рассмотрим использование аппарата CAA-M и системы ОДСП на примере разработки CUDA-программы для решения двумерной задачи конвективной диффузии, которая возникает при математическом моделировании циркуляции атмосферы в метеорологии. Данная тестовая задача состоит в решении совокупности уравнений конвективной диффузии:

$$\frac{\partial u}{\partial t} + v_1 \frac{\partial u}{\partial x_1} + v_2 \frac{\partial u}{\partial x_2} = \frac{\partial}{\partial x_1}\left(\mu_1 \frac{\partial u}{\partial x_1} \right) + \frac{\partial}{\partial x_2}\left(\mu_2 \frac{\partial u}{\partial x_2} \right) + f \qquad (1)$$

при $(x_1, x_2) \in \Theta,\ t \in [0;10]$,

$$u(0, x_1, x_2) = \sin(x_1 + x_2) \qquad (2)$$

при $(x_1, x_2) \in \Theta,\quad t = 0$,

$$u(t, x_1, x_2) = u_A(t, x_1, x_2) \qquad (3)$$

при $(x_1, x_2) \in \partial\Theta,\ t \in [0;10]$,

где

$$\Theta = [0,1] \times [0,1];\ v_k = \sin(x_k);\ \mu_k = 0.001 + 0.1 * \sin^2(x_k) > 0;$$

$$f(t, x_1, x_2) = (v_1 + v_2 - (1 + 0.1 * (\sin(2x_1) + \sin(2x_2)))) *$$

$$* \cos(x_1 + x_2 - t) + (\mu_1 + \mu_2) * \sin(x_1 + x_2 - t).$$

Здесь $u = u(t, x_1, x_2)$ — зависимая функция (например, скорость ветра); t — время; x_1, x_2 — координаты; Θ — двумерная пространственная область определения задачи; $f = f(t, x_1, x_2)$ — свободный член уравнения; v_k — коэффициент конвекции; μ_k – коэффициент диффузии. Далее для координат x_1 и x_2 использованы обозначения x и y соответственно.

Для решения задачи (1)–(3) в CUDA-программе применяется конечно-разностный численный метод, изложенный в [19]. В процессе решения область Θ разбивается на S_x и S_y равномерных шагов (узлов) по осям x и y соответственно. Таким образом, общее количество узлов составляет $S_x \times S_y$. Распараллеливание вычислений для данной задачи заключается в разбиении области размером $S_x \times S_y$ на подмножества и параллельном решении совокупности подзадач, определенных на этих подмножествах.

Далее приведена общая CAA-схема разработанного параллельного GPU-алгоритма для задачи конвективной диффузии. Схема построена в системе ОДСП в соответствии с рассмотренным ранее методом диалогового конструирования. Кроме описанных переменных S_x и S_y алгоритм содержит такие основные переменные: N_b — количество блоков в сетке CUDA; N_{th} — количество потоков в каждом блоке; $Time = 0$ — начальное значение времени; $T = 10$ — конечное значение времени; $dT = 0.001$ — временной шаг; t — время; m — количество вычислительных шагов; h_pX, d_pX, h_pY, d_pY — массивы аргументов; u_x и u_y — массивы для хранения значений функции u.

SCHEME Parallel algorithm for convection-diffusion problem for GPU;

main function =

Get command line arguments (S_x, S_y);
Initialize the data for convection-diffusion problem;
Start the timer;
Input of initial data (N_b, N_{th});
Fill the arrays of equation coefficients $(N_b, N_{th}, Time + m * dT)$;
Fill the array for function u with initial values (N_b, N_{th});
Synchronizer (all threads completed work);
Copy data from GPU to CPU (h_pX, d_pX, d_pX_size);
Copy data from GPU to CPU (h_pY, d_pY, d_pY_size);
Synchronizer (all threads completed work);
Declare integer variable $(t, 0)$;
WHILE $(m * dT * t < T)$
Fill the arrays of equation coefficients $(N_b, N_{th}, Time + m * dT)$;
Fill the border conditions with zero values (N_b, N_{th});
Fill the border conditions (N_b, N_{th});
Fill the arrays of equation coefficients $(N_b, N_{th}, Time)$;
Fill arrays u_x and u_y with initial values (N_b, N_{th});
Synchronizer (all threads completed work);
Compute u_x (N_b, N_{th});
Synchronizer (all threads completed work);
Compute u_y (N_b, N_{th});
Synchronizer (all threads completed work);
Compute average value on the basis of u_x and u_y (N_b, N_{th});
Synchronizer (all threads completed work);
Increment $(Time, m * dT)$;
Increment $(t, 1)$;
END OF LOOP;
Stop the timer and print the execution time;
Copy data from GPU to CPU for convection-diffusion problem;
Compare computed function values with real values and compute an error;
Write the obtained results for function u to files;
Deallocate memory for variables of convection-diffusion problem;

Реализация основной функции приведенной САА-схемы начинается с оператора, который считывает значение двух параметров командной строки и присваивает соответствующие значения переменным S_x и S_y. Далее выполняется инициализация данных и начинается отсчет времени. После этого операции с аргументами N_b и N_{th} осуществляют вызовы функций-ядер. Последние вводят начальные данные, заполняют массивы коэффициентов уравнений и массив для функции *u*. Синхронизатор, приведенный после вызовов функций, выполняет задержку вычислений до тех пор, пока все потоки не завершат работы. Отметим, что алгоритм содержит цикл WHILE, в теле которого находятся вызовы дополнительных функций-ядер, заполняющих массивы коэффициентов, граничных условий, а также вычисляющих значения u_x, u_y и среднее значение. Алгоритмы функций-ядер представляются в отдельных САА-схемах.

Общим результатом выполнения алгоритма является совокупность вычисленных значений функции *u* для координат *x* и *y*, где $(x, y) \in \Theta$. В конце программы выполняется сравнение полученных значений функции *u* с реальными и вычисляется погрешность. После этого значения координат и функции *u* записываются в файлы.

С использованием системы ОДСП на основе построенной схемы алгоритма автоматически сгенерирован текст программы на языке CUDA C. Проведен эксперимент по выполнению разработанной параллельной программы и соответствующей последовательной программы на одном из узлов раздела СКИТ-4 кластера Института кибернетики НАН Украины [20]. Данный узел содержит графический ускоритель NVIDIA Tesla M2075 (448 вычислительных ядер) и центральный процессор Intel Xeon E5-2670. Численные эксперименты выполнены для значений размера задачи от $S_x \times S_y = 64 \times 64$ до $S_x \times S_y = 2048 \times 2048$. Получены значения мультипроцессорного ускорения T_s / T_p, где T_s и T_p — время выполнения последовательной и параллельной программы соответственно. Максимальное значение ускорения при $S_x \times S_y = 2048 \times 2048$ составило 69 раз.

ЗАКЛЮЧЕНИЕ

Рассмотрено развитие алгеброалгоритмической методологии и инструментария для автоматизированного проектирования и генерации программ для графических ускорителей. Преимущество описанного подхода состоит в использовании языковых конструкций, близких к естественному языку, а также в применении метода, который обеспечивает синтаксическую правильность проектируемых алгоритмов и программ. Подход реализован в инструментальной системе, предназначенной для диалогового конструирования схем алгоритмов и генерации программ на основе компонентов повторного использования, т.е. операций алгебры алгоритмов. Применение инструментария проиллюстрировано на примере разработки параллельной программы для задачи конвективной диффузии. Проведен эксперимент по выполнению разработанной программы на графическом ускорителе, в результате которого получен хороший показатель распараллеливаемости вычислений.

СПИСОК ЛИТЕРАТУРЫ

1. Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А. Алгеброалгоритмические модели и методы параллельного программирования. — Киев: Академпериодика, 2007. — 631 с.
2. Дорошенко А.Е., Жереб К.А., Яценко Е.А. Формализованное проектирование эффективных многопоточных программ // Проблемы программирования. — 2007. — № 1. — С. 17–30.

3. Doroshenko A., Yatsenko O. Using ontologies and algebra of algorithms for formalized development of parallel programs // Fundamenta Inform. — 2009. — **93**, N 1–3. — P. 111–125.

4. Doroshenko A., Zhereb K., Yatsenko O. Formal facilities for designing efficient GPU programs // Proc. Int. Workshop "Concurrency, Specification, and Programming" (CS&P'2010). (Bernau, 27–29 Sept., 2010). — Berlin: Humboldt University, 2010. — P. 142–153.

5. Андон Ф.И., Дорошенко А.Е., Жереб К.А. Программирование высокопроизводительных параллельных вычислений: формальные модели и графические ускорители // Кибернетика и системный анализ. — 2011. — № 4. — С. 176–187.

6. Yatsenko O. On parameter-driven generation of algorithm schemes // Proc. Intern. Workshop "Concurrency, Specification, and Programming" (CS&P'2012). (Berlin, 26–28 Sept., 2012). — Berlin: Humboldt University, 2012. — P. 428–438.

7. Doroshenko A., Zhereb K., Yatsenko O. Developing and optimizing parallel programs with algebra-algorithmic and term rewriting tools // Proc. Intern. Conf. "Information and Communication Technologies in Education, Research, and Industrial Applications" (ICTERI'2013). (Kherson, 19–22 June, 2013), Revised Selected Papers. — Berlin: Springer-Verlag. — 2013. — **412**. — P. 70–92.

8. Яценко Е.А. Интеграция инструментальных средств алгебры алгоритмов и переписывания термов для разработки эффективных параллельных программ // Проблемы программирования. — 2013. — № 2. — С. 62–70.

9. Doroshenko A., Shevchenko R. A Rewriting framework for rule-based programming dynamic applications // Fundamenta Inform. — 2006. — **72**, N 1–3. — P. 95–108.

10. Wilt N. The CUDA handbook. A Comprehensive guide to GPU programming. — Boston: Addison-Wesley, 2013. — 528 p.

11. Sannella D., Tarlecki A. Foundations of algebraic specification and formal software development. — Berlin: Springer-Verlag, 2012. — 594 p.

12. Flener P. Achievements and prospects of program synthesis // Lecture Notes in Artificial Intelligence. — 2002. — **2407**. — P. 310–346.

13. Gulwani S. Dimensions in program synthesis // Proc. 12th Intern. ACM SIGPLAN symposium on Principles and practice of declarative programming. (Hagenberg, 26–28 July, 2010). — New York: ACM, 2010. — P. 13–24.

14. Ueng S., Lathara M., Baghsorkhi S.S., Hwu W.W. CUDA-lite: reducing GPU programming complexity CUDA-lite: reducing GPU programming complexity // Proc. 21st Intern. Workshop on Languages and Compilers for Parallel Computing (LCPC'2008). (Edmonton, 31 July–2 Aug, 2008). — Berlin: Springer-Verlag, 2008. — P. 1–15.

15. Haid W., Schor L., Huang K., Bacivarov I., Thiele L. Efficient execution of Kahn process networks on multi-processor systems using protothreads and windowed FIFOs // Proc. Workshop on Embedded Systems for Real-Time Multimedia (ESTImedia'09). (Grenoble, 15–16 Oct., 2009). — Los Alamitos: IEEE Computer Society Press, 2009. — P. 35–44.

16. Han T.D., Abdelrahman T.S. hiCUDA: A High-level language for GPU programming // IEEE Transactions on Parallel and Distributed systems. — 2011. — **22**, N 1. — P. 78–90.

17. Jung H., Yi Y., Ha S. Automatic CUDA code synthesis framework for multicore CPU and GPU architectures // Lecture Notes in Comp. Sci. — 2012. — **7203**. — P. 579–588.

18. Dubach C., Cheng P., Rabbah R., Bacon D.F., Fink S.J. Compiling a high-level language for GPUs (via language support for architectures and compilers) // Proc. 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'12). (Beijing, 11–16 June, 2012). — New York: ACM, 20112 — P. 1–12.

19. Прусов В.А., Дорошенко А.Е., Черныш Р.И. Метод численного решения многомерной задачи конвективной диффузии // Кибернетика и системный анализ. — 2009. — № 1. — С. 100–107.

20. Суперкомпьютер Института кибернетики НАН Украины — http: // icybcluster.org.ua

Поступила 12.09.2014