УДК 004.272.26

ПРИМЕНЕНИЕ РЕСУРСОЕМКИХ АЛГОРИТМОВ В СИСТЕМАХ МАШИННОГО ЗРЕНИЯ РЕАЛЬНОГО ВРЕМЕНИ

Круглов А.В., Круглов В.Н., Чирышев А.В., Чирышев Ю.В.

ГОУ ВПО «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина», Екатеринбург, email: v.krouglov@mail.ru

В системах машинного зрения реального времени при разработке алгоритмической части часто возникает необходимость поиска компромисса между эффективностью тех или иных алгоритмов и их быстродействием. Многие методы, демонстрирующие высокую точность выполнения, не могут использоваться за счет неприемлемых времен выполнения, что приводит к снижению эффективности работы всей системы. Однако с появлением общедоступных технологий выполнения параллельных вычислений на IBM PC-совместимых компьютерах появилась возможность разработки алгоритмов для систем машинного зрения на основе более ресурсоемких методов, при этом сохраняя реальный масштаб времени функционирования. Данная работа посвящена анализу и сравнению существующих решений для выполнения параллельных вычислений на базе многоядерных CPU или GPU, выявлению достоинств и недостатков технологий применительно к разработке систем машинного зрения реального времени и, в частности, задачам обработки изображений.

Ключевые слова: параллельные вычисления, CPU, GPU, производительность, OpenMP, OpenCL, MPI, Cuda, C++ AMP

APPLICATION OF RESOURCE-INTENSIVE ALGORITHMS IN REAL-TIME MACHINE VISION SYSTEMS

Kruglov A.V., Kruglov V.N., Chiryshev A.V., Chiryshev Y.V.

Ural Federal University n.a. the first President of Russia B.N. Yeltsin, Ekaterinburg, email: v.krouglov@mail.ru

It is often necessary to find a compromise between the efficiency of various algorithms and their performance in the development of the algorithmic part of machine vision systems. Many methods showing high precision execution cannot be used due to unacceptable time costs, and this fact reduces the efficiency of the entire system. However, with the advent of parallel computing technologies on IBM PCs the development of machine vision systems algorithms based on more demanding methods while maintaining real-time operation mode becomes possible. This work is devoted to the analysis and comparison of existing solutions for parallel computing based on multi-core CPU or GPU, and to the identifying of advantages and disadvantages of these solutions in relation to the development of real-time machine vision systems and image processing in particular.

Keywords: parallel computing, CPU, GPU, performance, OpenMP, OpenCL, MPI, Cuda, C++ AMP

В настоящее время существует довольно много разных технологий параллельного программирования. Причем эти технологии отличаются не только и не столько языками программирования, сколько архитектурными подходами к построению параллельных систем.

В общем случае можно выделить 3 основных направления в подходе к выполнению параллельных вычислений:

– системы с раздельной памятью;

– многопоточные приложения на многопроцессорных системах с общей памятью;

– использование графического процессора видеокарты.

Системы на базе нескольких компьютеров относят к классу систем для распределенных вычислений. Подобные решения используются довольно давно. Наиболее яркий пример технологии распределенных вычислений – MPI (Message Passing Interface – интерфейс передачи сообщений). MPI является наиболее распространённым стандартом интерфейса обмена данными в параллельном программировании, существуют его реализации для большого числа компьютерных платформ [4].

Технологии параллельного программирования в среде с общей памятью получили свое развитие относительно недавно – толчком к этому послужило широкое распространение многоядерных процессоров. Одним из первых стандартов распараллеливания программ является OpenMP, первая версия которого появилась в 1997 году [17]. Разработкой спецификации OpenMP занимаются несколько крупных производителей вычислительной техники и программного обеспечения, в частности, AMD, Fujitsu, HP, IBM, Intel, Microsoft, nVidia и др. Однако OpenMP не остался единственным средством написания программ, связанных с параллельными вычислениями. В 2008 году компанией Apple был выпущен фреймворк для написания компьютерных программ, связанных с параллельными вычислениями – OpenCL [14]. С его помощью возможна организация вычислений как на центральном, так и на графическом процессорах.

Производители графических карт также разработали свои программно-аппаратные архитектуры, позволяющие производить вычисления с использованием графических процессоров. Компания nVidia представила

в 2007 году архитектуру CUDA, примерно в то же время компания AMD выпустила свою технологию использования графических процессоров совместно с центральным процессором. Особенностями этих решений является их привязка к аппаратуре – графическим процессорам конкретного производителя [5].

К последним разработкам в области технологий параллельных вычислений относится разработанная Microsoft технология C++ Accelerated Massive Parallelism, построенная на платформе Microsoft DirectX. C++ AMP позволяет проектировать программы таким образом, чтобы они распределяли вычисления между центральным и графическим процессорами. Из преимуществ этой технологии стоит отметить открытую спецификацию, кроссплатформенность (в дальнейшем), поддержку мультивендорности как для центральных, так и для графических процессоров.

В данной статье перед авторами не ставилась задача определить лучшее решение для выполнения параллельных вычислений. Каждая из представленных технологий имеет свои достоинства и недостатки. В статье рассматривается эффективность и удобство применения наиболее распространенных технологий параллельного программирования в задаче разработки системы машинного зрения на основе IBM-совместимого компьютера.

Основная часть

Для рассмотрения были выбраны наиболее популярные в настоящий момент решения. В качестве теста выбран ленточный алгоритм перемножения матриц. Реализация алгоритмов проведена в среде Visual Studio 2010, язык программирования – C++. Далее приводится реализация для каждого решения.

OpenMP

OpenMP (Open Multi-Processing) – это стандарт параллельного программирования в среде с общей памятью. OpenMP предоставляет программистам набор прагм, процедур и переменных среды, позволяющих легко распараллеливать код, написанный на Фортране, C или C++ [17].

В OpenMP используется модель параллельного выполнения «ветвление-слияние». Программа OpenMP начинается как единственный поток выполнения, называемый начальным потоком. Когда поток встречает параллельную конструкцию, он создает новую группу потоков, состоящую из себя и неотрицательного числа дополнительных потоков, и становится главным в новой группе. Все члены новой группы (включая главный) выполняют код внутри параллельной конструкции, в конце которой имеется неявный барьер. После параллельной конструкции выполнение пользовательского кода продолжает только главный поток.

В OpenMP поддерживаются две основные конструкции для указания того, что работу в области параллельности следует разделить между потоками группы. Эти конструкции разделения работы – циклы и разделы. Кроме этого, существует программа, которая дает всем потокам указание ожидать друг друга перед тем, как они продолжат выполнение за барьером.

Для того чтобы использовать в проекте возможности OpenMP, необходимо явно указать это в свойствах проекта (Проект → Свойства → C/C++ → Язык → Поддержка OpenMP).

Ниже приведен участок кода программы, реализующей ленточное перемножение матриц.

```
#pragma omp parallel for      // указание выполнения нижеследующего цикла
средствами OpenMP
for(int i=0;i<n;i++)
{
      for(int k=0;k<n;k++)
      {
            double sum=0;
            for(int j=0;j<n;j++)
            {sum+=MatrixA[i*n+j]*MatrixB[j*n+k];      }
            MatrixC[i*n+k]=sum;
      }
```

Как видно из приведенного листинга, использование OpenMP не влечет за собой включение сложных синтаксических конструкций, технология хорошо интегрирована со средой разработки и ее использование удобно и интуитивно понятно для разработчика.

OpenCL

OpenCL (Open Computing Language) – открытый стандарт для универсального параллельного программирования различных типов процессоров [16]. Стандарт предоставляет программистам переносимый

и эффективный доступ ко всем возможностям гетерогенных вычислительных платформ.

Для координации работы всех устройств гетерогенной системы всегда есть одно главное устройство, которое взаимодействует со всеми остальными посредством OpenCL API. Такое устройство называется «хост» и определяется вне OpenCL. Хост посылает пакеты с информацией и исполнительные команды на ускорители и получает готовые данные. Хост обрабатывает и выполняет программный код, представляющий собой тело программы, на центральном процессоре под управлением операционной системы, используя C++ или другой язык программирования. Ускорители выполняют OpenCL код, написанный на языке OpenCL C99. Существует определенный OpenCL компилятор для центрального процессора, для графического процессора и для специальных карт-ускорителей [9, 11, 15, 16].

Созданные на OpenCL программы потенциально могут использовать имеющиеся ресурсы вычислительной системы следующим образом:

– определить доступные ресурсы в гетерогенной системе и выбрать подходящие;

– создать последовательность инструкций, которые будут выполняться на ресурсах;

– подготовить начальные данные для вычислений;

– передать управление на эти ресурсы в виде OpenCL-инструкций для обработки подготовленных данных;

– собрать результаты вычислений.

Плюсы технологии несомненны: ускорение вычислений, кроссплатформенность, способность исполнять код как под GPU, так и под CPU, поддержка стандарта целым рядом компаний: Apple, AMD, Intel, nVidia и некоторыми другими. Минусов не так много, но и они есть: необходимость установки и настройки драйверов для работы OpenCL на различных устройствах.

Для использования OpenCL в проекте требуется установить соответствующий SDK в зависимости от выбора аппаратного ускорителя – CPU, GPU. Получить SDK можно на сайте производителя – Intel, nVidia, AMD и пр.

Программирование на OpenCL включает подготовку кода для запуска в ядре (распараллеливаемые операции) и на хосте (программа, управляющая ядрами). Подробное описание использования данной технологии можно найти в официальной документации [14]. В качестве примера приведен фрагмент кода, выполняющий инициализацию OpenCL.

```
int Setup_OpenCL(const char *program_source)    //инициализация объектов
{
cl_device_id devices[16];
size_t cb;
cl_uint size_ret = 0;
cl_int err;
int num_cores;
cl_device_id device_ID;
char device_name[128] = {0};
      cl_platform_id intel_platform_id = GetIntelOCLPlatform();
      //выбор платформы Intel
if( intel_platform_id == NULL )
    {
printf("ERROR: Failed to find Intel OpenCL platform.\n");
return -1;
    }
cl_context_properties context_properties[3] = {CL_CONTEXT_PLATFORM,
(cl_context_properties)intel_platform_id, NULL };
      // выбор контекста устройства
      if(g_bRunOnPG)
      {
            g_context = clCreateContextFromType(context_properties,
CL_DEVICE_TYPE_GPU, NULL, NULL, NULL);
      }
      else
      {
            g_context = clCreateContextFromType(context_properties,
CL_DEVICE_TYPE_CPU, NULL, NULL, NULL);
      }
if (g_context == (cl_context)0)
return -1;
      // получение списка устройств CPU для данного контекста
```

```
err = clGetContextInfo(g_context, CL_CONTEXT_DEVICES, 0, NULL, &cb);
clGetContextInfo(g_context, CL_CONTEXT_DEVICES, cb, devices, NULL);
g_cmd_queue = clCreateCommandQueue(g_context, devices[0], 0, NULL);
if (g_cmd_queue == (cl_command_queue)0)
    {
Cleanup_OpenCL();
return -1;
    }
char *sources = ReadSources(program_source);    //чтание ядра OpenCL
g_program = clCreateProgramWithSource(g_context, 1, (const char**)&sources,
NULL, NULL);
if (g_program == (cl_program)0)
    {
printf("ERROR: Failed to create Program with source...\n");
Cleanup_OpenCL();
free(sources);
return -1;
    }
err = clBuildProgram(g_program, 0, NULL, NULL, NULL, NULL);
g_kernel = clCreateKernel(g_program, "MatrixMul", NULL);
free(sources);
      // выбор первого устройства
device_ID = devices[0];
return 0;
}
```

В целом код на OpenCL получается довольно громоздким за счет необходимости инициализации вычислительного устройства и компиляции ядер. Однако данный недостаток компенсируется кроссплатформенностью и отсутствием привязки к производителю.

MPI

Message Passing Interface (MPI) – программный интерфейс для передачи информации, который позволяет процессам, выполняющим одну задачу, обмениваться сообщениями. Разработан Уильямом Гроуппом и Эвином Ласком [4].

MPI является наиболее распространенным стандартом интерфейса обмена данными при организации параллельных вычислений, существуют его реализации для большого числа компьютерных платформ. Этот стандарт также используется при разработке программ для кластеров и суперкомпьютеров. В стандарте MPI описан интерфейс передачи сообщений, который может поддерживаться как платформой, так и приложением пользователя. В настоящее время существует большое количество бесплатных и коммерческих реализаций MPI. Существуют реализации для языков Фортран 77/90, C и C++ [8].

MPI предоставляет программисту единый механизм взаимодействия ветвей внутри параллельного приложения независимо от машинной архитектуры (однопроцессорные/многопроцессорные с общей/раздельной памятью), взаимного расположения ветвей (на одном процессоре или на разных). MPI предназначен для систем с раздельной памятью, его использование для организации параллельного процесса в системе с общей памятью оказывается слишком избыточно и сложно.

Существует бесплатная реализация данной библиотеки MPICH2, которая будет использоваться в данной статье. Для тестирования производительности использовался тестовый пример перемножения матриц, входящий в состав библиотеки.

Cuda

Технология CUDA – это программно-аппаратная вычислительная архитектура nVidia, основанная на расширении языка C, которая даёт возможность организации доступа к набору инструкций графического ускорителя и управления его памятью при организации параллельных вычислений [1,3,7,13]. CUDA помогает реализовывать алгоритмы, выполнимые на графических процессорах видеоускорителей GeForce восьмого поколения и старше (серии GeForce 8, GeForce 9, GeForce 200), а также Quadro, Tesla и ION.

Вычислительная архитектура CUDA основана на понятии мультипроцессора и концепции SIMD – одна команда на множество данных (Single Instruction Multiple Data). Концепция SIMD подразумевает, что одна инструкция позволяет одновременно обработать множество данных в отдельных потоках. Совокупность всех этих потоков, запущенных в рамках одной задачи, носит

название grid. Параллельность исполнения grid`a обеспечивается, в первую очередь, наличием на видеокарте большего количества идентичных скалярных процессоров, которыми, собственно, и выполняются потоки. Физически скалярные процессоры являются частями потоковых мультипроцессоров.

Преимущества CUDA перед традиционным подходом к GPGPU вычислениям:

– интерфейс программирования основан на стандартном языке программирования C;

– доступ к разделяемой между потоками памяти размером в 16 Кб на мультипроцессор;

– линейная адресация памяти, возможность записи по произвольным адресам;

– аппаратная поддержка целочисленных и битовых операций.

Основные ограничения CUDA:

– минимальная ширина блока в 32 потока;

– закрытая архитектура CUDA, принадлежащая nVidia.

Для работы с CUDA требуется видеокарта nVidia с поддержкой данной технологии, драйвер с поддержкой CUDA, непосредственно пакет CUDA SDK и дополнительные библиотеки (CUDA toolkit). Листинг программы перемножения матриц на CUDA приведен ниже.

```
 define BLOCK_SIZE 16
#define N 1024

__global__ void mulKernel(double *a, double *b, double *c) //CUDA-ядро
{
      int bx = blockIdx.x;
      int by = blockIdx.y;
      int tx = threadIdx.x;
      int ty = threadIdx.y;
      double sum = 0.0;
      int ia = N*BLOCK_SIZE*by + N*ty;
      int ib = BLOCK_SIZE*bx + tx;
      for (int k = 0; k < N ; k++)
      {
            double elementA = a[ia+k];
            double elementB = b[ib+k*N];
            sum += elementA*elementB;
      }
      intic = N*BLOCK_SIZE*by + BLOCK_SIZE*bx;
      c[ic+N*ty+tx] = sum;
}
int main()
{
      cudaSetDevice(0); //выбор устройства
      dim3 threads (BLOCK_SIZE,BLOCK_SIZE);     // количество потоков
      dim3 blocks (N/BLOCK_SIZE,N/BLOCK_SIZE);  //количество блоков
      double *dev_a = NULL;
      double *dev_b = NULL;
      double *dev_c = NULL;
      cudaMalloc((void**)&dev_a, N*N*sizeof(double));//выделение памяти на GPU
      cudaMalloc((void**)&dev_b, N*N*sizeof(double));
      cudaMalloc((void**)&dev_c, N*N*sizeof(double));
      cudaMemcpy(dev_a, a, N*N*sizeof(double), cudaMemcpyHostToDevice);
      //копирование данных с hosta на устройство
      cudaMemcpy(dev_b, b, N*N*sizeof(double), cudaMemcpyHostToDevice);
      cudaMemcpy(dev_c, c, N*N*sizeof(double), cudaMemcpyHostToDevice);
      mulKernel<<<blocks, threads>>>(dev_a, dev_b, dev_c);//запуск ядра

      cudaDeviceSynchronize();      //синхронизацияс host'ом
      cudaMemcpy(c, dev_c, N*N*sizeof(double), cudaMemcpyDeviceToHost);
      //выгрузка данных на CPU
      cudaFree(dev_c);  //освобождение памяти на GPU
      cudaFree(dev_a);
      cudaFree(dev_b);
      delete []a;
      delete []b;
      delete []c;
      return 0;
}
```

Реализация программы аналогична реализации на OpenCL: выбор и инициализация устройства, копирование данных из CPU в GPU, запуск ядра, копирование результатов в оперативную память.

C++ Accelerated Massive Parallelism

Microsoft представила свое видение разработки приложений для работы в гетерогенной среде с использованием всех возможностей многоядерных процессоров и GPU. Это новая платформа, разработанная при активной поддержке компаний AMD и nVidia, которая является частью одиннадцатой версии Visual Studio [10,18].

Платформа получила название C++ Accelerated Massive Parallelism или C++ AMP. Цели данной платформы следующие: дать возможность использовать преимущества параллельных вычислений и GPU одновременно на языке C++, так же как сегодня эти преимущества используются на языке C (или его подобии) в разных платформах разработки: OpenCL, CUDA, DirectCompute.

Представители Microsoft особенно подчеркивают перспективы данной платформы для разработки облачных вычислений в будущем. Платформа создавалась с расчетом того, что в скором времени сотни миллионов вычислительных модулей GPU будут использоваться в приложениях через облачные сервисы. C++ AMP позволит написать такой код уже сегодня, чтобы завтра он работал на подобных мощностях [6].

С концептуальной точки зрения AMP несколько напоминает OpenCL, так как последний тоже создавался для гетерогенного параллельного программирования и также не привязан к оборудованию какого-то одного производителя. Однако OpenCL – это существенно более низкоуровневая система и она не работает на основе вызова регулярных функций, а вручную компилируется в среде выполнения. Кроме того, буферы данных OpenCL должны в явном виде поддерживаться видеокартой [12].

Реализация программы на C++ AMP выполнена в Visual Studio 2012 (описываемая технология не поддерживается в более ранних версиях). Выдержки из кода программы, реализующей ленточное перемножение матриц, приведены далее.

```
 include <amp.h>
#include <iostream>
#include <assert.h>

template<typename _type>
void mxm_amp_simple(int M, int N, int W, const std::vector<_type>& va,
const std::vector<_type>& vb, std::vector<_type>& vresult)
{
array_view <const _type, 2> av_a(e_a, va); // копирование данных на видеокарту
array_view <const _type, 2> av_b(e_b, vb);
array_view <_type, 2> av_c(e_c, vresult);
av_c.discard_data();    //запрет копирования для av_c, выходной массив
parallel_for_each(av_c.extent, [=](index<2> idx) restrict(amp) // указание
выполнения параллельных вычислений
{
            _type result = 0;
for(int i = 0; i< N; ++i)
            {
index<2> idx_a(idx[0], i);
index<2> idx_b(i, idx[1]);
result += av_a[idx_a] * av_b[idx_b];
            }
av_c[idx] = result;
        });
av_c.synchronize();
}
int main()
{
accelerator default_device;   // выбор устройства выполнения
std::wcout<<L"Usingdevice : "<<default_device.get_description() <<std::endl;
if (default_device == accelerator(accelerator::direct3d_ref))
std::cout<<"WARNING!! Running on very slow emulator! Only use this accelerator
for debugging."<<std::endl;
      ...
initialize_array(v_a, M * N);
initialize_array(v_b, N * W);
mxm_amp_simple(M, N, W, v_a, v_b, v_c_simple);
return 0;
}
```

Функция `mxm_amp_simple()` реализует параллельные вычисления, в `main()` выполняется выбор ускорителя – устройства выполнения параллельных вычислений.

Сравнение производительности

Результаты сравнения описанных технологий приведены в таблице. Тестирование проводилось на IBM-PC со следующей комплектацией:

– CPU – Intel Core i7 2.8 GHz;

– GPU – NVIDIA GeForce GTS 450.

В тесте проводилось перемножение матриц ленточным способом размером 1024 элемента. Вычисления производились для чисел с двойной точностью. Количество операций с плавающей точкой для данной задачи составляет Nflop = 2*(1024^3) = 2.14 DPGFLOP.

Сравнение производительности

Технология	Без применения параллельных вычислений	OpenMP	OpenCL (CPU)	OpenCL (GPU)	MPI	CUDA	C++ AMP
Производительность, GFLOPS	0.25	0.92	3.84	18.45	7.53	16,7	10,92

Как можно видеть по результатам тестов, наилучшие результаты получены при использовании технологий OpenCL и CUDA. Для данной задачи OpenCL показал даже лучшее время выполнения, хотя авторы работы предполагают, что для задач, связанных с моделированием и обработкой изображений, технология CUDA покажет более высокую эффективность.

Также нельзя не отметить результаты, полученные с помощью технологии C++ AMP. С учетом того, что данная спецификация только недавно появилась, а также учитывая гораздо более простую по сравнению с OpenCL синтаксическую структуру, при аналогичных возможностях в плане мультивендорности данная технология имеет все шансы для того, чтобы де-факто стать стандартом в области параллельного программирования.

На данный момент в зависимости от целей разработки того или иного решения, а также области его применения, рекомендуется использование определенной технологии выполнения параллельных вычислений:

– при жесткой связи программной и аппаратной части решения использование технологии CUDA;

– OpenCL или C++ AMP в случае, когда заранее неизвестна аппаратная конфигурация машины, на которой будет запускаться приложение.

Технологии OpenMP и MPI, конечно, уступают по производительности, так как используют параллельные вычисления на CPU. Тем не менее вариант использования данных решений совместно с вычислениями на GPU могут дать впечатляющие результаты при условии правильной реализации структуры программы, особенно в части передачи данных между устройствами.

Работа выполнена в рамках договора № 02.G25.31.0055 (проект 2012-218-03-167).

Список литературы

1. Введение в технологию CUDA [Электронный ресурс] URL: http://www.uraldev.ru/articles/id/33 (дата обращения: 11.09.2013).

2. Как и зачем мерить FLOPSы [Электронный ресурс] URL: http://habrahabr.ru/company/intel/blog/144388/ (дата обращения: 25.09.13).

3. Курс лекций по CUDA [Электронный ресурс] URL: http://www.nvidia.ru/object/cuda_state_university_courses_new_ru.html (дата обращения: 11.09.2013).

4. Основы MPI [Электронный ресурс] URL: http://habrahabr.ru/post/121925/ (дата обращения: 14.09.2013).

5. Программирование с использованием технологий AMD Stream и Nvidia CUDA [Электронный ресурс] URL: http://www.nestor.minsk.by/kg/2010/07/kg00703.html (дата обращения: 11.09.2013).

6. Спецификация C++ AMP / Microsoft corporation, 2012. – 145 с.

7. Технология NVIDIA CUDA [Электронный ресурс] URL: http://www.xakep.ru/post/47507/ (дата обращения: 11.09.2013).

8. Установка и начала использования библиотеки MPI [Электронный ресурс] URL: http://habrahabr.ru/post/47795/ (дата обращения: 14.09.2013).

9. Язык параллельных вычислений OpenCL [Электронный ресурс] URL: http://madcash.ru/raspredelennye-vychisleniya/opencl/ (дата обращения: 15.09.2013).

10. Learn C++ AMP [Электронный ресурс] URL: http://blogs.msdn.com/b/nativeconcurrency/ archive/2012/08/30/learn-c-amp.aspx (дата обращения: 17.09.2013).

11. Matthew Scarpino. OpenCL in Action / eBook, 2011. – 456 с.

12. Microsoft приносит GPU-вычисления на C++ с помощью C++ AMP [Электронный ресурс] URL: http://www.3dnews.ru/news/612811 (дата обращения: 17.09.2013).

13. NVIDIA CUDA- неграфические вычисления на графических процессорах [Электронный ресурс] URL: http://www.ixbt.com/video3/cuda-1.shtml (дата обращения: 11.09.2013).

14. OpenCL: официальный сайт [Электронный ресурс] URL: http://opencl.ru/ (дата обращения: 15.09.2013).

15. OpenCL: Мост в параллельные миры [Электронный ресурс] URL: http://www.mql5.com/ru/articles/405 (дата обращения: 15.09.2013).

16. The open standard for parallel programming of heterogeneous systems [Электронный ресурс] URL: http://www.khronos.org/opencl/ (дата обращения: 15.09.2013).

17. The OpenMP API specification for parallel programming [Электронный ресурс] URL: http://openmp.org/wp/about-openmp/ (дата обращения: 05.09.2013).

18. VS 11 Developer Preview gotchas with C++ AMP [Электронный ресурс] URL:http://blogs.msdn.com/b/nativeconcurrency/archive/2011/09/19/vs-11-developer-preview-gotchas-with-c-amp.aspx (дата обращения: 17.09.2013).

References

1. *Vvedenie v technologiyu CUDA* (Introduction to CUDA) Available at: http://www.uraldev.ru/articles/id/33 (accessed: 11 September 2013).

2. *Kak i zachem merit FLOPS* (Why and How to measure FLOPS) Available at: http://habrahabr.ru/company/intel/blog/144388/ (accessed: 25 September 2013).

3. *Kurs lektsiy CUDA* (Course on CUDA) Available at: http://www.nvidia.ru/object/cuda_state_ university_courses_new_ru.html (accessed:11 September 2013).

4. *Osnovy MPI* (MPI basics) Available at: http://habrahabr.ru/post/121925/ (accessed: 14 September, 2013).

5. *Programmirovanie s ispolzovaniem technologiy AMD Stream i Nvidia CUDA* (Programming with AMD Stream and Nvidia CUDA) Available at: http://www.nestor.minsk.by/kg/2010/07/kg00703.html (accessed: 11 September 2013).

6. *C++ AMP specifikacija* (C++ AMP specification). Microsoft corporation, 2012. 145 p.

7. *Technologija NVIDIA CUDA* (NVIDIA CUDA technology) Available at: http://www.xakep.ru/post/47507/ (accessed: 11 September 2013).

8. *Ustanovka i nachala ispolzovanija biblioteki MPI* (Installation and using MPI) Available at: http://habrahabr.ru/post/47795/ (accessed: 14 September 2013).

9. *Yazyk parallelnych vychisleniy OpenCL* (OpenCL, the language for parallel computing) Available at: http://madcash.ru/raspredelennye-vychisleniya/opencl/ (accessed: 15 September 2013).

10. Learn C++ AMP, Available at: http://blogs.msdn.com/b/nativeconcurrency/archive/2012/08/30/ learn-c-amp.aspx (accessed: 17 September 2013).

11. Matthew Scarpino. OpenCL in Action. eBook, 2011. 456 p.

12. *Microsoft prinosit GPU-vychislenija na C++ s pomoschyu C++ AMP* (Microsoft brings GPU-computing to C++ with C++ AMP) Available at: http://www.3dnews.ru/news/612811 (accessed: 17 September 2013).

13. *NVIDIA CUDA–negraficheskie vychislenija na graficheskich processorach* (NVIDIA CUDA – nongraphic computing on GPU) Available at: http://www.ixbt.com/video3/cuda-1.shtml (accessed: 11 September 2013).

14. *OpenCL: officialniy sayt* (OpenCL: official page) Available at: http://opencl.ru/ (accessed: 15 September 2013).

15. *OpenCL: Most v parallelnie miry* (OpenCL: The Bridge to Parallel Worlds) Available at: http://www.mql5.com/en/articles/405 (accessed: 15 September 2013).

16. The open standard for parallel programming of heterogeneous systems, Available at: http://www.khronos.org/opencl/ (accessed: 5 September 2013).

17. The OpenMP API specification for parallel programming, Available at: http://openmp.org/wp/about-openmp/ (accessed: 5 September 2013).

18. VS 11 Developer Preview gotchas with C++ AMP, Available at: http://blogs.msdn.com/b/ nativeconcurrency/archive/2011/09/19/vs-11-developer-preview-gotchas-with-c-amp.aspx (accessed: 17 September 2013).

Рецензенты:

Доросинский Л.Г., д.т.н., профессор, заведующий кафедрой «Информационные технологии», УрФУ, г. Екатеринбург;

Поршнев С.В., д.т.н., профессор, заведующий кафедрой «Радиоэлектроника информационных систем», УрФУ, г. Екатеринбург.

Работа поступила в редакцию 22.11.2013.