УДК 681.513.54

СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ CUDA, OPENCL И C++ AMP НА ПРИМЕРЕ ЗАДАЧИ ПРОГНОЗИРОВАНИЯ ЭНЕРГОПОТРЕБЛЕНИЯ ПРИ ПОМОЩИ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

А.В. Маликов, Р.В. Таранов

В данной статье описано сравнение трёх архитектур для параллельных вычислений: ComputeUnifiedDeviceArchitecture (CUDA), OpenComputingLanguage (OpenCL) и Accelerated-MassiveParallelism (C++ AMP). Сравнение проводится на примере задачи прогнозирования электропотребления с помощью искусственных нейронных сетей.Выявлены слабые стороны и преимущества для каждой из архитектур. Полученные результаты наглядно демонстрируют, какую из архитектур можно считать наиболее эффективной.

This article describes a comparison of the three architectures for parallel computing: Compute Unified Device Architecture (CUDA), Open Computing Language (OpenCL) and Accelerated Massive Parallelism (C ++ AMP). A comparison is carried out on the example of a power consumption problem of the forecasting using artificial neural networks. Weaknesses and advantages for each of the architectures were identified. The results demonstrate what architectures can be considered the most effective.

Ключевые слова: прогнозирование электроэнергии, нейронная сеть, параллельный алгоритм, CUDA, OpenCL, C++ AMP.

Keywords: forecasting, neural network, parallel algorithm, CUDA technology, OpenCL, C++ AMP.

1. Введение

Большая потребность в увеличении вычислительной производительности в области науки и техники привела к использованию гетерогенных вычислений на графических процессорах, выступающих в качестве сопроцессоров для параллельной обработки арифметических операций с данными [1].

Параллельные вычисления внесли большой вклад в развитие компьютерных технологий и различных областей наук, таких как: математическое моделирование, интеллектуальный анализ и обработка данных и т.д. Параллельные вычисления являются эффективной формой обработки информации [2].

В процессе развития технологий были разработаны новые компьютерные архитектуры, получены решения с несколькими вычислительными процессорами на одной плате, а также наблюдается тенденция развития процессоров с несколькими ядрами [2].

Есть много причин для использования параллельных вычислений. Из них выделяют три основных:

1) Одновременное выполнение нескольких вычислений.
2) Способность решения сложных задач.
3) Увеличение вычислительной производительности.

Традиционно параллельные вычисления при моделировании процессов, связанных с большой научно-экономической значимостью, называются GrandChallengeProblems (GCP). Как правило GCP имитируют некоторые явления, которые невозможно измерить с помощью экспериментов (погодные явления, физические процессыи т.д.). Это связано с тем, что моделирование таких процессов связано с обработкой больших объемов данных [3].

Чтобы использовать всю мощь многоядерных GPU для решения задач математического моделирования сложных процессов, были разработаны новые модели программирования. Из-за

широкой доступности интерфейсов прикладного программирования наиболее популярными технологиями являются CUDA, OpenCL и C++ AMP[4].

Производительность OpenCL, CUDA и C++ AMP нелегко сравнивать, потому что нет взаимно-однозначного соответствия между API. Например, не существует прямого аналога функции clSetKernelArg OpenCL в CUDA. ФункциюclSetKernelArg можно вызвать один раз для установки конфигурации, а ядро вызывать несколько раз. В CUDA это невозможно, так как аргументыфункции ядра устанавливаются, когда ядро вызывается: вызов ядра переводится на внутренние функции cudaSetupArgument и cudaLaunch.Кроме того, каждая среда языка программирования GPGPU при инициализации вызывает функций API в строгом порядке. Например, первое обращение к API CUDA может быть выполнен функциями cudaMalloc или cudaSetDevice в зависимости от того, принимает ли программист устройство по умолчанию или делает выбор из нескольких видеокарт. Однако, когда функция вызывается, следует долгое ожидание, потому что CUDA инициализируется. Но это не безнадежно, если разработчик тщательно продумывает реализуемое решение.

Блок кода в программе, написанной на одном языке, как правило, может быть переведен на эквивалентный код на другом языке программирования GPGPU. Например, фрагмент кода, который будет выполнять поиск и выбор устройства в CUDA, имеет эквивалентный блок кода в OpenCL. Эти блоки кода называются фазами. Для программ, работающих с GPU, можно выделить следующие фазы:

- Выбор графического процессора (GPU) для решения задачи и выделение памяти на этом устройстве.
- Копирование данных из памяти центрального процессора (CPU) в память GPU.
- Вызов ядра: запуск кода функции ядра, выполняющего решение задачи на GPU.
- Копирование данных из памяти GPU в память CPU.
- Освобождение ресурсов GPU.

На данный момент произведено немало исследований с использованием этих трех технологий для решения различных типов распараллеливаемых задач. Однако, принимая во внимание все преимущества и недостатки этих технологий, нельзя говорить об однозначном преимуществе какой-либо из перечисленных технологий. Также не было произведено технологическое и теоретическое сравнение этих технологий.Таким образом, целью данной работы является сравнение производительности технологий OpenCL, CUDA и C++ AMP на примере решения задачи прогнозирования электропотребления, а также оценка целесообразности и экономичности их использования по отношению к аппаратным вопросам, программному обеспечению, тенденциям развития технологий и используемых средств.

2. Параллельные вычисления

Параллельная программа имеет особый принцип работы. В отличие от последовательной, где одновременно выполняется только одно действие, в параллельных программах одновременно выполняются сразу несколько операций. Для этого необходимо основную задачу разбить на подзадачи, которые могут выполняться независимо друг от друга. Следовательно, проектировать алгоритм необходимо с учетом этой особенности. При разделении алгоритма на блоки, которые будут выполняться параллельно, необходимо также учитывать то, что блоки должны быть равными, т.е. их выполнение должно занимать одинаковое количество времени. Следует помнить, что при распараллеливании появляются временные затраты на передачу данных. Обычно пересылки требуют достаточно большого времени для своего осуществления, поэтому другой важной целью распараллеливания является минимизация объема и количества пересылок данных [5].

2.1 Технология CUDA

CUDA – это архитектура параллельных вычислений от NVIDIA, позволяющая существенно увеличить вычислительную производительность благодаря использованию GPU (графических процессоров). На сегодняшний день разработчики программного обеспечения, ученые и исследователи широко используют CUDA в различных областях, включая обработку видео и изображений, вычислительную биологию и химию, моделирование динамики жидкостей, восстановление изображений, полученных путем компьютерной томографии, сейсмический анализ, трассировку лучей и многое другое [6].

CUDA не использует графических API и свободна от ограничений, свойственных этим API. Основными преимуществами являются её простота – все программы пишутся на «расширенном» языке C, наличие хорошей документации, набор готовых инструментов и библиотек, кроссплатформенность. Для решения задач CUDA использует очень большое количество параллельно выполняемых нитей, при этом обычно каждой нити соответствует один элемент вычисляемых данных [7].

CUDA разбивает задачу в сети блоков, каждый блок содержит несколько потоков. Блоки могут работать в любом порядке. Блок должен выполнить от начала до завершения всю задачу, и может быть запущен на одном из N процессоров. Каждый процессор получает равное распределение задач [8].

2.2 OpenCL

OpenCL является новым отраслевым стандартом для реализации параллельных вычислений на различных современных центральных процессорах (CPU), графических процессорах (GPU) и других микропроцессорных конструкциях [9].

OpenCL определяет набор основных функций, которые поддерживаются всеми устройствами, а также опциональные функции, которые могут быть реализованы только на высокофункциональных устройствах, а также включает в себя механизм расширения, который позволяет производителям выставить уникальные аппаратные инструменты и экспериментальные программные интерфейсы для удобства разработчиков приложений. Не смотря на то что OpenCL является мульти платформенной технологией, это не гарантирует полную мобильность и корректность вычислений на всех аппаратных архитектурах[10].

OpenCL является основой, которая предоставляет возможность стандартизовать методы кодирования независимо от типов процессоров или производителей. Разработчики программного обеспечения будут иметь возможность писать параллельный код, который не зависит от аппаратной платформы. Спецификация языка OpenCL представляет собой расширения языка C для обеспечения параллельного программирования. При разработке программного обеспечения с использованием OpenCL, необходимы следующие инструменты[11]:

- OpenCL компилятор;
- Библиотека OpenCL RuntimeLibrary.

2.3 C++ AMP

В июне 2011 года на саммите AMDFusionDeveloperSummit 2011 Microsoft объявила о параллельном ускорителе AcceleratedMassiveParallelism (C++ AMP), представляющий собой расширение к языку программирования C++ для параллельного программирования графического процессора.

C++ AMP ускоряет выполнение кодапосредством использования преимуществ параллельной обработки данных на устройствах, таких как модуль графических вычислений (GPU) на выделенной видеокарте. С помощью C++ AMP возможно реализовать алгоритмы обработки многомерных данных таким образом, что их выполнение может быть ускорено с помощью параллелизма на различном оборудовании. Модель программирования C++ AMP включает в себя многомерные массивы, индексацию, функции работы с памятью и библиотеку математических функций. Можно использовать расширения языка C++ AMP для мониторинга, как данные пе-

ремещаются из CPU в GPU и обратно в режиме отладки, что дает возможность улучшить производительность [12].

Расширение C++ AMP имеет важное значение, поскольку оно нейтрализует разрыв между алгоритмами PRAM и их реализацией на GPU. Для запуска C++ AMP необходим только DirectX 11, установленный на Windows 7 или более позднюю версию системы. Если условие не выполнено, то код запустится на центральном процессоре в последовательном режиме [12].

3. Эксперименты и результаты

3.1 Формализация задачи

Для сравнения производительности представленных технологий параллельных вычислений рассмотрим пример помесячного прогнозирования электропотребления с применением искусственных нейронных сетей(ИНС).

Необходимо написать программы, которые будут производить прогнозирование электропотребления с применением каждой из трех технологий распараллеливания.

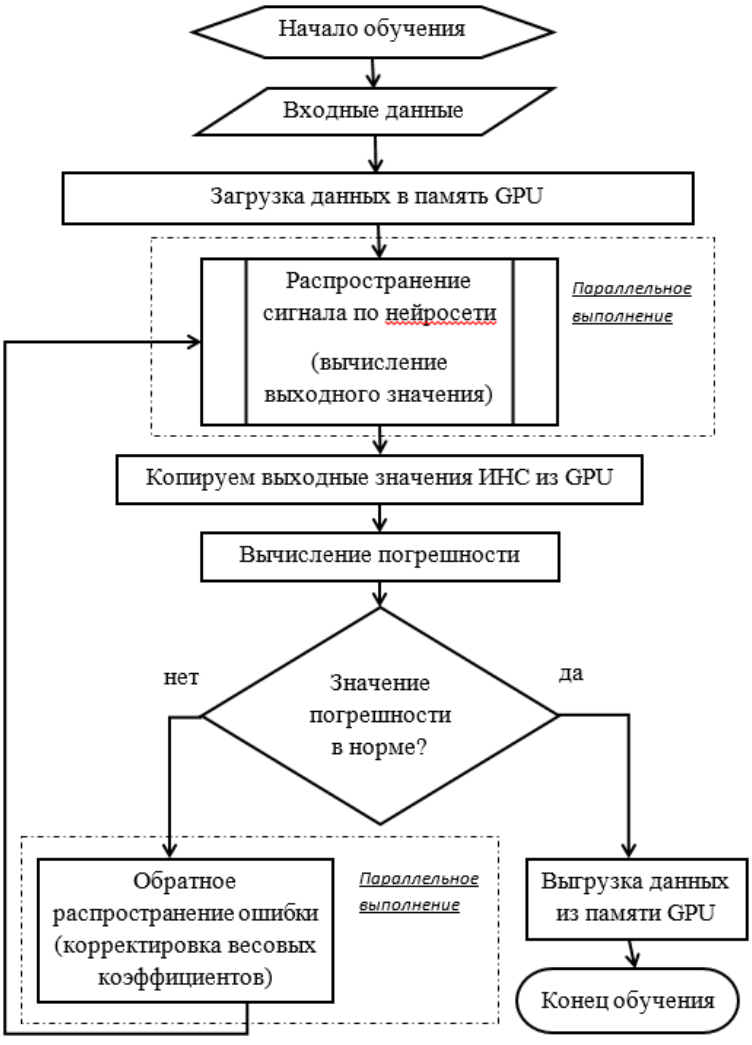

Рисунок 1 – Алгоритм обучения ИНС с распараллеливанием вычислений по технологии CUDA

За входные параметры ИНС приняты – средняя месячная температура, количество выходных в месяце, продолжительность дня, плановые объемы производства на месяц. А также потребление электроэнергии за предыдущий месяц, максимальное и минимальное месячные значения потребления за предыдущий год.Цель процесса обучения заключается в нахождении оп-

тимальных значений весовых коэффициентов, при которых нейронная сеть выдаёт значение в пределах заданной погрешности для всех обучающих пар.

Обучение ИНС будем проводить по алгоритму обратного распространения ошибки, схематически алгоритм представлен на рисунке № 1.Тот факт, что нейроны, находящиеся в одном слое, не зависят друг от друга, позволяет распараллелить вычислительный процесс. На схеме такие блоки выделены пунктирной линией.

В качестве функции активации будет использоваться нелинейная функция, представленная в формуле (0.1).

$$OUT = F(\xi) = \frac{1}{1+\exp(-\xi)} \quad 1$$

где $\xi = \sum_{k-1}^{L} x_k \omega_k$; x_k – входы нейрона; ω_k – синаптические веса входов; L – количество входов нейрона.

Для корректировки весовых коэффициентов будут использованы формулы (0.2) и (0.3). Формула коррекции весов для выходного слоя имеет вид:

$$\omega_{p-k}(i+1) = \omega_{p-k}(i) + \eta \delta_k OUT_p \quad 2$$

где i – номер итерации обучения; ω_{p-k} – синаптический вес, соединяющий нейрон p скрытого слоя с нейроном k выходного слоя; $\delta_k = OUT_k(1-OUT_k)(T_k - OUT_k)$; η – коэффициент скорости обучения; OUT_p – выход нейрона; T_k – целевое значение выхода нейрона. Формула для коррекции весов скрытого слоя записывается в виде:

$$\omega_{p-q}(i+1) = \omega_{p-q}(i) + \eta \delta_q OUT_p \quad 3$$

где ω_{p-q} – синаптический вес, соединяющий нейрон p предыдущего слоя с нейроном q скрытого слоя; OUT_p – выход нейрона; $\delta_q = OUT_q(1-OUT_q)\sum_{k=1}^{N} \delta_k \omega_{q-k}$; N – количество нейронов следующего слоя.

Таблица 1 – Распределение функций по фазам работы с GPU

	CUDA	OpenCL	C++ AMP
1	2	3	4
Выбор GPU и выделение памяти	cudaGetDeviceCount cudaGetDeviceProperties cudaSetDevice cudaMalloc	clGetPlatformIDs clGetPlatformInfo clGetDeviceIDs clGetDeviceInfo clCreateContext clCreateProgramWithSource clBuildProgram clGetProgramBuildInfo clCreateKernel clCreateBuffer clSetKernelArg clCreateCommandQueue	allocate vector <accelerator>get_accelerators() allocate vector::iterator accelerators.begin() accelerators.end() get_description() allocate array<> allocate extent<> allocate grid<>

Продолжение таблицы 1

Копирование данных из CPU в GPU	cudaMemcpy	clEnqueueWriteBuffer	copy
Вызов функции ядра	kernel<<<…>>>(…) cudaDeviceSynchronize	clEnqueueNDRangeKernel clWaitForEvents	parallel_for_each flush wait
Копирование данных из GPU в CPU	cudaMemcpy	clEnqueueReadBuffer	copy
Освобождение ресурсов GPU	cudaFree	clReleaseMemObject clReleaseKernel clReleaseContext clReleaseProgram	allocate grid<> allocate extent<> allocate array<> deallocate accelerator_view deallocate vector<accelerator> deallocate vector::iterator

В таблице № 1 представлены основные функции расширений языка программирования C++ по каждой технологии с распределением по фазам работы с GPU.

3.2 Эксперименты

- Все вычисления производились на персональном компьютере:
- Процессор (CPU): Intel(R) Core(TM) i5 760 2.8GHz;
- ОЗУ: 8,00 ГБ;

Тип системы: Windows 7 Профессиональная 64-разрадная;

Для выполнения программы с параллельной реализацией алгоритма обучения ИНС по технологии CUDA необходимо наличие графического процессора (видеокарты) NVIDIA. В данной работе использовалась видеокартаNVIDIAGeForceGT 240.Для выполнения программы на базе технологии OpenCLнеобходимо наличие библиотеки OpenCLRuntimeLibrary. Для технологии C++ AMP требуется наличие DirectX 11.

3.2.1 Выбор GPU и выделение памяти

В таблице № 2 представлено время, которое затрачено каждой из программ для предварительных установок, таких как выбор устройства, установка параметров, выделение памяти и т.д. На данном этапе программа OpenCL тратит значительно больше времени. Это связано с тем, что OpenCL тратит время на компиляцию функций, которые выполняют предустановки графического процессора. CUDA и C++ AMPне имеют этой проблемы, т.к. код предварительно скомпилирован на промежуточном языке.

Таблица 2 – Время предустановок перед копированием данных в память GPU

CUDA, мс	OpenCL, мс	C++ AMP, мс
41.3	460	79

3.2.2 Копирование данных между памятью CPU и GPU

В таблице № 3 показано время копирования данных в память GPU и обратно. Время копирования в память GPU для CUDA, OpenCL и C++ AMPприблизительно одинаково. Тем не менее, на копирование из памяти GPUдля C++ AMP потребовалось значительно больше времени чем для CUDA и OpenCL.

Таблица 3 – Время копирования данных

	CUDA, мс	OpenCL, мс	C++ AMP, мс
в память GPU	51.1	61.7	68.3
из памяти GPU	81.7	83.1	145

3.2.3 Вызов функции ядра

Результаты выполнения функций ядра представлены в таблице № 4. C++ AMP тратит значительно больше времени на накладные расходы (около 70 мс.), связанные с первым вызовом функции ядра. Тем не менее, время за 10 тыс. эпох обучений отличается не значительно.

Таблица 4 – Время работы функции ядра

Количество эпох	CUDA, мс	OpenCL, мс	C++ AMP, мс
1	0.536	0.544	71.5
10000	3053	3022	3157

4. Вывод

CUDA, OpenCL и C++ AMP являются расширениями языка программирования, позволяющими использовать мощности GPU для математических вычислений при решении сложных, нетривиальных задач, которые могут быть распараллелены.

Технология CUDA является фирменной архитектурой компании NVIDIA и может использоваться только на графических процессорах, производимых этой компанией. OpenCL и C++ AMP являются открытыми стандартами и могут использоваться на оборудовании различных производителей. Для разработчиков технология CUDA является более «зрелой», что подтверждается наличием API-интерфейсов более высокого уровня, которые значительно удобнее в использовании при написании кода. С другой стороны, C++ AMP позволяет переносить вычисления на GPU без внесения большого количества изменений в код программы. Реализация системы от Microsoft включена в популярную среду разработки VisualStudio начиная с версии 2012.OpenCL также активно улучшается, бесспорным преимуществом является тот факт, что OpenCL является очень популярной технологией среди поставщиков приложений, которые активно внедряют её в свои разработки.

Что же касается вычислительных мощностей, по результатам проведенных экспериментов видно небольшое преимущество технологии CUDA. Все эксперименты проводились по 5 раз, в таблицах представлены средние значения. Эти значения также совпадают с результатами других исследований, CUDA показало результат, не более чем на 30% лучше, чем OpenCL[13]. Худший результат у C++ AMP.

Основываясь на результатах, полученных в данной работе можно заметить, что для правильного выбораархитектуры при решении параллельных задач, необходимо учитывать не только значение производительности, но и доступность аппаратной части системы, стоимость разработки программного обеспечения, а также тенденции развития технологий.

Литература

1. Shi, Guochun. Application acceleration with the Cell broadband engine. Computing in Science and Engineering. / KindratenkoVolodymyr, PratasFrederico, Trancoso Pedro, Gschwind Michael.2010; 12(1): 76–81.

2. S. M.Smith. The GPU Computing Revolution. [S.l.: s.n.] 2011.

3. D. Kirk. NVIDIA CUDA software and GPU parallel computing architecture. [S.l.]: NVIDIA Corporation, 2008.

4. L. C. M. Paula et al. Parallelization of a Modified Firefly Algorithm using GPU for Variable Selection in a Multivariate Calibration Problem. International Journal of Natural Computing Research, v. 4, n. 1, p. 31-42, 2014.

5. Антонов, А. С. Введение в параллельные вычисления. М., 2002.

6. Таранов, Р.В. Вычисления на GPU. Применение библиотек ускоренных по технологии CUDA. Фундаментальные и прикладные исследования в современном мире. Материалы IX Международной научно-практической конференции. Том 1. 2015г. С. 100-109.

7. Боресков, А. В. Основы работы с технологией CUDA. / Харламов, А. А. ДМК Пресс, Москва, 2010.

8. Shane Cook. CUDA Programming. A Developer's Guide to Parallel Computing with GPUs. Elsevier Inc. 2013.

9. MunshiAaftab. OpenCL Specification Version 1.0. Dec, 2008. http://www.khronos.org/registry/cl/

10. Stone, J. OpenCL: a parallel programming standard for heterogeneous computing systems. / Gohara, D. Shi, G.Computing in science & engineering, v. 12, n. 3, p. 66, 2010.

11. Tsuchiyama, R. The OpenCL Programming Book. [S.l.]: Fixstars, 2010

12. Интернет ресурс: https://msdn.microsoft.com/ru-ru/library/hh265136.aspx. Дата обращения: 13.04.16

13. Fang, J. A comprehensive performance comparison of CUDA and OpenCL. / Varbanescu, A. L. Sips, H. In: Parallel processing international conference (ICPP), 2011, Taipei City.Taipei City: [s.n.]. p. 216-225, 2011.

УДК 697.34:697.444

НАРУШЕНИЯ ТЕПЛОВОГО РЕЖИМА ЗДАНИЙ ПРИ ВЫСОКИХ ТЕМПЕРАТУРАХ НАРУЖНОГО ВОЗДУХА

Т.А. Рафальская

Часто в переходный период отопительного сезона наблюдаются нарушения температурного режима жилых помещений. Проведённое исследование позволяет спрогнозировать и скорректировать режимы работы тепловых пунктов в этот период.

Often in a transition period of a heating season infringements of a temperature mode of premises, when temperature of internal air considerably below or above demanded are observed. Carried out research allows to predict and correct operating modes of heating points during this period.

Ключевые слова: система теплоснабжения, отопление, горячее водоснабжение, тепловые сети, температурный режим зданий.

Keywords: heat supply system, heating, hot water supply, heat supply networks, temperature mode of buildings.

В настоящее время расчёт основных параметров работы систем теплоснабжения ведётся для двух точек температурного графика центрального регулирования тепловой нагрузки: при наружной температуре, расчётной для проектирования отопления $t_{но}$, определяемой по [1], и при температуре наружного воздуха, соответствующей точке нижнего излома температурного графика $t_{ни}$, при которой температура воды в подающей магистрали теплосети остаётся постоянной независимо от температуры наружного воздуха (τ_1 на рис. 2, *а*).