УДК 004.032.24

РЕШЕНИЕ ПРИКЛАДНЫХ ЗАДАЧ С ИСПОЛЬЗОВАНИЕМ ГЕТЕРОГЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Т.Е. Лощухина, В.А. Дорофеев

Томский политехнический университет
E-mail: Loshuhina.Tatiana@yandex.ru

Актуальность работы *обусловлена широким распространением графических процессоров и возможностью их использования для повышения производительности с помощью организации параллельных вычислений.*
Цель работы*: выявить положительные и отрицательные стороны применения параллельных гетерогенных вычислительных систем при решении нетрадиционных, а именно неграфических задач, задействованных в информационных системах прикладного уровня, провести сравнительный анализ нескольких технологий гибридных параллельных вычислений и рассмотреть практический пример применения гетерогенных вычислений.*
Методы исследования: *реализация алгоритма поиска оптимального маршрута на выбранных вычислительных устройствах, оценка производительности каждой реализации.*
Результаты: *изучена зависимость производительности алгоритма поиска оптимального маршрута от количества узлов исходного графа и количества вычислительных устройств. Разработан алгоритм обновления рабочей группы при размерности исходного буфера некратной размерности этой группы.*

Ключевые слова:
Параллельный алгоритм, гибридные технологии, гетерогенные вычислительные системы, графовая модель, методы синхронизации.

Введение

Идея использования гибридных вычислительных систем появились сравнительно недавно, около десяти лет назад, однако к настоящему моменту этот подход получил существенное распространение и выделился в отдельное направление в сфере информационных технологий. Под гибридными системами подразумевается любая система, в аппаратную конструкцию которой входит два и более блока вычислительных модулей. Самой распространённой конструкцией гибридных систем на сегодня принято считать аппаратную связку из процессоров общего назначения и графических процессоров (или карт-ускорителей для вычислений). В основном такие системы применяются для организации параллельных вычислений. При этом параллельные вычисления на графических процессорах уже являются не только средством повышения производительности решения задач, нацеленных на наложение фильтрации графических текстур и повышения детализации графических элементов, но и нацелены на массово-параллельную обработку любых данных, над которыми должны быть проведены алгебраические операции любой сложности.

Параллельный алгоритм Флойда для гетерогенной системы

Для разработки функционала под гибридные системы в настоящий момент наиболее актуальными являются следующие технологии: CUDA (поддерживается только для видеокарт и карт-ускорителей от NVIDIA), OpenCL, ATI APP (работает только на видеокартах и ускорителей от AMD), C++ AMP. Так как изначально мы нацеливались на кроссплатформенность в исследовании производительности и удобства реализации алгоритма на конкретной платформе, то и технологиями, которые больше всего подходили под указанные требования, были OpenCL и C++ AMP. Чтобы произвести оценку эффективности применения указанных технологий при решении определённого класса задач (а именно подготовка, обновление и доступ к данным), было решено разработать ряд динамически подключаемых библиотек (DLL), ответственных за применение функциональных возможностей рассматриваемой платформы. Таким образом, вся работа была сведена к нескольким этапам:

1) конструирование интерфейса для управления и контроля доступных платформ и сопряжённых с ними устройств;
2) разработка DLL-библиотеки, решающей выбранную задачу с использованием заданной платформы;
3) оценка времени, затраченного на исполнение процедур;
4) сравнение производительностей параллельной гетерогенной системы с производительностью при использовании стандартной многопоточности и последовательным вариантом выполнения тех же задач.

В качестве примера в данной статье рассматривается решение задачи выбора оптимального маршрута на карте. Начальный этап – инициализация данных – определяет выбор алгоритма. На этом этапе формируется граф, состоящий из узлов и представляющий собой пересечения улиц на карте, и рёбер, отражающих расстояния между вершинами. Очевидно, что алгоритм для поиска маршрута должен быть годным для распараллеливания, и это позволило существенно сократить список алгоритмов поиска оптимального маршрута в графе.

В качестве алгоритма поиска маршрута был выбран алгоритм Флойда. Суть метода состоит в

следующем: в распоряжении алгоритма имеются две матрицы (матрица значений длины маршрута между указанными узлами $D_{n\times n}$ и матрица смежности $S_{n\times n}$, указывающая промежуточные пункты на пересечении стартовой и финальной позиции), размера $n\times n$, где n – число вершин в графе [1].

Для того чтобы подготовить данные к использованию алгоритмом, они инициализируются следующим образом:

1. Если из пункта «А» существует путь в пункт «Б», то $\{d_{ij}=l;\ s_{ij}=j\}$, где s_{ij} – элемент матрицы $S_{n\times n}$, $_{dij}$ – элемент матрицы $D_{n\times n}$, i – порядковый номер пункта «А» и номер строки матрицы, j – порядковый номер пункта «Б» и номер столбца матрицы, l – длина маршрута от «А» до «Б».
2. Если между пунктами «А» и «Б» нет связи, то $\{d_{ij}=-1;\ s_{ij}=-1\}$, равно как и в случае при $\{i=j\}$.

Как сказано выше, алгоритм исполняется столько раз, сколько есть вершин в графе, при этом номер итерации учитывается при обновлении данных. Так, на k-той итерации производятся следующие действия над каждым элементом матрицы (одновременно): если $\{d_{ik},d_{kj}\}$ положительны и $\{i<>j,j<>k,i<>k\}$, то в случае $d_{ij}>(d_{ik}+d_{kj})$, как и $d_{ij}=-1$, производится замена $\{d_{ij}=d_{ik}+d_{kj};s_{ij}=k\}$, в противном случае замена не производится.

Результатом алгоритма будут две матрицы: первая будет указывать, существует ли путь между двумя конкретными вершинами (вне зависимости от того, смежные они или нет); вторая позволит выстроить путь от узла «А» к узлу «Б» путём восстановления порядка следования вершин. Распишем механизм восстановления пути подробно. Изначально имеется только подмножество из двух вершин $\{i,j\}$. Если $s_{ij}<>j$, то подмножество дополняется третьей промежуточной вершиной между предварительно рассматриваемыми, а итоговое подмножество примет вид $\{i,s_{ij},j\}$. После каждого случая повторного добавления промежуточного узла путь восстановления маршрута производится заново до тех пор, пока равенство $s_{ij}=j$ не будет истинным для каждой смежной пары [2].

Особенности разработки функционала под гетерогенные системы

Под функцией ядра (или кёрнел-функцией) понимается функция, которая исполняется в изолированном потоке на отдельном вычислительном модуле исполняемого устройства для обработки элемента (или «рабочего элемента») исходного массива данных, а под «рабочей группой» – двумерный массив из рабочих элементов. Любая система параллельного гетерогенного типа располагает ограниченным количеством параллельно запускаемых копий кёрнел-функций. Это объясняется тем, что аппаратные ресурсы вычислительной системы ограничены. Как правило, используемая платформа гетерогенной системы содержит вспомогательный инструментарий для определения допустимой размерности рабочего блока данных [3], как, например, OpenCL. Однако одна и та же технология, в зависимости от платформы, может иметь разный набор программных интерфейсов, что ограничивает нас при решении задачи общим для всех платформ набором интерфейсов.

Вопрос о допустимой размерности рабочей группы можно легко обойти, установив её размерность равной числу, кратному 16 [4, 5]. То есть рабочая группа 16×16 самый выигрышный вариант при работе с параллельными технологиями для гибридных вычислительных систем. Более того, число реальных данных, подлежащих обновлению, может значительно превышать размер допустимого числа одновременно запускаемых потоков и быть некратным размерности рекомендуемой рабочей группы. Что касается последнего аспекта, то он довольно просто обходится, если исполняемому устройству известна исходная размерность буфера рабочих данных. Рассмотрим данный этап подробнее (рис. 1).

Введём пояснение принятых обозначений: M – размерность матрицы буфера данных; L – размерность допустимых областей рабочей группы; $N=(M/L)$ – число полных рабочих групп на исходный буфер; $\Delta=(M\%L)$ – размерность остатка буфера данных; $t=(L-\Delta)$ – размерность избыточного обновления буфера; $A=(M-L)$ – позиция $(N+1)$-й рабочей группы.

На рис. 1 изображена ситуация, когда размерность буфера данных не кратна значению допустимой рабочей группы и приведены два способа обхода данной проблемы. В первом случае (рис. 1, *а*) обновление производится с учётом позиции расположения $(N+1)$-го блока рабочей группы. При таком способе организации данных не изменяется время исполнения, так как обновление функциональных элементов производится единовременно; более того, при условии, что расположение данных удовлетворяет условию $j>(A+t)$, нетронутые данные будут обновлены, а те, которые обновлялись прежде, останутся нетронутыми, так как возможное преобразование над ними уже выполнено. Ситуация, показанная на рис. 1, *б*, никаких дополнительных расчётов не требует, при этом значение размерности матрицы буфера данных обязательно должно быть передано устройству исполнения. Значение аргумента M для кёрнел-функции имеет два значения: *во-первых*, позволяет поместить обновлённый элемент на ту же исходную позицию буфера данных; *во-вторых*, позволит контролировать выход за границы памяти исходного диапазона буфера данных, что вызовет сбой работы эксплуатируемого приложения. При этом второе утверждение действительно только для метода обновления, изображённого на рис. 1, *б*, тогда как первое заявление справедливо для обеих ситуаций. Если в качестве аргумента M будет рассматриваться параметр L (то есть значение размерности блока рабочей группы), то обрабатываемые блоки будут расположены последовательно в исходном буфере, что нарушит их изначальное блочное расположение.

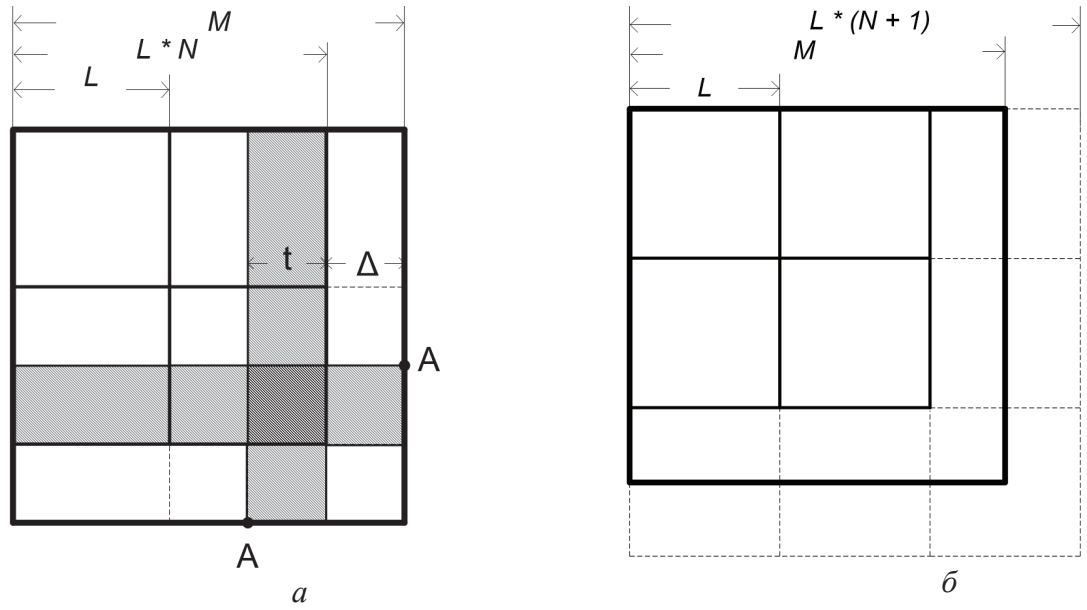
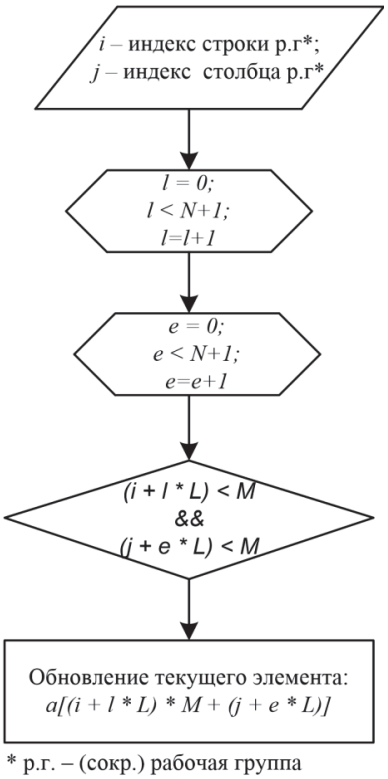


Рис. 1. *Варианты обновления буфера данных*

Вернёмся к программной реализации, а именно к способу построения такой организации памяти в рамках параллельных вычислительных технологий. Некоторые технологии параллельных гетерогенных вычислительных систем дают возможность запускать на параллельное исполнение не только элементы в пределах одной рабочей группы, но и блоки из рабочих групп. Однако такими возможностями обладает не каждая технология, что можно легко обойти с помощью вложенных циклов внутри функции-ядра. Такой подход не приветствуются, так как противоречит идее параллельности исполнения. Кроме того, время программного исполнения в этом случае возрастает по экспоненциальному закону, что сильно снижает производительность. В то же время с помощью вложенных циклов можно обновить все элементы, расположенные на одной позиции в каждой из рабочих групп. Блок-схема для такого механизма приведена на рис. 2.

После того как подходящий алгоритм выбран и решён вопрос со способом инициализации исходных данных, встает вопрос о способах оценки эффективности применения гибридных вычислений на прикладных задачах. Эксплуатируемая система представляет собой источник информации о местности в виде карты с нанесёнными на неё объектами, которые в совокупности образуют ориентированный взвешенный граф. Чтобы подготовить данные к исполнению алгоритма, формируется матрица $V_{n \times n}$ из векторов v_{ij}. Каждый вектор v_{ij} свидетельствует о наличии либо отсутствии пути между вершинами i и j. Если путь есть, то $v_{ij}=\{x_i,y_i,x_j,y_j\}$. После пересылки этой информации исполнительному устройству в распоряжении программы окажется буфер данных с начальными значениями для запуска алгоритма поиска оптимального маршрута между указанными вершинами, а также время, за которое была произведена операция. При регулярном наращивании количества объектов и измерениях времени выполнения задачи были построены графики зависимости времени выполнения от количества объектов с учётом следующих факторов: используемая технология вычислений для гибридных систем и исполнительное устройство.



* р.г. – (сокр.) рабочая группа

Рис. 2. *Обновление всего буфера исходных данных в пределах одного блока рабочей группы*

Экспериментальные исследования проводились на компьютере с процессором Intel Core i5–4570T с частотой 2,9 ГГц и 16 Гб оперативной памяти. В качестве тестируемого графического устройства был выбран интегрированный в данный процессор Intel HD Graphics 4600, поддерживающий OpenCL. Для сравнения приведены графики времени решения той же задачи в многопоточном режиме, реализуемом стандартными средствами платформы .NET и в однопоточном режиме (рис. 3–7).

Из приведённых диаграмм видна экспоненциальная зависимость времени исполнения от количества объектов при эксплуатации системы на аппаратуре с поддержкой гибридных вычислений. При тестировании той же системы с использованием стандартных многопоточных способов распараллеливания видно, что временная зависимость стремится к константному значению, что является наиболее предпочтительным результатом, однако время выполнения в данном режиме

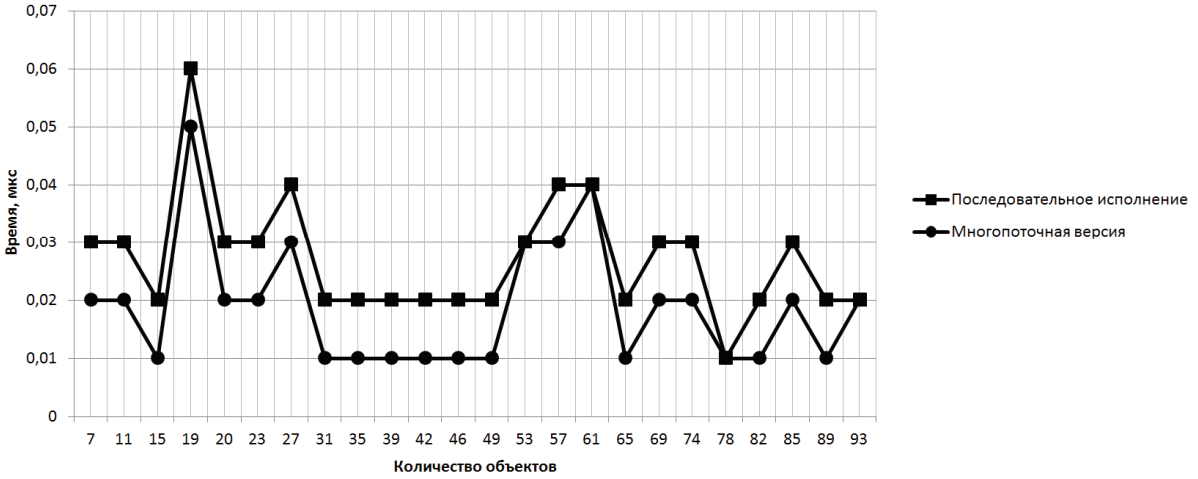


Рис. 3. *Зависимость времени выполнения от количества объектов в однопоточной и многопоточной версиях инициализации*

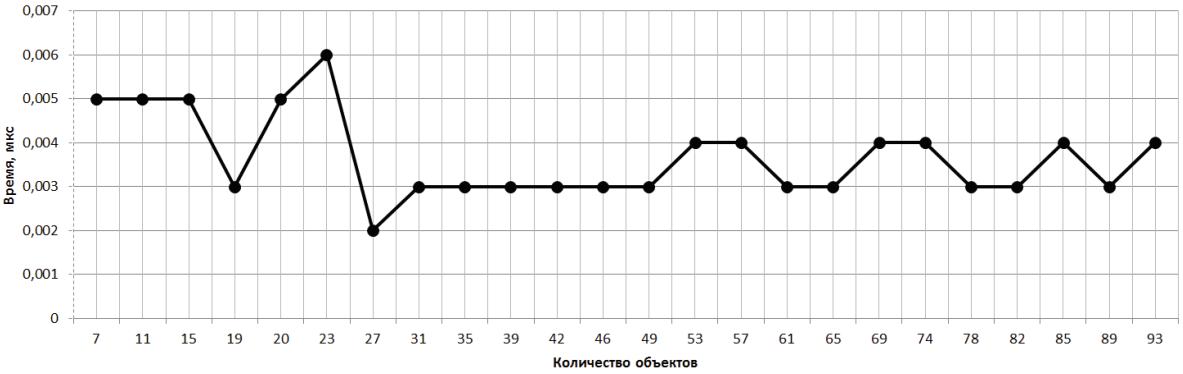


Рис. 4. *Зависимость времени выполнения от количества объектов при инициализации при использовании технологии C++AMP*

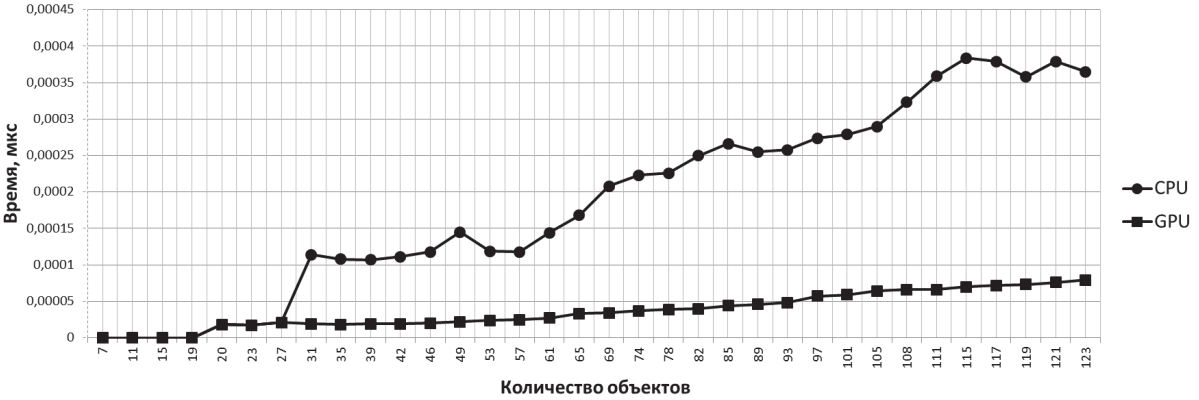


Рис. 5. *Временная зависимость от числа объектов при инициализации с использованием OpenCL-стандарта для разных устройств исполнения*

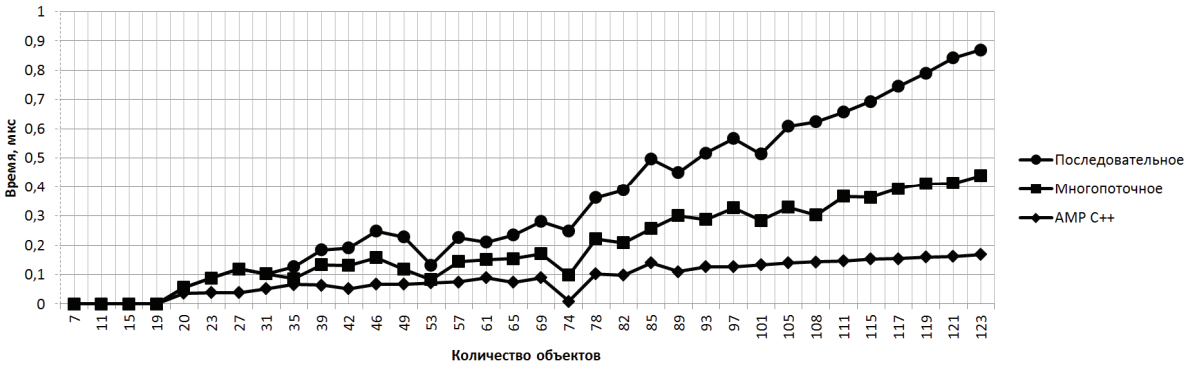


Рис. 6. *Временная зависимость от числа объектов при исполнении алгоритма в многопоточном, однопоточном режимах и с использованием параллельной технологии для гетерогенной системы*

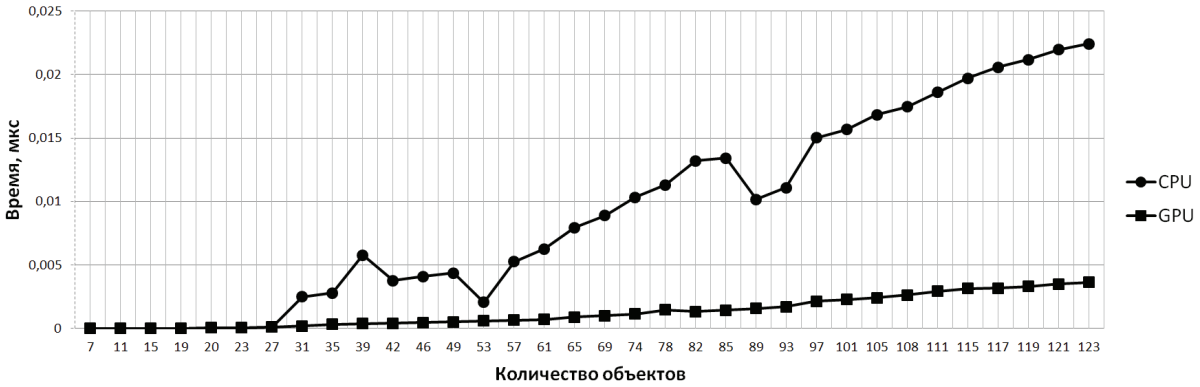


Рис. 7. *Временная зависимость от числа объектов при исполнении алгоритма поиска оптимального маршрута с использованием OpenCL-стандарта на разных вычислительных устройствах*

значительно выше, чем при запуске того же инструментария на гетерогенной вычислительной системе.

Выводы

В ходе выполнения работы была изучена зависимость производительности алгоритма поиска оптимального маршрута от количества узлов исходного графа и количества вычислительных устройств. Эксперименты показали, что наиболее предсказуемые результаты получаются на платформе OpenCL: прирост ускорения почти постоянный, что в определённых пределах можно использовать для экстраполяции значений при масштабировании приложения.

Другим результатом работы стал алгоритм обновления рабочей группы в тех случаях, когда размерность исходного буфера данных не кратна размерности этой группы, или же размерность буфера неизвестна в момент проектирования приложения. Предложенный способ обеспечивает корректное заполнение рабочей группы и последующее извлечение обработанных данных без повреждения структуры исходных данных.

СПИСОК ЛИТЕРАТУРЫ

1. Модели и структуры данных / В.Д. Далека, А.С. Деревянко, О.Г. Кравец, Л.Е. Тимановская. – Харьков: ХГПУ, 2000. – 241 с.
2. Гергель В.П. Теория и практика параллельных вычислений. – М.: Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2007. – 423 с.
3. Scarpino M. OpenCL in action. – S.I.: Manning, 2012. – 458 p.
4. Мот Д. Введение в C++AMP на основе кода // Журнал MSDN magazine. 2012. URL: http://msdn.microsoft.com/ru-ru/magazine/hh882446.aspx (дата обращения: 15.04.2013).
5. Мот Д. Введение в мозаичное заполнение C++ AMP // Журнал MSDN magazine. 2012. http://msdn.microsoft.com/ru-ru/magazine/hh882447.aspx (дата обращения: 15.04.2013).

Поступила 26.05.2013 г.

UDC 004.032.24

SOLVING THE APPLIED TASKS USING THE HETEROGENEOUS COMPUTING SYSTEMS

T.E. Loshchukhina, V.A. Dorofeev

Tomsk Polytechnic University

The urgency *of the discussed issue is caused by the wide spreading of graphics processors and the ability to use them to improve the performance by parallel computing.*
The main aim of the study *is to identify the positive and negative aspects of using the heterogeneous parallel computing systems for solving non-traditional, i.e. non-graphics tasks involved in application level of information systems; to a analyze several hybrid parallel computing technologies and to review the practical example of heterogeneous computing usage.*
The methods used in the study: *algorithm for finding the optimal route implementation on the selected computing devices, evaluation of the performance of each implementation.*
The results: *The authors have studied the dependence of the performance of the algorithm for finding optimal route on a number of the original graph nodes and the number of computing devices; they developed the algorithm for updating the working group at the source buffer dimension that is not multiple to the group dimension.*

Key words:
Parallel algorithm, hybrid technology, heterogeneous computing, graph model, synchronizing methods.

REFERENCES

1. Daleka V.D., Derevyanko A.S., Kravec O.G., Timanovskaya L.E. *Modeli i struktury dannykh* [Models and data structures]. Harkov, HGPU, 2000. 241 p.
2. Gergel V.P. *Teoriya i praktika parallelnykh vychisleniy* [Parallel computing theory and practice]. Moscow, Internet-Universitet Informacionnyh Tehnologiy; BINOM. Laboratoriya znaniy, 2007. 423 p.
3. Scarpino M. *OpenCL in action*. S.I., Manning, 2012. 458 p.
4. Mot D. Vvedenie v C++AMP na osnove koda. *Zhurnal MSDN magazine*, 2012. Available at: http://msdn.microsoft.com/ru-ru/magazine/hh882446.aspx (accessed 15 April 2013).
5. Mot D. Vvedenie v mozaichnoe zapolnenie C++ AMP. *Zhurnal MSDN magazine*, 2012. Available at: http://msdn.microsoft.com/ru-ru/magazine/hh882447.aspx (accessed 15 April 2013).

УДК 004.822; 004.4'2

АНАЛИЗ СТРУКТУРНЫХ ИЗМЕНЕНИЙ В ОНТОЛОГИЯХ НА ЯЗЫКЕ OWL 2

И.А. Заикин

Томский политехнический университет
E-mail: zaikin@tpu.ru

Актуальность работы обусловлена необходимостью коллективной распределённой разработки веб-онтологий, одной из задач которой является сравнение версий онтологий и анализ изменений. Для повышения эффективности совместной работы необходимо предоставить разработчикам обработанную информацию об изменениях в онтологии (какие сущности были изменены, добавлены или удалены в новой версии онтологии), а также о том, какие именно изменения были произведены над каждой сущностью. В работе предложен метод анализа структурных изменений, полученных с использованием прямой семантики языка OWL 2, позволяющий выявлять изменённые, добавленные и удалённые из онтологии сущности, а также сопоставлять каждой затронутой сущности множество изменений. Приведены основные алгоритмы, решающие данную задачу, описание их программной реализации, а также результаты тестирования производительности алгоритмов.

Ключевые слова:
Онтология, OWL 2, аксиома, сущность, изменение.

Введение

Онтологией в информатике называют формальное описание некоторой предметной области или задачи на языке, понятном как людям, так и вычислительным машинам. Одним из самых распространённых языков описания онтологий в настоящее время является язык веб-онтологий OWL 2 [1], стандартизированный консорциумом всемирной паутины (W3C), с использованием которого ежегодно создаётся множество онтологий, сложность которых также возрастает. Разработка сложных онтологий требует участия не одного, а нескольких специалистов: специалистов по языку онтологий, специалистов в предметной области, специалистов по управлению качеством. Совместная разработка сложных онтологий невозможна без инструментальных средств поддержки коллективной работы. В разработке программного обеспече-