ТЕХНОЛОГІЇ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

УДК 681.03

А.В. Чистяков, И.С. Ислямова

О НЕКОТОРЫХ ОСОБЕННОСТЯХ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ СОВРЕМЕННЫХ КОМПЬЮТЕРОВ

Национальный авиационный университет

Кафедра инженерии программного обеспечения

Научный руководитель – Иванова Л.Н., к.т.н., доцент

Рассмотрены некоторые вопросы разработки прикладного программного обеспечения для современных компьютеров. Предложена технология планирования вычислений при создании программного обеспечения для решения вычислительных задач на компьютерах гибридной архитектуры. Приведены некоторые результаты компьютерных экспериментов.

Розглянуто труднощі розробки прикладного програмного забезпечення для сучасних комп'ютерів. Запропоновано технологію планування обчислень при створенні програмного забезпечення для розв'язування обчислювальних задач на комп'ютерах гібридної архітектури. Наведено деякі результати комп'ютерних експериментів.

An overview of some aspects the development of application software for modern computers. The technology of planning computer solutions by creating software to solve computational problems on computers of the hybrid architectures. Some results of computer experiments.

Ключевые слова: параллельное программирование, планирование вычислений, гибридные архитектуры, NVIDIA CUDA, GPGPU, OpenMP, MPI, PVM.

Введение

Современная наука находится в стадии динамичного развития. В каждой отрасли народного хозяйства возникают задачи, требующие обработки огромных объемов информации с высоким быстродействием и точностью. Объемы расчетов, необходимых для решения задач в различных сферах науки и промышленности постоянно растут, поэтому для их решения приходится строить все более мощные вычислительные системы.

Долгое время повышение производительности традиционных одноядерных процессоров в основном осуществлялось за счет последовательного увеличения тактовой частоты (около 80% производительности процессора определяла именно тактовая частота) с одновременным увеличением количества транзисторов на одном кристалле. Однако дальнейшее повышение тактовой частоты (при тактовой частоте более 3,8 ГГц чипы просто перегреваются!), натыкается на ряд фундаментальных физических барьеров. Кроме того, преимущества более высокой тактовой частоты сильно уменьшаются из-за задержек при обращении к памяти, так как время доступа к памяти не соответствует возрастающим тактовым частотам.

Типы современных компьютеров

Основным способом увеличения мощности современных компьютеров стало объединение все большего числа вычислительных элементов

в огромные вычислительные комплексы. Однако этот процесс в последние годы несколько замедлился из-за достижения пиковых возможностей технологии их производства. Недостатком такого увеличения вычислительных мощностей является то, что компьютеры получают все большие размеры, и как следствие становятся более сложными и гораздо более дорогими в эксплуатации, уже сегодня для них необходимо строить отдельные строения с развитой инфраструктурой, а потребление электроэнергии уже сопоставимо с потреблением небольшого городка.

Параллельно с совершенствованием центрального процессора (CPU) в последнее время происходит очень активное развитие графических процессоров (GPU) [1].

Ноябрь 2008 − фирма Intel представила линейку 4-ядерных центральных процессоров Intel Core i7, в основу которых положена микроархитектура нового поколения Nehalem. Процессоры работают на тактовой частоте 2,6 − 3,2 ГГц.

Декабрь 2008 − началась поставка 4-ядерного центрального процессора AMD Phenom II 940 (кодовое название − Deneb). Работает на частоте 3 ГГц.

Май 2009 − компания AMD представила версию графического процессора ATI Radeon HD 4890 с тактовой частотой ядра, увеличенной с 850 МГц до 1 ГГц. Это первый графический процессор, работающий на частоте 1 ГГц. Вычислительная мощность чипа, благодаря увеличению частоты, выросла с 1,36 до 1,6 терафлоп. Процессор содержит 800 (!) вычислительных ядер, поддерживает видеопамять GDDR5, DirectX 10.1, ATI CrossFireX и все другие технологии, присущие современным моделям видеокарт

Основными отличиями GPU по сравнению с CPU являются:

- архитектура максимально направлена на увеличение скорости расчета текстур и сложных графических объектов;
- пиковая мощность типичного GPU намного выше, чем у CPU;
- благодаря специализированной конвейерной архитектуре GPU намного эффективнее в обработке графической информации, чем CPU.

Сегодня наиболее известные типы компьютеров, которые используются для построения больших проектов, такие.

1). ***Мэйнфреймы*** или большие компьютеры − это компьютеры, созданные для обработки больших объемов информации. Наиболее крупный их производитель − фирма IBM. Отличаются исключительной надежностью, высоким быстродействием, очень большой пропускной способностью каналов ввода-вывода. К ним могут присоединяться тысячи терминалов (дисплеев с клавиатурой) или персональных компьютеров для работы пользователей. Большинство крупных корпораций, банков, зарубежных правительственных учреждений обрабатывают свои данные именно на таких компьютерах. Хотя они могут стоить миллионы долларов, спрос на них не падает, так как централизованные хранение и обработка данных обходятся дешевле, чем обслуживание с помощью распределенных систем обработки данных, состоящие из сотен и тысяч персональных компьютеров.

2). ***Суперкомпьютеры*** − вычислительная машина, которая значительно превосходит по своим техническим параметрам большинство существующих компьютеров. Как правило, современные суперкомпьютеры представляют собой большое количество высокопроизводительных серверных компьютеров, соединенных друг с другом локальной высокоскоростной магистралью для достижения максимальной производительности в рамках распараллеливания вычислительной задачи. Эти компьютеры предназначены для решения задач, требующих огромных объемов вычислений. Основные потребители супер-ЭВМ − ученые, авиаторы, судостроители, военные, метеорологи, геологи, и многие другие. Супер-ЭВМ стоят десятки миллионов долларов (если не дороже), их производят лишь несколько крупных фирм, например, Cray Researsh (сейчас это подразделение фирмы Silicon Graphics), Hitachi и др. Такие компьютеры целесообразно использовать в глобальных системах, грид-среде и других компьютерных сетях.

3). ***Рабочие станции*** (в том числе с параллельной архитектурой) − персональные компьютеры, дополненные мощностями (комплекс технических средств, программное обеспечение, устройства), которые позволяют работать с большими объемами данных. Они имеют производительность, как у самых мощных персональных компьютерах или даже в несколько раз больше. Это компьютерная система, которая зачастую работает в составе компьютерной сети и настроена на выполнение задач инженеров, экономистов, программистов и других специалистов.

4). ***Вычислительный кластер*** − это совокупность компьютеров (вычислительных узлов с распределенной памятью), объединенных

единой коммуникационной сетью, используемых для решения одной или нескольких проектов. Современные вычислительные кластеры в мире могут насчитывать несколько тысяч вычислительных узлов. На сегодняшний день широко используется для решения научно-технических задач, в том числе в сетях.

5). ***Гибридные системы*** [2] − объединение нескольких вычислительных узлов, каждый из которых снабжен, как минимум, одним графическим устройством. Специалисты в разных развитых странах мира отмечают, что именно за таким компьютером они видят будущее компьютеризации.

Системы программирования для решения математических задач

Появление мощных компьютеров новой архитектуры привело к созданию новых средств программирования. Наиболее популярными компьютерами для высокопроизводительных вычислений являются многоядерные (компьютеры MIMD-архитектуры): каждый узел у них есть система с распределенной памятью. Для создания программного обеспечения на такие компьютеры появились специальные программные средства PVM (Parallel Virtual Machine) [3], MPI (Message Passing Interface) [4], [5] и другие.

PVM общедоступный программный пакет, позволяющий объединять разнородный набор компьютеров в общий вычислительный ресурс («виртуальную параллельную машину») и предоставляющий возможности управления процессами с помощью механизма передачи сообщений. PVM − это продукт продвижения гетерогенных сетевых исследовательских проектов, распространяемый авторами и институтами, в которых они работают. Поддерживается языками программирования Фортран, Си, Си++.

Чаще всего для написания программ для компьютеров с распределенной памятью используется система MPI − стандартизированный интерфейс для построения программ по модели обмена сообщениями. Существуют бесплатные и коммерческие реализации для различных суперкомпьютерных платформ.

Преимущества:

- возможность использования в языках Фортран, Си, Си++, возможность совмещения обменов сообщениями и вычислений, несколько режимов передачи сообщений;
- широкий набор коллективных операций;
- широкий набор редукционных операций;
- удобные средства именования адресатов сообщений;
- возможность задания типа передаваемой информации.

Недостатки:

- слишком сложный для прикладного программиста (программирование на низком уровне, необходимо знать: как поделить между процессорами работу программ, как поделить между процессорами данные, как организовать доступ к удаленным данным или обмен данными, которые в процессе вычислений нужны одному процессору, в то время как хранятся на другом).

Для программирования многопоточных приложений на многопроцессорных системах с общей памятью (SMP-системах) используется система OpenMP (Open Multi-Processing) [6]. OpenMP − это система программирования, которая расширяет последовательную программу (Фортран, Си, C++) набором директив распараллеливания. Предполагается, что потоки выполняются параллельно на машине с несколькими процессорами (количество процессоров не обязательно должно быть больше или равно количеству потоков). Задачи, выполняемые потоками параллельно, также как и данные, требуемые для выполнения этих задач, описываются с помощью специальных директив препроцессора соответствующего языка − *прагм*. Программа начинает свое выполнение как один процесс (главная нить), пока не встретится параллельная область программы (ограничивается директивами *PARALLEL* и *END PARALLEL*). При входе в параллельную область главная нить порождает некоторое число подчиненных ей нитей, образуя группу нитей. Все операторы программы, находящиеся в параллельной конструкции, выполняются всеми нитями группы параллельно, пока не произойдет выход из параллельной области или не встретится одна из конструкций распределения работ. При выходе из параллельной конструкции все порожденные ранее нитки сливаются с главной, которая продолжает выполняться дальше последовательно. “Инкрементальное распараллеливание” программы, которое используется в OpenMP идеально подходит для разработчиков, желающих быстро распараллелить свои вычислительные программы с большими параллельными циклами. Разработчик не создает новую параллельную программу, а просто последовательно добавляет в текст последовательной программы соответствующие OpenMP-директивы.

В последние годы стали активно использоваться графические процессоры для неграфических вычислений (моделирование физических процессов, обработка сигналов, вычислительная математика / геометрия, операции с базами данных, вычислительная биология, вычислительная экономика, компьютерная визуализация и т. д.). Однако при использовании графических процессоров для неграфических задач были значительные трудности, связанные с тем, что не было никакого стандартного интерфейса для программирования. Разработчики использовали OpenGL или Direct3D, но это было очень неудобно, потому что приходилось знать и учитывать много технических особенностей вычислительной среды графических ускорителей.

Устройство GPU отличается от того, как устроен CPU. Прежде всего, отличается работа с памятью. Так, не все центральные процессоры имеют встроенные контроллеры памяти, а у всех GPU обычно есть по несколько контроллеров, вплоть до восьми 64-битных каналов в чипе NVIDIA GT200. Кроме того, на видеокартах применяется более быстрая память, и в результате видеочипам доступна в разы большая пропускная способность памяти, что весьма важно для параллельных расчетов, оперирующих с огромными потоками данных.

В универсальных процессорах большие количества транзисторов и площадь чипа идут на буферы команд, аппаратное предсказание ветвления и огромные объемы начиповой кэш-памяти. Все эти аппаратные блоки нужны для ускорения исполнения немногочисленных потоков команд. Видеочипы тратят транзисторы на массивы исполнительных блоков, управляющие потоками блоки, разделяемую память небольшого объема и контроллеры памяти на несколько каналов. Вышеперечисленное не ускоряет выполнение отдельных потоков, оно позволяет чипу обрабатывать нескольких тысяч потоков, одновременно исполняющихся чипом и требующих высокой пропускной способности памяти.

Кроме того, есть существенные отличия в кэшировании. Универсальные центральные процессоры используют кэш-память для увеличения производительности за счет снижения задержек доступа к памяти, а GPU используют кэш или общую память для увеличения полосы пропускания. CPU снижают задержки доступа к памяти при помощи кэш-памяти большого размера, а также предсказания ветвлений кода. Эти аппаратные части занимают большую часть площади чипа и потребляют много энергии. Видеочипы обходят проблему задержек доступа к памяти при помощи одновременного исполнения тысяч потоков − в то время, когда один из потоков ожидает данных из памяти, видеочип может выполнять вычисления другого потока без ожидания и задержек.

Есть множество различий и в поддержке многопоточности. CPU исполняет 1-2 потока вычислений на одно процессорное ядро, а видеочипы могут поддерживать до 1024 потоков на каждый мультипроцессор, которых в чипе несколько штук. И если переключение с одного потока на другой для CPU стоит сотни тактов, то GPU переключает несколько потоков за один такт.

Кроме того, центральные процессоры используют SIMD (одна инструкция выполняется над многочисленными данными) блоки для векторных вычислений, а видеочипы применяют SIMT (одна инструкция и несколько потоков) для скалярной обработки потоков. SIMT не требует, чтобы разработчик преобразовывал данные в векторы, и допускает произвольные ветвления в потоках.

Вкратце можно сказать, что в отличие от современных универсальных CPU, видеочипы предназначены для параллельных вычислений с большим количеством арифметических операций. И значительно большее число транзисторов GPU работает по прямому назначению — обработке массивов данных, а не управляет исполнением (flow control) немногочисленных последовательных вычислительных частей программы.

В 2005−2006 г. корпорация NVIDIA (один из крупнейших производителей графических, медиа - и коммуникационных процессоров) начала разработку единого и удобного стандарта, и в 2007 г. представила первоначальную версию технологии CUDA для программирования на GPU. Разработчики ставили цель, чтобы GPU использовались не только для создания изображений в 3D приложениях, но и применялись в других параллельных расчетах.

Технология NVIDIA CUDA [7] − архитектура CUDA основана на концепции одна команда на множество данных (Single Instruction Multiple Data, SIMD) и понятии мультипроцессора. Концепция SIMD подразумевает, что одна инструкция позволяет одновременно обработать множество данных. Мультипроцессор − это многоядерный SIMD процессор, позволяющий в каждый определенный момент времени выполнять на всех ядрах только одну инструкцию. Каждое ядро мультипроцессора скалярное, т.е. оно не поддерживает векторные

операции в чистом виде. В CUDA есть понятие устройство (*device*), что обозначает программа на видеокарте, поддерживающая драйвер CUDA, а также − хост (*host*), т.е. программа в обычной оперативной памяти компьютера, использующая CPU и выполняющая управляющие функции по работе с устройством.

CUDA предполагает специальный подход к разработке, не совсем такой, как принят в программах для CPU. Нужно помнить о разных типах памяти, о том, что локальная и глобальная память не кэшируется и задержки при доступе к ней гораздо выше, чем у регистровой памяти, так как она физически находится в отдельных микросхемах. Особенностью архитектуры CUDA является блочно-сеточная организация, необычная для многопоточных приложений (рис. 1). При этом драйвер CUDA самостоятельно распределяет ресурсы устройства между потоками. На рис. 1. ядро обозначено как *Kernel*.

CUDA приложения программируются на C++ с использованием специального SDK и представляет собой комбинацию кода, который выполняется на CPU и ядер (*kernels*), которые выполняются на GPU. Каждая задача делится на независимые части − подзадачи (подобно тому, как делается при помощи OpenMP), которые решаются с помощью набора взаимодействующих между собой нитей. CUDA использует большое количество отдельных нитей для вычислений, часто каждому вычисляемому элементу соответствует одна нить. Все нити группируются в иерархию − *grid, block, thread*. Распределение вычислений и данных должно быть согласованным и выполняться с учетом чередования последовательных и параллельных частей программы. Верхний уровень − *grid* − отвечает ядру и объединяет все нити, которые выполняют данное ядро. *Grid* представляет собой одномерный или двухмерный массив блоков (*block*). Каждый блок является одно / двух / трехмерный массив нитей (*threads*). При этом каждый блок − полностью независимый набор взаимодействующих между собой нитей, нити из разных блоков не могут между собой взаимодействовать.

Выполнение расчетов на GPU показывает отличные результаты в алгоритмах, использующих параллельную обработку данных. То есть, когда одну и ту же последовательность математических операций применяют к большому объему данных. При этом лучшие результаты достигаются, если отношение числа арифметических инструкций к числу обращений к памяти достаточно велико. Это предъявляет меньшие требования к управлению исполнением, а высокая плотность математики и большой объем данных отменяет необходимость в больших кэшах, как на CPU. При большом количестве GPU система CUDA позволяет ускорить решение задачи в сотни раз.

На сегодняшний день уже создано много библиотек программ для математических расчетов, особенно по вычислительной математике. В основном в них используется CUDA версии 2.1. Одним из самых больших недостатков в этой системе программирования есть то, что связь между отдельными GPU поддерживается только через CPU, что существенно замедляет работу программы и усложняет процесс программирования.

Однако в феврале 2011 года компания NVIDIA анонсировала новую версию набора инструментов NVIDIA CUDA 4.0 для разработки параллельных приложений с помощью графических процессоров. Новый набор инструментов поддерживает технологию NVIDIA GPU Direct 2.0, которая обеспечивает равноправную связь между GPU в рамках одного сервера или рабочей станции, что упрощает и ускоряет мультипроцессорное программирование и работу приложений. Унифицированная виртуальная адресация (UVA) организует единое адресное пространство для основной системной памяти и памяти GPU, что делает параллельное программирование еще быстрее и проще. Набор также включает библиотеки примитивов Thrust Си++ − структур данных и алгоритмов Си++ с открытым кодом для параллельных расчетов.

Для эффективного решения задач на компьютерах гибридной архитектуры, которые могут насчитывать несколько сотен многоядерных CPU и GPU, создаются специальные алгоритмы и программы по гибридной технологии [8]. Гибридная технология − это два уровня параллелизма − параллелизм задачи на подзадачи и параллелизм внутри подзадачи. В случае объединения нескольких вычислительных многоядерных узлов с несколькими графическими устройствами задача распараллеливается на отдельные подзадачи путем распределения между ядрами процессора элементов массивов и витков циклов с помощью функций библиотеки OpenMP, которые в свою очередь распараллеливаются на GPU. Обмен данными между отдельными узлами выполняется с помощью функций библиотеки MPI: организуется взаимодействие подзадач − обмен между ними граничными значениями. Основной недостаток подхода: программисту нужно знать и уметь

использовать сразу несколько разных моделей параллелизма и разные инструментальные средства.

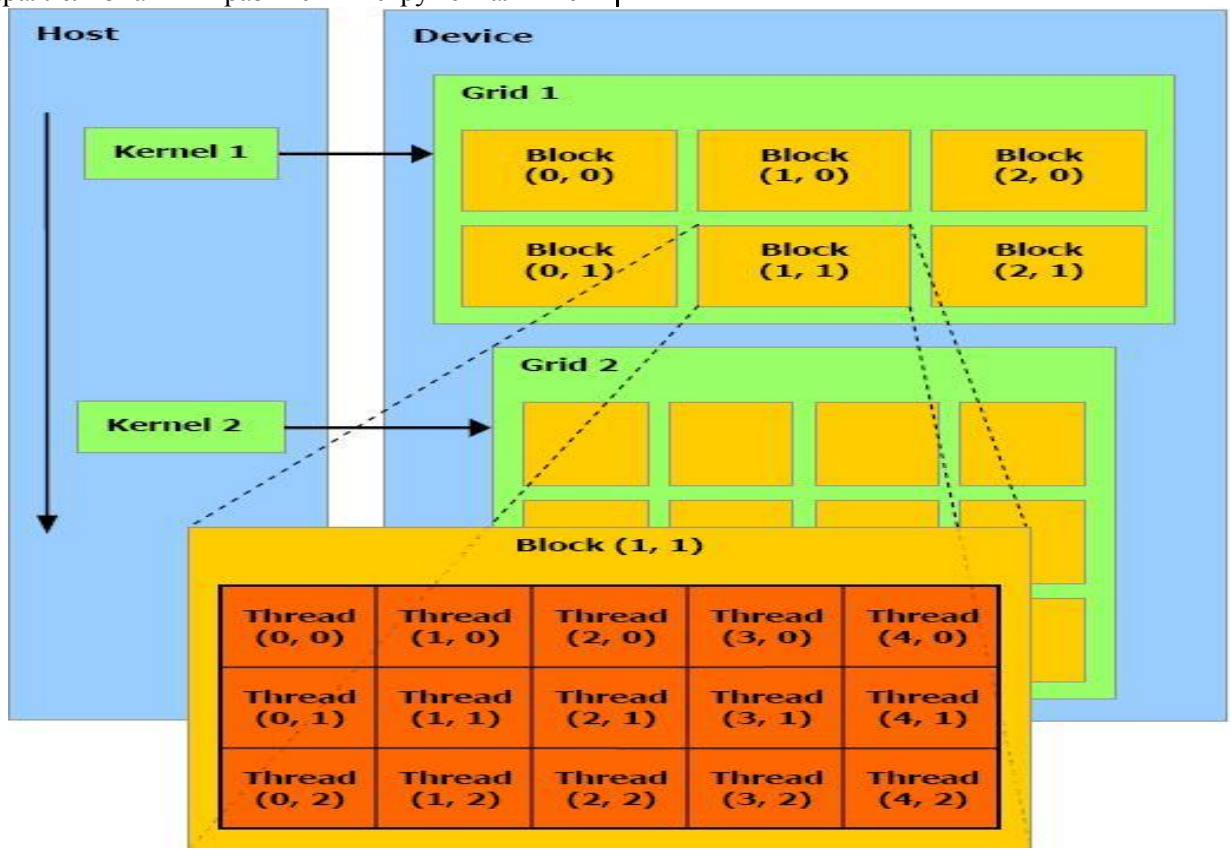


Рис. 1. Организация потоков при использовании технологии CUDA

Таким образом, создание алгоритмов и программ для решения задач с использованием той или иной архитектуры, которая может быть наиболее эффективной по времени выполнения задачи, является достаточно непростой работой. Пользователю необходимо не только обладать навыками программирования MPI, OpenMP, CUDA и т.д., но также хорошо знать архитектуру компьютера, вычислительные возможности CPU и GPU для оптимального выбора количества процессов, ядер, подключения графического ускорителя с целью скорейшего получения решения задачи с высокой точностью.

Планирование вычислений при разработке прикладного программного обеспечения

Вместе с бурным развитием вычислительной техники бурно развивается область компьютерной программной технологии (software engineering), которая требует периодического пересмотра подходов к созданию прикладного программного обеспечения.

Успешное использование CPU и GPU, высокая производительность компьютера зависит от успешной разработки вычислительного программного обеспечения. В последнее время все чаще отмечается, что для облегчения работы пользователей на сложных вычислительных комплексах необходимо создавать библиотеки и пакеты прикладных программ с некоторым дружественным интерфейсом по детальной планировке вычислений на гибридных системах с правильным выбором ее вычислительных компонентов. Сначала необходимо разработать эффективные алгоритмы и программы для одного GPU и CPU, а также для нескольких этих ресурсов одновременно (гибридное задание), после чего определить ключевые параметры задачи и топологию компьютера, при которых решение задачи будет получено с самым высоким быстродействием и с оптимальным количеством необходимых вычислительных ресурсов.

Используя созданное программное наполнение и имея знания об условиях использования его компонент на различных архитектурах гибридной системы, можно разрабатывать прикладное программное обеспечение с автоматическим планированием вычислительных средств, что значительно повысит производительность работы как пользователя, так и ком-

пьютера. При этом необходимо предоставлять пользователю соответствующие рекомендации относительно целесообразности использования компонент гибридной системы для решения конкретной задачи.

Покажем на простом примере умножения двух матриц $C = AB$, как зависит время решения задачи от ее объема и архитектуры компьютера. Как известно, именно эта математическая операция чаще всего выполняется при решении прикладных задач и на ее выполнение тратится много компьютерного времени (n^3-операций, n − порядок матриц).

На рис. 2 схематически показано распараллеливание задачи на гибридной системе, имеющей три CPU и три GPU.

Напомним, что умножение двух квадратных матриц выполняется по формуле:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \ i, j = 1,\ldots, n \qquad (1)$$

Для эффективного использования вычислительных узлов компьютера гибридной архитектуры при выполнении этой математической операции можно организовать планирование вычислений по такой схеме.

1). Если размеры матриц такие, что матрицы полностью размещаются в кэш-памяти компьютера, то задачу (1) можно решать на CPU (один процесс) обычным алгоритмом, состоящим из трех циклов.

2). Если размеры матриц такие, что матрицы не помещаются в кэш-память, то целесообразно использовать блочные алгоритмы [9], при этом необходимо определить размеры блоков матриц такие, чтобы они полностью размещались в кэш-памяти.

3). Для каждого компьютера программу блочного умножения матриц можно сделать более эффективной, если выполнить разложение внутреннего цикла с учетом конвейера инструкций компьютера.

4). Если CPU − многоядерный процессор, то можно распараллелить задачу на подзадачи между ядрами, используя OpenMP (коммуникационные потери не должны превысить время выполнения арифметических операций).

5). Если CPU имеет графический ускоритель, то целесообразно организовать вычисления по программе с использованием CUDA: основная часть программы (задание исходных данных, управление GPU, обмены информацией между CPU и GPU) будет выполняться на CPU (*хост*), а параллельная часть программы (выполнение циклов) будет выполняться на GPU (*device*). При этом время выполнения задачи может уменьшиться в десятки раз (для больших размеров матриц и при большом количестве графических ускорителей − в сотни раз).

6). Если компьютер состоит из нескольких CPU и GPU, то необходимо предусмотреть программы с распараллеливанием по двухуровневой модели: заранее определить оптимальное количество процессов для наиболее быстрого решения задачи. Затем распараллелить задачу на подзадачи с помощью MPI, при этом выполнение циклов каждой подзадачи на GPU организовать с помощью CUDA. Запустить задачу на выполнение на установленное количество процессов.

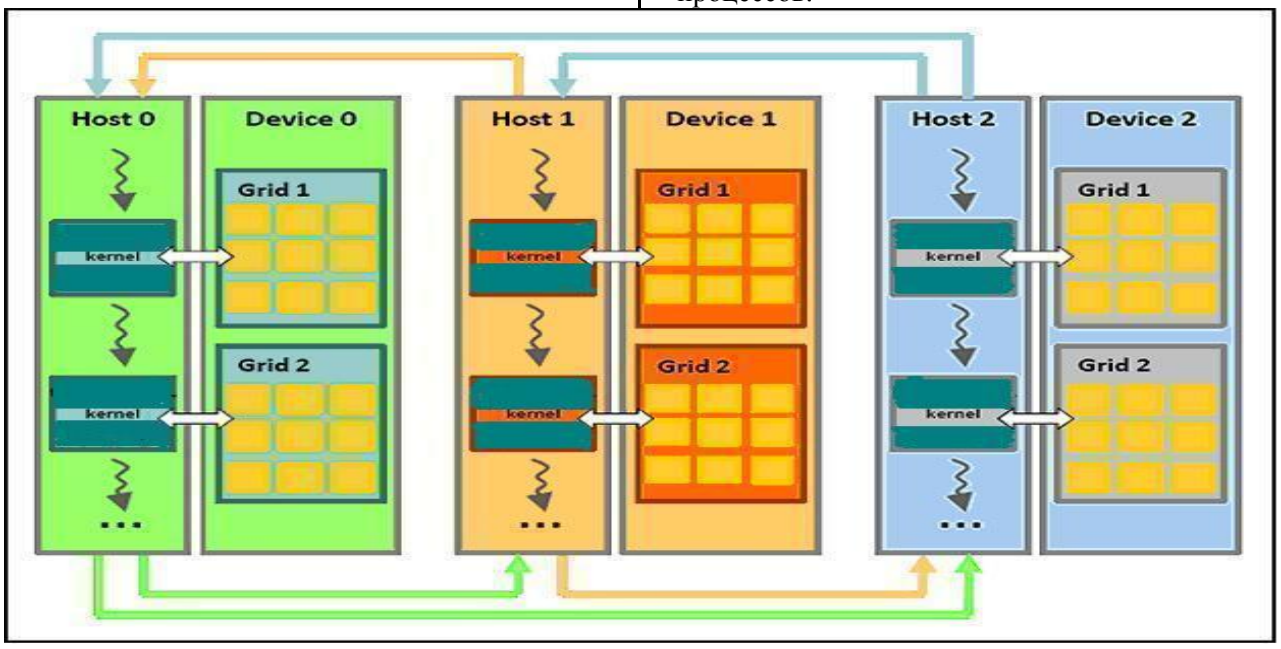


Рис. 2. Схема умножения матриц на компьютере с тремя CPU и тремя GPU

В табл.1-3 демонстрируются результаты перемножения матриц различными вариантами на гибридном компьютере, состоящего из двух вычислительных узлов, соединенных коммуникационной сетью Infiniband. В вычислительном узле используется два четырехядерных процессора Intel Core i7 930. Тактовая частота ядра составляет 2.8 ГГц. На каждом узле есть графическая видеокарта GeForce GTX480. Время выполнения задач приведено в сек.

В табл. 1 представлены результаты решения задачи на одном процессоре тремя вычислительными схемами из первых трех вариантов. С наибольшей скоростью получили решение на одном процессоре блочным алгоритмом (размер блока 80) с разложением внутреннего цикла.

Из табл. 2 видно, что время решения задачи (начиная с порядка 1000), которая распараллелена на несколько ядер на процессоре с помощью системы OpenMP, на двух ядрах несколько больше чем обычным способом на одном ядре (коммуникационные расходы достаточно большие по сравнению с временем выполнения арифметики). Однако, начиная с трех ядер, задача для всех представленных порядков решается значительно быстрее.

В табл. 3 показано как улучшается время решения задачи на четырех процессах с ростом порядка матриц по параллельной программе с использованием MPI+CUDA (задача распараллеливалась на CPU и GPU) по сравнению с временем решения по параллельной программе с использованием MPI (задача распараллеливалась только на CPU).

Таблица 1. Время умножения матриц вариантами 1−3

Порядок матриц	Вариант 1	Вариант 2	Вариант 3
500	1	0.23	0.1
1000	13	5	3
2000	118	54	38
3000	452	128	61

Таблица 2. Время умноження матриц на различном количестве ядер компьютера

Порядок матриц	Обычный алгоритм (1 ядро)	Алгоритм с использованием OpenMP		
		2 ядра	3 ядра	4 ядра
500	1	0.8	0.61	0.44
1000	13	18.9	7.4	4.01
2000	118	160	54	32
3000	452	612	255	109

Таблиця 3. Время умноження матриц на 4 процессах с использованием MPI и MPI+CUDA

Порядок матриц	Время решения на CPU	Время решения на CPU+GPU
1000	2.8	4.7
2000	25	5.14
3000	90	6.35

Выводы

В настоящее время, когда архитектура компьютерных комплексов очень усложнилась, усложняется также и программное обеспечение, которое должно учитывать как параметры задачи, так и условия эффективного решения на компьютерной системе.

Современной тенденцией в мире информационных технологий является переход к новой парадигме прикладного программного обеспечения – автоматическое планирование решения задачи с целью наименьших затрат компьютерного времени при оптимальном использовании многоядерных GPU и CPU.

Многие авторы отмечают, что для компьютерных систем сегодня особенно актуально создание прикладного программного обеспечения с автоматическим выбором вычислительных ресурсов, алгоритмов и программ решения задач.

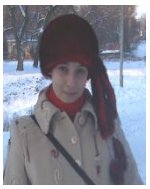
Как следует из приведенных результатов экспериментов, использование вычислительных ресурсов гибридной системы дает значительные преимущества по эффективности вычисления (уменьшение времени решения задачи в десятки раз), если они учитывают как параметры задачи, так и вычислительные ресурсы. С ростом количества многоядерных процессоров и графических ускорителей время решения больших задач может уменьшиться в сотни раз.

Литература

1. Графические процессоры для высокопроизводительных вычислений: http://gpu.parallel.ru/
2. Свенч А.А., Файзуллин Р.Т., Хныкин И.Т. Гибридная суперкомпютерная система. // Сборник трудов Международной научной конференции «Параллельные вычислительные технологии 2011», Московский государственный университет имени М.В. Ломоносова, Москва, 2011 г. (http://kvap.ru/pavt2011_cd/full.html).
3. Немнюгин С.А., Стесик О.Л. Параллельное программирование для многопроцессорных вычислительных систем. − СПб.: БХВ-Петербург, 2002. − 400 с.
4. К.Ю. Богачёв Основы параллельного программирования. − М.: БИНОМ. Лаборатория знаний, 2003. − 342 с.
5. Корнеев В.Д. Параллельное программирование в MPI. - Новосибирск: Изд-во СО РАН, 2000. – 213 с.
6. Чистяков А.В., Ислямова И.С. Метод и технологии параллельного программирования при решении прикладных задач. // Інженерія програмного забезпечення. № 3 – 2010. – С.37 − 50.
7. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. – М.: ДМК Пресс, 2010. – 232 с.
8. Chorley M.J., Walker D.W. Performance analysis of a hybrid MPI/OpenMP application on multi-core clusters. Jornal of Computatinal Science 1(3). – 2010. – P. 168–174.
9. Fox G.C., Otto S.W. and Hey A.J.G. Matrix Algorithms on a Hypercube I: Matrix Multiplication Parallel Computing. – 1987. – 4. – P. 17–31.

Сведения об авторах

Чистяков Алексей Валериевич – студент 5 курса Национального авиационного университета. Научные интересы - параллельное программирование, прикладное и объектно-ориентированное программирование, инженерия программного обеспечения, компьютерные сети.
E-mail: bmwwmb@gala.net



Ислямова Инна Сергеевна – студентка 5 курса Национального авиационного университета. Научные интересы - параллельное программирование, прикладное и объектно-ориентированное программирование, объектно-ориентированные базы данных, инженерия программного обеспечения.
E-mail: Inna_Islyamova@ukr.net

Иванова Любовь Николаевна – к.т.н., доцент кафедры инженерии программного обеспечения Национального авиационного университета, научные интересы - параллельное программирование, инженерия программного обеспечения.
E-mail: lubov.ivanova@livenau.net