2. *Соболь И.М., Картышов С.В., Кульчицкая И.А., Левитан Ю.Л.* О многокритериальной оптимизации математических моделей // Математическое моделирование. – 1994. – № 6. – С. 85-93.
3. *Бондарев А.Е.* Решение задачи оптимизационного анализа с помощью параллельных вычислений // Новые информационные технологии в автоматизированных системах: Материалы пятнадцатого научно-практического семинара. – М.: МГИЭМ, 2012. – С. 89-94.

Статью рекомендовал к опубликованию д.ф.-м.н. А.К. Алексеев.

Бондарев Александр Евгеньевич – Институт прикладной математики им. М.В. Келдыша РАН; e-mail: bond@keldysh.ru; 125047, г. Москва, Миусская пл., 4; тел.: 84992507817; отдел № 2; старший научный сотрудник; к.ф.-м.н.

Bondarev Alexander Evgen'evich – Keldysh Institute of Applied Mathematics RAS; e-mail: bond@keldysh.ru; 4, Miusskaya sq., Moscow, 125047, Russia; phone: +74992507817; department № 2; senior researcherэ; dr. of phis.-math. sc.

УДК 519.688

С.В. Поляков, Ю.Н. Карамзин, О.А. Косолапов, Т.А. Кудряшова, С.А. Суков

ГИБРИДНАЯ СУПЕРКОМПЬЮТЕРНАЯ ПЛАТФОРМА И РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ РЕШЕНИЯ ЗАДАЧ МЕХАНИКИ СПЛОШНОЙ СРЕДЫ СЕТОЧНЫМИ МЕТОДАМИ*

Рассмотрена проблема разработки параллельных приложений для решения задач механики сплошной среды на современных вычислительных системах с гибридной архитектурой, включающей центральные и графические процессоры. Для их решения сформулирована концепция гибридных параллельных вычислений, включающая анализ особенностей гибридного вычислителя и его программного оснащения, а также предложения по реализации параллельных программ. В частности, предложены три основные модели параллельных вычислений, использующие графические вычислители эпизодически, постоянно или в тесной связке с центральными процессорами. Также рассмотрены специфические проблемы реализации сеточных численных алгоритмов на гибридных вычислителях. Приведён пример реализации конечно-объёмной схемы на неструктурированной сетке при решении систем уравнений Эйлера и Навье–Стокса на графическом ускорителе.

Механика сплошной среды; математическое моделирование; численные подходы на основе метода сеток; параллельные алгоритмы; комплексы программ для гибридных вычислительных систем.

S.V. Polyakov, Yu.N. Karamzin, O.A. Kosolapov, T.A. Kudryashova, S.A. Soukov

HYBRID SUPERCOMPUTER PLATFORM AND APPLICATIONS PROGRAMMING FOR THE SOLUTION OF CONTINUOUS MECHANICS PROBLEMS BY GRID METHODS

The developing of parallel applications for the solution of continuum mechanics problems on modern computer systems with hybrid architecture (including the central and graphical processors) was considered. For the solving of the problem the conception of hybrid parallel computations was formulated. This conception includes the analysis of both architecture of modern hybrid computer and its software and the special ways for the construction of parallel programs. In particular, three main models of the parallel calculations, using graphic processors incidentally, constantly or in a close sheaf with the central processors were offered. Specific problems of reali-

* Работа выполнена при поддержке Российского фонда фундаментальных исследований (проекты № 11-01-12086-офи-м, 12-01-00339, 12-01-00345).

zation of grid numerical algorithms on hybrid computers were also considered. The example of implementation of the finite-volume scheme on unstructured grid at the decision of systems of the equations of Euler and Navier-Stokes on graphic accelerators was given.

Continuum mechanics; mathematical modeling; numerical approach based on grid technique; parallel algorithms; software tools for hybrid computer systems.

Введение. Особенностью современного этапа развития прикладных параллельных программных средств моделирования является переход к гибридным программным решениям. Причиной этому послужило широкое внедрение гибридных компьютерных архитектур в повседневную практику научных исследований. Способствовало этому процессу развитие программных сред и языков программирования, ориентированных на спецвычислители, среди которых в первую очередь следует отметить графические ускорители. Главной задачей с в настоящее время, на наш взгляд, является формирование единой концепции гибридного параллельного программирования, которую можно было бы использовать в последующие годы как основу развития параллельных приложений в науке и технике.

В настоящей работе авторы попытались сформулировать свое видение проблемы применительно к одному широко распространённому классу задач, а именно задачам механики сплошной среды (МСС), решаемым методом сеток. В результате была разработана концепция гибридных параллельных вычислений, которую можно положить в основу программной части гибридной суперкомпьютерной платформы. Среди множества вопросов, примыкающих к этой проблеме, были рассмотрены два: алгоритмы решения задач и особенности их реализации на доступных вычислительных системах.

Особенности современной гибридной архитектуры. Для достижения поставленных целей обратимся сначала к обзору аппаратных и программных средств, имеющихся в наличии у большинства исследователей. Сразу отметим, что в качестве аппаратных решений будем различать 3 класса вычислительных систем (ВС): 1) настольные или мобильные персональные компьютеры (ПК), 2) бюджетные кластеры (БК) лабораторий и небольших организаций, 3) суперкомпьютерные вычислительные системы (СКВС) крупных организаций и центров коллективного пользования (ЦКП). По сути эти решения отличаются количеством независимых вычислительных узлов и способами их интеграции в единую вычислительную систему. Архитектура современного вычислительного узла этих систем де-факто является гибридной. Рассмотрим её подробнее.

На рис. 1 показана гибридная архитектура современного вычислительного узла, характерная практически для любой современной компьютерной системы. Здесь предполагается, что узел имеет k центральных (CPU) и p графических (GPU) процессоров, а также k банков общей оперативной памяти. Каждый CPU имеет l ядер, поддерживающих по m конвеерных устройств обработки данных, и доступ к любому банку оперативной памяти через общую шину данных. Каждый GPU имеет q мультипроцессоров и $q*r*n$ ядер, сгруппированных в r блоков по n конвеерных устройств обработки данных, и доступ к собственной оперативной памяти (прямой доступ), к памяти других GPU (в том числе внешних) по шине PCI Express, а также к оперативной памяти своего узла. В отличие от CPU в графических ускорителях имеется также возможность управления кэш-памятью.

При использовании подобного вычислительного узла в настольном исполнении получаем мощную рабочую станцию, вполне пригодную для проведения научных расчётов среднего уровня. При объединении небольшого количества (порядка 10–50) узлов с подобной архитектурой с помощью среднескоростной сети (например, Gigabit Ethernet) получаем бюджетный кластер террафлопного диапазона производительности. Если для объединения применяется одна или несколько

высокоскоростных сетей (10-Gigabit Ethernet, InfiniBand и т.д.), а число узлов измеряется сотнями и более, получаем суперкомпьютерную систему вплоть до петафлопного уровня производительности.

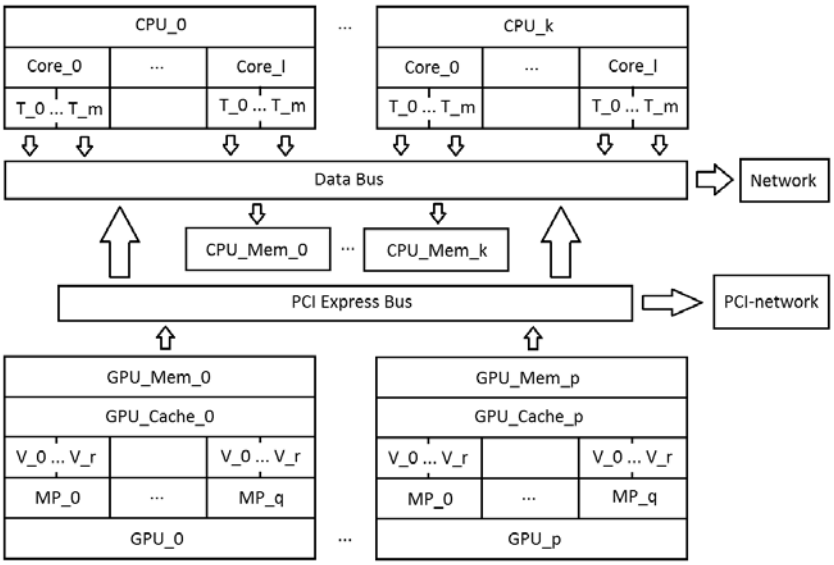

Рис. 1. Гибридная архитектура современного вычислительного узла кластера или персонального компьютера

С точки зрения управляющей программной среды ситуация складывается следующим образом. В настольных и мобильных аппаратных решениях в основном используется операционная система Windows (далее ОС Windows), а также Mac OS X; существенно реже встречается операционная система UNIX в версии Linux (далее ОС Linux). Как правило, это уже 64-битные версии ОС. Для реализации кластерных решений используются в основном ОС Linux, существенно реже ОС Windows в версии Windows Server или Windows Cluster Server. Таким образом, можно отметить, что подавляющее число современных вычислителей управляется ОС Windows и Linux.

Специфика разработки неспециальных приложений для вычислителей с универсальной центральной архитектурой отличается тем, что прикладному программисту фактически не требуется знать детальную структуру узла, а лишь использовать количество ядер l или параллельных потоков $l*m$, а также объём доступной оперативной памяти, которые легко определить во время исполнения кода. При этом инициализация многопоточного приложения внутри вычислительного узла производится за очень короткое время (порядка нескольких микросекунд) и может повторяться многократно в течение всего цикла решения задачи (так, например, происходит при использовании технологии OpenMP). Ввод/вывод с каждой нити приложения осуществляется так же, как в однопоточном приложении, и не вызывает больших проблем.

Разработка любого приложения для вычислителей с гибридной архитектурой, напротив, требует детального знания об устройстве узлов. При этом только часть информации можно получить во время исполнения кода. В этой ситуации требуется изначально ориентироваться на конкретную аппаратную платформу. Время доступа к различным типам памяти гибридного узла сильно отличается. Количество одновременно исполняемого кода и обрабатываемого массива данных

сильно ограничено. Ввод/вывод данных возможен напрямую только из тех нитей приложения, которые выполняются на CPU. Следует также отметить, что время инициализации GPU достаточно велико (порядка нескольких секунд), поэтому инициализация этих устройств производится один раз в начале цикла вычислений. Эти и другие особенности гибридных вычислительных систем приходится учитывать при разработке конкретных приложений.

В данной работе авторы конкретизировали аппаратно-программную платформу. Здесь речь пойдёт об архитектуре CPU-GPU на базе центральных процессоров Intel (или AMD) и графических ускорителей NVIDIA. С одной стороны, данное сужение класса вычислителей не является фатальным, поскольку они составляют более 90 % используемого парка вычислительных систем и доступны практически любому пользователю. С другой стороны, это сужение позволяет большую часть анализа аппаратуры проводить во время исполнения программы и разработать общие подходы к разработке приложений. Также было решено ограничиться рассмотрением систем, управляющихся 64-битными ОС Windows и Linux. Основная цель предлагаемого исследования – сформировать подход к разработке параллельных приложений для такого рода систем, который удовлетворял бы следующим требованиям:

1) разработанный параллельный код позволяет выполнять расчёты задачи на гибридных вычислительных системах из указанных выше трёх классов: ПК, БК, СКВС;
2) разработанный параллельный код легко переносится с одной аппаратно-программной платформы на другую;
3) для разработки параллельного кода потребуется только свободно распространяемое или условно свободно распространяемое программное обеспечение (ПО).

Гибридная параллельная платформа. Предлагаемая концепция параллельных вычислений на гибридных вычислительных системах указанного выше класса была выработана на основе анализа литературы и собственного опыта авторов по организации параллельных вычислений на гибридных ВС. Базой для реализации вычислений на GPU послужили работы [1–3]. Вопросы кросс-платформенной реализации были рассмотрены с использованием [4–5]. Для изучения общих вопросов параллельного программирования имеется уже очень широкий пласт литературы (см., например, [6–9]).

Сформулируем общие предложения, которые могут быть положены в основу концепции гибридных параллельных вычислений.

1. В основе любого параллельного решателя, реализующего прикладную задачу, должно находиться простое MPI-приложение, ориентированное на распределённую модель вычислений. Распределённая модель вычислений выбрана в связи с тем, что требуется создавать приложения, ориентированные на произвольное сколь угодно большое количество вычислителей. К тому же поддержка MPI-программ реализована на большинстве аппаратно-программных платформ, и перенос таких программ с одной платформы на другую не вызывает особых затруднений.

2. Основные языки программирования, которые обычно используются для реализации MPI-приложений: C, C++, Fortran. Естественно остановиться на этом множестве и в данном случае. Однако версии самих языков требуют уточнения. В качестве единой свободно распространяемой версии компиляторов с языков C, C++, Fortran в случае 64-битной адресации можно было бы использовать GNU C/C++/Fortran. Однако это решение подойдёт только для ОС Linux. Дело в том, что для 64-битной версии ОС Windows реализован только компилятор GNU C/C++ (в пакете MinGW64) и компилятора с языка Fortran в нём нет. Другой проблемой является то, что компилятор C/C++ для GPU (пакет CUDA Toolkit), сво-

бодно распространяемый компанией NVIDIA, под ОС Linux является надстройкой над GNU C/C++. В ОС Windows пакет компиляторов CUDA Toolkit базируется на полукоммерческой среде Microsoft Visual Studio и требует её наличия на компьютере. Поэтому в ОС Windows приходится использовать Microsoft Visual C/C++, а в ОС Linux можно использовать свободно распространяемый компилятор GNU C/C++. Свободно распространяемый 64-битный компилятор с языка Fortran для ОС Windows мы не найдём, а в ОС Linux это GNU Fortran. Заметим, что полное кросс-платформное решение имеется у Intel для трёх языков C/C++/Fortran, однако оно будет платным.

3. Выбор реализации MPI зависит от двух параметров. Во-первых, от типа сетевого интерфейса между узлами; во-вторых, от типа ОС. В первом случае имеются следующие альтернативы: в качестве сетевого решения может использоваться Ethernet, InfiniBand, Myrinet, SCI и другая аппаратура связи. Поскольку первые два типа интерфейсов захватили большую часть рынка коммуникаций, то можно ориентироваться только на них. При этом среди различных реализаций MPI выделяются две группы: реализации, поддерживающие только один тип коммуникаций, и реализации, поддерживающие сразу несколько сетевых интерфейсов. В первую группу можно поместить MPICH, MPICH2 и LAMMPI (поддерживают коммуникации в сетях Ethernet), MVAPICH и MVAPICH2 (поддерживают коммуникации в сетях InfiniBand). Во вторую группу попал OpenMPI. Он поддерживает оба способа коммункаций – Ethernet и InfiniBand. Если далее рассмотреть фактор, связанный с типом ОС, то выяснится, что практически все реализации MPI имеют версии для ОС Linux, и только некоторые из них имеют версии для Windows. С точки зрения создания свободно распространяемого ПО следует использовать MPICH, который имеет реализации для ОС Windows и Linux. Однако это решение подойдёт только для сетей Ethernet. Условно бесплатной альтернативой MPICH под ОС Windows является MS MPI, входящий в пакет Microsoft HPC Pack 2008. В случае интерфейса InfiniBand для ОС Linux имеем 2 решения: MVAPICH и OpenMP. Коммерческий вариант для обеих ОС и обоих типов сетевого интерфейса имеется только у Intel. Важно также отметить, что с точки зрения содержания MPI-программы конкретная реализация MPI не влияет на итоговый текст создаваемого параллельного приложения.

4. Поскольку каждый узел рассматриваемого семейства вычислительных систем имеет многоядерную архитектуру, то естественно использовать дальнейшее распараллеливание прикладной задачи по ядрам CPU на общей памяти внутри узла. Реализацию этой идеи можно производить с помощью технологий OpenMP или PThreads. Оба эти средства имеют свои преимущества и недостатки. Однако общая их особенность состоит в том, что они встроены в язык программирования высокого уровня. При этом функции библиотеки PThreads являются неотъемлемой частью стандарта языка ANSI C/C++, а интерфейс OpenMP включён как в ANSI C/C++, так и в Fortran. Комбинируя оба подхода, можно создавать весьма гибкие прикладные параллельные программы. Версии интерфейсов OpenMP и PThreads зависят от версий компиляторов и runtime-библиотек ОС. Текст параллельной программы в этом случае зависит только от сделанного выбора: OpenMP или PThreads.

5. Во многих случаях производительность параллельного приложения можно существенно увеличить, если наиболее затратные части кода выполнять на GPU. Для реализации такой аппаратной возможности предлагается использовать технологию CUDA. Технология CUDA состоит из двух основных частей. Во-первых, это специальный драйвер устройства (в дальнейшем CUDA driver), обеспечивающий управление GPU со стороны ОС и других приложений (версии драйвера поставляются для большинства аппаратно-программных платформ). Во-вторых, это компилятор с языка C/C++, входящий в CUDA Toolkit, позволяющий писать либо отдель-

ные программы для GPU, либо фрагменты кода для GPU, встраиваемые в код для CPU. Использование технологии CUDA позволяет задействовать все ресурсы вычислительного узла, в том числе ресурсы всех CPU и всех GPU одновременно.

6. Реализация технологии CUDA накладывает определённые требования на разработку приложения. Во-первых, компилятор CUDA C/C++ является надстройкой над Microsoft Visual C/C++ в ОС Windows и надстройкой над GNU C/C++ в ОС Linux. Поэтому предустановка этих компиляторов на вычислителях становится необходимой. Во-вторых, использование нескольких GPU на узле требует создания соответствующего количества нитей CPU, управляющих отдельными GPU. Тем самым использование технлогии CUDA тесно связано с технологиями OpenMP или PThreads. В-третьих, передача данных между отдельными GPU в рамках одного узла может осуществляться либо через основную память посредством CPU, либо непосредственно по каналам прямого доступа шины PCI Express. Реализация первого механизма не требует привлечения дополнительных средств программирования. Однако в этом случае при выполнении приложений имеется существенное снижение скорости передачи данных между различными GPU. Одна из реализаций второго механизма состоит в использовании библиотеки Shmem, с помощью которой можно организовать единое адресное пространство памяти всех GPU, в том числе располагающихся за пределами узла (в других узлах). В-четвёртых, при разработке MPI-приложений, использующих вычисления на GPU, компиляция функций для GPU и так называемых "обёрток" производится компилятором CUDA C/C++, а сборка приложения в целом производится средствами пакета MPI для CPU.

7. При реализации параллельной программы решения сеточной задачи на гибридном вычислителе следует прежде всего определить модель размещения и обработки основных массивов данных: 1) только в памяти CPU; 2) только в памяти GPU; 3) смешанный способ хранения. В первом случае подразумевается, что GPU используется только как сопроцессор, обрабатывающий небольшие, но вычислительно ёмкие блоки данных. Во втором случае подразумевается, что вся задача укладывается в память одного или нескольких GPU, а CPU используется только для загрузки GPU и операций ввода/вывода. В третьем случае подразумевается, что информация настолько велика по объёму, что суммарной памяти GPU недостаточно для размещения всех обрабатываемых данных.

8. Выбор модели размещения и обработки данных определяет два важных следствия. Первое – это тип основного интерфейса передачи данных между вычислителями (в случае 1) – это передача данных по сети, в случае 2) – это передача данных по шине PCI Express, в случе 3) – смешанная схема обменов). Второе – это тип используемого средства программирования коммуникаций (в случае 1) – это MPI, в случае 2) – Shmem, в случае 3) – использование обоих средств).

9. Применительно к задачам механики сплошной среды, решаемым параллельными сеточными методами, можно добавить ещё несколько специфических моментов. Отметим основной из них.

Эффективное решение задач МСС на сетках различного типа существенно зависит от двух факторов: 1) является ли сетка динамической (меняет свою структуру и объём в процессе вычислений) или статической (не меняется в процессе вычислений); 2) является ли сетка структурированной (имеет фиксированный шаблон связей её элементов) или неструктурированной (каждый элемент сетки может иметь индивидуальный шаблон связей с другими элементами).

Если сетка динамическая, то к известным проблемам балансировки вычислительной загрузки узлов добавляется ещё и реализация перестроений сетки во время вычислений, что влечёт за собой изменение всей вычислительной модели.

На сегодняшний день гибридные вычислительные системы мало приспособлены к реализации такой вычислительной схемы, и в этом случае лучше оставаться в рамках вычислений на центральных процессорах, которые обладают большей гибкостью архитектуры и большей оперативной памятью.

Если используемая сетка статическая и структурированная (например, декартова или топологически ей эквивалентная), то перенос всех или большинства вычислений на GPU не представляется сложной проблемой. Здесь следует лишь учесть, что применяемый метод геометрического параллелизма на CPU остаётся крупнозернистым как на межузловом уровне, так и внутри узлов. Если же вычисления ведутся на GPU, то внутри узла следует применять модель мелкозернистого параллелизма.

Если используемая сетка статическая и неструктурированная, то при переводе вычислений на GPU следует преобразовать её в локально-структурированный объект. Для этого, например, её предварительно следует упорядочить по размеру и структуре шаблона и разместить близкие по характеристикам элементы сетки в памяти GPU смежным образом. Далее необходимо для каждого GPU сформировать и использовать единый фиксированный шаблон связей, вмещающий любой конкретный шаблон сетки, принадлежащий данному GPU.

Решение проблем динамической балансировки загрузки GPU решается для фиксированного локального шаблона. Сама блочная структура организации вычислений на GPU и аппаратная поддержка балансировки загрузки внутри GPU подсказывают методику общего алгоритма балансировки. В целом можно отметить, что практически любой алгоритм балансировки, имеющий в качестве динамически подбираемых параметров размеры двумерного или трёхмерного блока сетки, будет заведомо лучше, чем его неблочный вариант.

Эти и другие приёмы можно использовать в качестве основы для разработки конкретных параллельных сеточных приложений.

Пример реализации гибридных сеточных вычислений. Для иллюстрации изложенной выше концепции была проведена адаптация известных сеточных подходов к решению задач механики сплошной среды на неструктурированных сетках на гибридном суперкомпьютере с графическими ускорителями. Рассматривалась проблема моделирования вязких и невязких газодинамических течений на неструктурированных гибридных сетках. Дискретная модель расчетной области состояла из элементов четырех фиксированных типов: шестигранник, четырехугольная пирамида, тетраэдр и треугольная призма. Аппроксимация систем уравнений Эйлера и Навье–Стокса проводилась методом конечных объёмов с использованием схемы Роу для вычисления конвективных потоков, а также методов повышенного порядка аппроксимации на гибридных сетках, основанных на полиномиальной реконструкции значений газодинамических переменных внутри сеточных элементов. Целью работы было определить основные проблемы реализации алгоритмов решения рассматриваемого класса задач для GPU в сравнении с многоядерными процессорами общего назначения. Все вычислительные эксперименты проводились на суперкомпьютере К-100 (ИМП им. М.В. Келдыша РАН). Данная СКВС состоит из вычислительных модулей, каждый из которых содержит в себе 2 6-ядерных процессора Intel Xeon X5670 и 3 графических процессора nVidia Tesla C2050. Результаты проведенных исследований показали следующее.

Графические процессоры nVidia Tesla C2050 поддерживают одновременное выполнение большого числа нитей (до 1536 нитей на каждом потоковом мультипроцессоре, которых здесь 14). При этом влияние загрузки мультипроцессора (отношение числа фактически выполняемых на нем нитей к максимально допустимому числу) на эффективность работы программного обеспечения достаточно велико. Приемлемая скорость чтения/записи упорядоченных в глобальной памяти устройства данных достигается в случае загрузки ресурсов на 50 % и более (табл. 1).

В свою очередь, для достижения высокой производительности непосредственно вычислений коэффициент загрузки ресурсов должен быть не ниже 33 % (табл. 2). На практике уровень загрузки мультипроцессора ограничивается размерами его регистровой и разделяемой памяти. В пределе каждой нити доступно лишь 63 32-битовых регистра (выполнение 512 нитей с 33 %-ной загрузкой). При необходимости использования более чем 63 регистров на нить недостающие регистры эмулируются в глобальной памяти устройства, что, естественно, снижает эффективность самих вычислений.

Таблица 1

Зависимость скорости чтения/записи данных от загрузки мультипроцессора

Размер блока нитей	32	64	128	256	512	1024
Загрузка мультипроцессора (%)	16,7	33,3	66,7	100	100	66,7
Скорость чтения/записи данных (ГБ/сек)	57,2	85,6	102	97,5	100,4	97

Таблица 2

Зависимость производительности вычислений от загрузки мульти-процессора (на примере численного интегрирования квадратичного полинома)

Размер блока нитей	32	64	128	256	512	1024
Загрузка мультипроцессора (%)	16,7	33,3	66,7	100	100	66,7
Производительность вычислений (GFlops)	219,4	236,3	236,5	236,7	238	237,3

Для эффективного использования возможностей графического процессора необходимо разделить исходную сложную алгоритмическую задачу на относительно простые этапы. А каждый из таких этапов постараться представить в виде множества независимых подзадач с минимальной ресурсоемкостью и максимальным объемом вычислений на единицу данных. В этом случае, во-первых, увеличивается загрузка мультипроцессора, что положительно влияет на скорость чтения/записи данных. А во-вторых, появляется возможность "скрыть" обращения к памяти одних блоков нитей за вычислениями других блоков (или наоборот), одновременно выполняемых на одном мультипроцессоре.

Оценка эффективности вычислений на CPU и GPU проводилась на примере программных реализаций двух основных вычислительных модулей расчетного алгоритма. Это вычисление коэффициентов полиномиальной реконструкции значений пяти газодинамических переменных для каждого элемента сетки и вычисление конвективных потоков через грани расчетных ячеек. Сравнивалось время работы программ на одном CPU (6 процессорных ядер) и одном GPU. В случае использования CPU распараллеливание вычислений выполнялось в рамках модели общей памяти (интерфейс разработки прикладного программного обеспечения OpenMP). В качестве исходных данных тестовых задач были взяты промежуточные результаты расчета задачи невязкого газодинамического обтекания сферы на тетраэдральной сетке, содержащей порядка 700 тысяч элементов (около 1,3 миллиона граней расчетных ячеек).

Алгоритмически задача определения коэффициентов полиномов для каждой из сеточных ячеек сводится к перемножению постоянной на протяжении всего расчета матрицы A[9][N] на матрицу B[N][5], значения элементов которой зависят от текущих значений газодинамических переменных в ячейках шаблона соответствующего сеточного элемента. Здесь N – размер шаблона. В рассматриваемом случае N принимает значения от 24 до 50. В результате умножения A x B формируется матрица P[9][5] с девятью коэффициентами квадратичного полинома для каждой из пяти газодинамических переменных.

Для распараллеливания вычислений на CPU достаточно установить простую директиву распараллеливания OpenMP над основным вычислительным циклом по элементам сетки, указав статический метод распределения витков цикла между нитями.

При реализации этого же алгоритма на GPU используется уже иной принцип. Назовем элементарным заданием процедуру вычисления коэффициентов полиномов для одного сеточного элемента. В рамках предложенного алгоритма обработка одного элементарного задания выполняется 32 нитями, что соответствует размеру варпа. Поскольку мультипроцессор порождает, планирует выполнение и выполняет нити варпами, то такой подход дает возможность, во-первых, минимизировать дисбаланс загрузки, связанный с переменным размером шаблона, и, во-вторых, исключить необходимость синхронизации нитей при считывании данных (элементы матрицы A) в разделяемую память. Для обеспечения оптимальной загрузки устройства нити группируются в блоки размером 128 нитей (4 варпа). Таким образом, каждый блок нитей обрабатывает 4 элементарных задания, т.е. n-й блок нитей вычисляет коэффициенты полиномов для ячеек с индексами 4n, 4n+1, 4n+2 и 4n+3. Ограничения по регистровой памяти для используемой программной реализации позволяют одновременно запустить на каждом мультипроцессоре 7 блоков нитей, что соответствует уровню загрузки в 58 %.

Времена работы соответствующих программных модулей приведены в табл. 3. Времена вычисления конвективных потоков приведены в табл. 4. При использовании CPU распараллеливание вычислений происходит на основе метода геометрического параллелизма: каждая нить OpenMP обрабатывает отведенную ей подобласть сетки. Обработка подобласти заключается в вычислении потоков функции распределения через грани вошедших в нее ячеек с последующим суммированием результата вычислений.

Таблица 3

Сравнение эффективности вычислений на CPU и GPU на примере задачи вычисления коэффициентов полиномиальной реконструкции

Вычислитель	CPU (6 ядер)	GPU (448 ядер)
Время работы (секунд)	0,145	0,045
Производительность (GFlops)	16,37	52,74

Таблица 4

Сравнение эффективности вычислений на CPU и GPU на примере задачи вычисления конвективных потоков через грани расчетных ячеек

Вычислитель	CPU (6 ядер)	GPU (448 ядер)
Время работы (секунд)	0,059	0,0168
Производительность (GFlops)	10,91	38,3

При использовании GPU любой вариант алгоритма, основанный на проходе по граням контрольных объемов, будет иметь минимальную эффективность. Объем исходных данных для вычисления конвективного потока через грань расчетной ячейки и число внутренних переменных в программной реализации схемы Роу существенно превосходят ограничение в 63 регистра на нить. При этом процедура вычисления потока по Роу не распадается на подзадачи с возможностью параллельного выполнения на SIMD-архитектуре. Поэтому с целью оптимизации использования ресурсов GPU исходная задача была разделена на 3 алгоритмически относительно простых этапа: 1) вычисление значений газодинамических переменных слева и справа от центров граней; 2) вычисление потоков через грани расчетных ячеек; 3) суммирование потоков через грани в ячейки сетки. Узким местом в данном случае остался второй этап, где избежать эмуляции регистров в памяти

устройства не представляется возможным. В свою очередь задачи, решаемые на первом и третьем этапах алгоритма, достаточно просты, что обеспечивает высокую загрузку мультипроцессора.

Основываясь на представленных в табл. 3 и 4 результатах, можно сказать, что использование GPU позволяет существенно (более чем в 3 раза) снизить время вычислений. К положительным моментам в использовании GPU следует также отнести слабую зависимость производительности вычислений от нумерации элементов сетки, что является большой проблемой для программных реализаций рассматриваемых вычислительных алгоритмов на CPU. С другой стороны, с точки зрения энергопотребления величина ускорения в 3 раза лишь немного выше минимально допустимого для сравниваемых устройств уровня в 2.5 раза. Кроме того, можно с большой вероятностью прогнозировать, что в случае повышения алгоритмической или вычислительной сложности расчетного алгоритма (например, реализации процедуры интегрирования потока через грань с использованием метода повышенного порядка) преимущество GPU будет постепенно снижаться.

Заключение. В заключение отметим, что представленная концепция гибридных параллельных вычислений является лишь ориентиром для разработки конкретных приложений в области решения задач механики сплошной среды сеточными методами. Однако её использование на практике может существенно сократить цикл разработки параллельных программ для гибридных вычислительных систем.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Сандерс Дж., Кэндрот Э.* Технология CUDA в примерах: введение в программирование графических процессоров. – М.: ДМК Пресс, 2011. – 232 с.
2. NVIDIA CUDA C Programming Guide. Version 4.0. Santa Clara (CA, USA): NVIDIA Corporation, 2011. – 187 p.
3. *Боресков А.В., Харламов А.А.* Основы работы с технологией CUDA. – М.: ДМК Пресс, 2010. – 232 с.
4. *Logan S.* Cross-platform development in C++ : building Mac OS X, Linux, and Windows applications. Crawfordsville (Indiana, USA): RR Donnelly, 2007. – 575 p.
5. *Стефенс Д.Р., Диггинс К., Турканис Д., Когсуэлл Д.* C++. Сборник рецептов. – М.: КУДИЦ-ПРЕСС, 2007. – 624 с.
6. *Ортега Дж.* Введение в параллельные и векторные методы решения линейных систем. – М.: Мир, 1971. – 367 с.
7. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 608 с.
8. *Dongarra J., Foster I., Fox J. et al.* Sourcebook of Parallel Computing. San Francisco (CA, USA): Elsevier Science, 2003. – 852 p.
9. *Гергель В.П.* Теория и практика параллельных вычислений. СПб.: "Интернет-университет информационных технологий – ИНТУИТ.ру", "БИНОМ. Лаборатория знаний", 2007. – 424 с.

Статью рекомендовал к опубликованию д.ф.-м.н., профессор М.В. Якобовский.

Поляков Сергей Владимирович – Институт прикладной математики им. М.В. Келдыша РАН; e-mail: polyakov@imamod.ru; 125047, г. Москва, Миусская пл., 4; тел.: 84999730385; отдел № 16; зав. сектором; д.ф.-м.н.; с.н.с.

Карамзин Юрий Николаевич – e-mail: karamzin@imamod.ru; отдел № 16; г.н.с.; д.ф.-м.н.; профессор.

Кудряшова Татьяна Алексеевна – e-mail: kudryashova@imamod.ru; тел.: 84992507910; отдел № 16; к.ф.-м.н.; с.н.с.

Суков Сергей Александрович – e-mail: pm18@imamod.ru; тел.: 84992507823; отдел № 16; к.ф.-м.н.; с.н.с.; магистр.

Косолапов Олег Александрович – Info Industries Group; e-mail: firimar@mail.ru; 119017, г. Москва, Малый Толмачевский пер., 8/11, стр. 3; отдел разработки ПО; программист.

Polyakov Sergey Vladimirovich – Keldysh Institute for Applied Mathematics, Russian Academy of Sciences; e-mail: polyakov@imamod.ru; 4, Miusskaya square, Moscow,125047, Russia; phone: +74999730385; department № 16; head of laboratory; dr. of phis.-math. sc.; senior scientist.

Karamzin Yuri Nikolaevich – e-mail: karamzin@imamod.ru; department № 16; main research associate; dr. of phis.-math. sc.; professor.

Kudryashova Tatiana Alekseevna – e-mail: kudryashova@imamod.ru; phone: +74992507910; department № 16; cand. of phis.-math. sc.; senior scientist.

Soukov Sergey Aleksandrovich – e-mail: pm18@imamod.ru; phone: +74992507823; department № 16; cand. of phis.-math. sc.; senior scientist; master of science.

Kosolapov Oleg Aleksandrovich – Info Industries Group; e-mail: firimar@mail.ru; 8/11, Malyi Tolmachevskiy pereulok, building 3, Moscow, 119017, Russia; software department; programmer.