Лекция 8. Концепция гибридной параллельной вычислительной платформы

1. Предварительные замечания

Особенностью современного этапа развития прикладных параллельных программных средств математического моделирования является переход к гибридным решениям. Причиной этому послужило широкое распространение гибридных компьютерных архитектур и развитие программных сред и языков программирования, ориентированных на специальные вычислители, среди которых в первую очередь следует отметить графические ускорители.

В настоящей работе сформулирована концепция гибридных параллельных вычислений, которую можно положить в основу программной части гибридной суперкомпьютерной платформы (технологии) – ГСКП (ГСКТ). Среди множества вопросов, примыкающих к этой проблеме, были рассмотрены два: алгоритмы решения задач и особенности их реализации на доступных широкому кругу пользователей вычислительных системах.

2. Особенности современных аппаратно-программных средств

Для достижения поставленных целей обратимся к обзору аппаратных и программных средств, имеющихся в наличии у большинства исследователей. В качестве аппаратных решений будем различать три класса вычислительных систем (ВС):

1) настольные или мобильные персональные компьютеры пользователей (ПКП),

2) бюджетные кластеры лабораторий (БКЛ) 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.

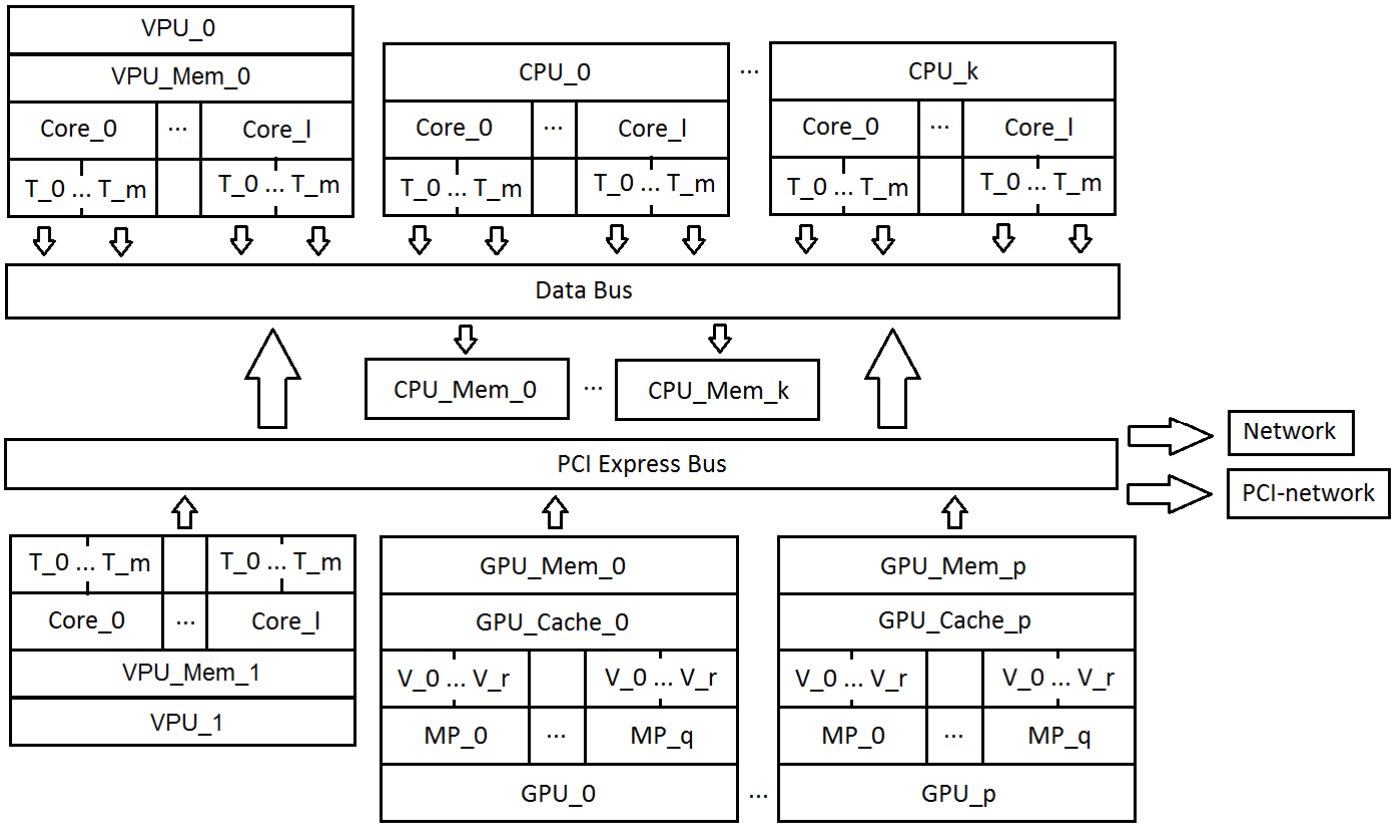

Рис.1. Гибридная архитектура современного вычислительного узла кластера или персонального компьютера.

На Рис. 1 показана гибридная архитектура современного вычислительного узла, характерная практически для любой современной компьютерной системы. Здесь предполагается, что узел имеет k центральных (ЦПУ/CPU) и p графических (ГПУ/GPU) процессоров, а также k банков общей оперативной памяти, тесно связанных каждый с одним ЦПУ, но доступных всем ЦПУ.

Каждый ЦПУ имеет l ядер, поддерживающих по m конвеерных устройств обработки данных, и доступ к любому банку оперативной памяти через общую шину данных.

Каждый ГПУ 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 ГПУ (в том числе внешних) по шине PCI Express (NVlink в новых ВС), а также к оперативной памяти своего узла. В отличие от ЦПУ 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, в том числе сдвоенный или счетверенный) получаем бюджетный кластер терраflопного диапазона производительности. Если для объединения применяется одна или несколько высокоскоростных сетей (10/40/100-Gigabit Ethernet, 56/100/200-Gigabit InfiniBand, 50/100-Gigabit Omni Path и т.д.), а число узлов измеряется сотнями и более, получаем суперкомпьютерную систему вплоть до экзофлопсного уровня производительности.

С точки зрения управляющей программной среды гибридных вычислительных систем (ГВС) ситуация складывается следующим образом. В настольных и мобильных аппаратных решениях в основном используется 64х-битная операционная система Windows (далее ОС Windows), а также Mac OS X. Существенно реже встречается операционная система UNIX в версии Linux (далее ОС Linux). 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 вычислительных установок управляется ОС Windows или ОС Linux.

3. Постановка задачи

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 также, 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 ЦПУ. Следует также отметить, что время инициализации ГПУ 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.

Для упрощения дальнейшего анализа была конкретизирована аппаратно-программная платформа. Ее основу составляет архитектура ЦПУ-ГПУ на базе центральных процессоров Intel (или AMD) и графических ускорителей NVIDIA (или AMD). Данное сужение класса вычислителей не является фатальным, поскольку они составляют более 90% используемого парка вычислительных систем и доступны практически любому пользователю. Однако это сужение позволяет большую часть анализа аппаратуры проводить во время исполнения программы и разработать общие подходы к разработке приложений.

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 ГВС из указанных выше трёх классов: ПКП, БКЛ, 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).

4. Гибридная параллельная вычислительная платформа

Предлагаемая концепция параллельных вычислений на ГВС указанного класса выработана на основе анализа литературы и опыта авторов в разработке кросс-платформных приложений.

Сформулируем общие предложения концепции гибридных параллельных вычислений.

П1. 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.

П2. 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 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 свободно распространяемый компилятор C/C++ для GPU (пакеты CUDA Toolkit или AMD ROCm) в ОС Linux является надстройкой над GNU C/C++. В ОС Windows пакет компиляторов CUDA Toolkit (AMD ROCm) 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 отсутствует, в ОС Linux это GNU Fortran. Заметим, что полное кроссплатформное решение имеется у Intel и IBM для трёх языков C/C++/Fortran, однако оно будет платным.

П3. Выбор реализации MPI зависит от двух параметров. Во-первых, от типа сетевого интерфейса между узлами МВС. Во-вторых, от типа операционной системы. В первом случае имеются следующие альтернативы: в качестве сетевого решения может использоваться Ethernet, InfiniBand, Myrinet, SCI, PCI Express, NVLINK, OMNI Path 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 (***гомогенная реализация***), и реализации, поддерживающие сразу несколько сетевых интерфейсов (***гетерогенная реализация***). В первую группу можно поместить MPICH, MPICH2 и LAMMPI (поддерживают коммуникации в сетях Ethernet), MVAPICH и MVAPICH2 (поддерживают коммуникации в сетях InfiniBand). Также сюда попадают реализации MPI от Intel, IBM, NVidia, Melanox. Во вторую группу попал OpenMPI. Он поддерживает множество способов коммуникаций, в том числе Ethernet и InfiniBand, а также гетерогенность.

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 (версии 2008/2012/2016/2019). В случае интерфейса InfiniBand для ОС Linux имеем 2 решения: MVAPICH и OpenMPI. Коммерческий вариант для обеих ОС и обоих типов сетевого интерфейса имеется только у Intel и IBM. Важно также отметить, что с точки зрения содержания MPI-программы ***конкретная реализация MPI не влияет на итоговый текст создаваемого параллельного приложения***.

П4. Поскольку каждый узел рассматриваемого семейства вычислительных систем имеет многоядерную архитектуру, то естественно использовать дальнейшее распараллеливание прикладной задачи по ядрам (нитям ядер – трэдам) 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. В частности, библиотека процессов PThreads, является неотъемлемой частью ядра ОС Linux. При использовании ОС Windows имеется её аналог (условно назовём его WThreads), который отличается по интерфейсу, но не по функциональным возможностям. В итоге, текст параллельной программы в этом случае зависит только от сделанного выбора: OpenMP или PThreads.

П5. Во многих случаях производительность параллельного приложения можно существенно увеличить, если наиболее затратные части кода выполнять на ГПУ. Для реализации такой аппаратной возможности предлагается использовать технологию CUDA от NVidia или ROCm от AMD. Остановимся на первом варианте. Технология CUDA состоит из двух основных частей. Во-первых, это специальный драйвер устройства (в дальнейшем CUDA driver), обеспечивающий управление ГПУ 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 отдельные программы для GPU, либо фрагменты кода для ГПУ, встраиваемые в код для ЦПУ. Использование технологии CUDA позволяет задействовать все ресурсы вычислительного узла, в том числе ресурсы всех ЦПУ и всех ГПУ одновременно.

Альтернативой технологии CUDA является технология OpenCL, позволяющая разрабатывать многопоточные приложения для вычислителей с различной архитектурой, в том числе, отличной от ЦПУ и ГПУ. В частности, в рамках этой технологии могут быть созданы программы для вычислений на графических картах компании AMD (ATI Radeon) и других производителей. Однако эта технология вцелом не является пока универсальной (конкурирующие версии производятся IBM, Intel, NVidia, AMD). В этой ситуации OpenCL, разработанный сторонними производителями, плохо реализуется на графических процессорах компании NVidia, которые наиболее распространены во всём мире. По этим причинам данная технология пока не рекомендуется к использованию в рамках создаваемой Гибридной параллельной вычислительной платформы.

П6. 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 ГПУ на узле требует создания соответствующего количества нитей ЦПУ, управляющих отдельными ГПУ. Тем самым использование технологии CUDA тесно связано с технологиями OpenMP или PThreads.

В-третьих, передача данных между отдельными ГПУ в рамках одного узла может осуществляться либо через основную память посредством ЦПУ, либо непосредственно по каналам прямого доступа шины PCI Express (NVlink). Реализация первого механизма не требует привлечения дополнительных средств программирования. Однако в этом случае имеется существенное снижение скорости передачи данных между различными ГПУ. Одна из реализаций второго механизма состоит в использовании библиотеки Shmem, с помощью которой можно организовать единое адресное пространство памяти всех ГПУ, в том числе располагающимися за пределами узла (в других узлах). В последних версиях CUDA Toolkit реализована более эффективная модель второго механизма обмена – GPUDirect.

В-четвёртых, при разработке MPI-приложений, использующих вычисления на ГПУ, компиляция функций для ГПУ и так называемых "обёрток" производится компилятором CUDA C/C++ (nvcc), а сборка приложения в целом производится средствами пакета MPI для ЦПУ. В последних версиях CUDA Toolkit всю сборку можно осуществить одним и тем же компилятором nvcc в связи с тем, что появилась совместная реализация MPI от Melanox и NVidia.

П7. При реализации параллельной программы решения какой-либо задачи на гибридном вычислителе следует, прежде всего, определить модель размещения и обработки основных массивов данных: 1) только в памяти ЦПУ; 2) только в памяти ГПУ; 3) смешанный способ хранения.

В первом случае подразумевается, что ГПУ используется только как сопроцессор, обрабатывающий небольшие но вычислительно ёмкие блоки данных.

Во втором случае подразумевается, что вся задача укладывается в память одного или нескольких ГПУ, а ЦПУ используется только для загрузки ГПУ и реализации операций ввода-вывода.

В третьем случае подразумевается, что задача настолько велика по объёму, что суммарной памяти ГПУ недостаточно для размещения всех обрабатываемых данных.

П8. Выбор модели размещения и обработки данных определяет два важных параметра параллельной программы.

Первый параметр – тип основного интерфейса передачи данных между вычислителями. В случае 1) – передача данных по сети, в случае 2) – передача данных по шине PCI Express (NVlink), в случае 3) – смешанная схема обменов.

Второй параметр – тип используемого средства программирования коммуникаций. В случае 1) это MPI, в случае 2) – Shmem или CUDA GPU Direct, в случае 3) – использование обоих средств.

П9. Итоговая реализация гибридного параллельного приложения выполняется с помощью одной из следующих параллельных технологий:

1) MPI (простое MPI-приложение, использующее только ЦПУ и MPI-коммуникации как между узлами, так и внутри каждого узла),

2) MPI+OpenMP (MPI-приложение, использующее только ЦПУ и MPI-коммуникации между узлами, а также распараллеливание на общей памяти внутри узлов с помощью технологии OpenMP),

3) MPI+PThreads (MPI-приложение, использующее только ЦПУ и MPI-коммуникации между узлами, а также распараллеливание на общей памяти внутри узлов с помощью технологии PThreads),

4) MPI+OpenMP+CUDA (MPI-приложение, использующее ЦПУ и ГПУ, MPI-коммуникации между узлами, распараллеливание на общей памяти ЦПУ внутри узлов с помощью технологии OpenMP и распараллеливание внутри ГПУ с помощью технологии CUDA),

5) MPI+PThreads+CUDA (MPI-приложение, использующее ЦПУ и ГПУ, MPI-коммуникации между узлами, распараллеливание на общей памяти ЦПУ внутри узлов с помощью технологии PThreads и распараллеливание внутри ГПУ с помощью технологии CUDA),

6) MPI+OpenMP+CUDA+Shmem или MPI+OpenMP+CUDA+GPUDirect (MPI-приложение, использующее ЦПУ и ГПУ, MPI-коммуникации между узлами, распараллеливание на общей памяти ЦПУ внутри узлов с помощью технологии OpenMP и распараллеливание внутри ГПУ с помощью технологии CUDA, коммуникации между ГПУ различных узлов с помощью функций библиотеки Shmem),

7) MPI+PThreads+CUDA+Shmem или MPI+PThreads+CUDA+GPUDirect (MPI-приложение, использующее ЦПУ и ГПУ, MPI-коммуникации между узлами, распараллеливание на общей памяти ЦПУ внутри узлов с помощью технологии OpenMP и распараллеливание внутри ГПУ с помощью технологии CUDA, коммуникации между ГПУ различных узлов с помощью функций библиотеки Shmem).

8) MPI+CUDA+GPUDirect или CUDA+GPUDirect – современные технологии.

Замечание 1. Вместо технологии CUDA от NVidia во многих случаях уже можно с успехом использовать технологию ROCm от AMD.

Замечание 2. Наряду с векторными операциями ЦПУ и ГПУ возможно использовать режим вычислений TensorFlow. Он аппаратно поддерживается в разных вариантах на платформах Intel, IBM, Nvidia, AMD. TensorFlow – открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронных сетей. Библиотека TensorFlow также часто применяется для решения прикладных задач оптимизации и распознавания образов.

Литература

1. Message Passing Interface (MPI) Forum Home Page. http://www.mpi-forum.org/
2. OpenMP Home Page. http://openmp.org/wp/
3. NVidia CUDA Toolkit Home Page. https://developer.nvidia.com/cuda-toolkit
4. AMD ROCm Home Page. https://rocm.docs.amd.com/en/latest/
5. Боресков А.В., Харламов А.А., Марковский Н.Д. и др. Параллельные вычисления на GPU. Архитектура и программная модель CUDA. – М., Изд-во Московского ун-та, 2012. – 336 с.
6. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров. М.: ДМК Пресс, 2011. - 232 с.
7. NVIDIA CUDA C Programming Guide. Version 4.0. Santa Clara (CA, USA): NVIDIA Corporation, 2011. - 187 p.
8. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. М.: ДМК Пресс, 2010. - 232 с.

9. Logan S. Cross-platform development in C++ : building Mac OS X, Linux, and Windows applications. Crawfordsville (Indiana, USA): RR Donnelly, 2007. - 575 p.
10. Стефенс Д.Р., Диггинс К., Турканис Д., Когсуэлл Д. C++. Сборник рецептов. М.: КУДИЦ-ПРЕСС, 2007. - 624 с.
11. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем. М.: Мир, 1971. - 367 с.
12. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002. - 608 с.
13. Корнеев В.Д. Параллельное программирование в MPI. - Москва-Ижевск, Институт компьютерных исследований, 2003.
14. Dongarra J., Foster I., Fox J., et al. Sourcebook of Parallel Computing. San Francisco (CA, USA): Elsevier Science, 2003. - 852 p.
15. Гергель В.П. Теория и практика параллельных вычислений. СПб.: "Интернет-университет информационных технологий - ИНТУИТ.ру", "БИНОМ. Лаборатория знаний", 2007. - 424 с.