УДК 004.428.4

БИБЛИОТЕКА ДЛЯ ДИНАМИЧЕСКОЙ АДАПТАЦИИ ПРОГРАММ К ГЕТЕРОГЕННЫМ АРХИТЕКТУРАМ ВИДА CPU + GPU

С.А. Гризан, М.А. Кривов

Научный руководитель – профессор Легалов А.И.

Сибирский федеральный университет

За последние два года архитектура высокопроизводительных вычислительных систем претерпела ряд существенный изменений. Появившаяся в 2007 году технология NVidia CUDA позволила задействовать колоссальные вычислительные мощности существующих графических ускорителей, сделав возможным создание «домашних» суперкомпьтеров. Действительно, используя аппаратные компоненты, выпущенные в 2010 году, можно собственноручно собрать систему с производительностью свыше 6 TFlops, в то время как пиковая производительность суперкомпьютера «СКИФ-Аврора», установленного в Южно-Уральском Государственном Университете также в 2010 году, равна 24 TFlops.

Выпуск специализированных графических ускорителей серии NVidia Tesla, предназначающихся исключительно для вычислений и обладающих повышенной надёжностью, позволил оснащать узлы классических кластеров данными ускорителями, в результате чего широкое распространение также стали получать так называемые гетерогенные суперкомпьютеры. К примеру, в ноябрьской редакции списка Топ500 три из пяти самых производительных суперкомпьютеров мира оснащены графическими ускорителями.

Если проследить за дальнейшими планами производителей микропроцессоров, то легко заметить тенденцию к повышению значимости графических сопроцессоров и в будущем. В новой архитектуре процессоров AMD Fusion графическое ядро уже находится на одном кристалле вместе с вычислительными ядрами, используя общую память. Решение под кодовым названием Knight Corner от компании Intel позволит устанавливать платы с 50 дополнительными x86-совместимыми ядрами. Таким образом, если в настоящее время гетерогенность является «приятным» дополнением, позволяющим после некоторой оптимизации получить дополнительное ускорение, то через 2-3 года игнорирование возможностей сопроцессоров будет приводить к использованию долей процента от реально достижимой производительности.

Решаемые проблемы

При реализации алгоритмов для систем с гетерогенными архитектурами с использованием современных распространённых технологий перед программистом встаёт ряд проблем, связанных с неоднородностью вычислителей. Одной из основных проблем является необходимость выбора оптимального вычислителя для каждой части

задачи. Другими словами, без знания специфики архитектуры не всегда понятно, какую часть задачи на каком процессоре лучше считать. Согласно идеологии NVidia, на графические ускорители нужно переносить всё,что удаётся распараллелить под модель SIMT, оставляя центральному процессору только последовательные части программы. Однако, при решении ряда задач, оптимально подходящих для видеокарт, центральный процессор класса Intel Xeon может показать схожую производительность. Соответственно, если же алгоритм «плохо» отображается на архитектуру видеокарт, то их использование может лишь замедлить программу. Причём об этом факте станет известно только после портирования программы на графические ускорители.

При использовании стандарта OpenCL, данная проблема становится менее явной — любое OpenCL-ядро можно выполнять как на графическом, так и центральном процессоре. Однако всё равно это не позволит достичь максимальной производительности, так как часть вычислителей будет простаивать. Очевидным решением является совместное использование графического и центрального процессоров, но тогда встаёт проблема распределения вычислительной нагрузки между ними. При портировании теста Linpack на видеокарты Tesla C1060 оптимальным оказалось выполнять 68% вычислений на графических ускорителях, а оставшиеся 32% - на центральных процессорах. Понятно, что данная оценка верна только для связки «конкретная система + конкретная реализация алгоритма», и при малейших аппаратных или программных изменениях потребуется проведение дополнительных тестов и измерений, которые могут оказаться достаточно сложными.

Другой проблемой программирования под гетерогенные системы является подбор «магических» констант при отображении алгоритма на архитектуру графических ускорителей. Только в технологии NVidia CUDA подобными параметрами являются размер блока и топология нитей, объём выделяемой памяти на каждом уровне и порог для развёртывания циклов. Правильно подобранные подобные параметры могут существенно повысить работу программы, при этом для них не существует универсальных значений — всё зависит как от специфики ускорителя, так и обрабатываемых данных.

Описание библиотеки

Ниже приведено краткое описание библиотеки ttgLib, проводящей динамическую адаптацию программ для гетерогенных архитектур и решающей названные проблемы.

Основная идея библиотеки заключается в использовании динамических параметров, непосредственное значение которых будет определяться во время работы программы и, более того, будет изменяться в зависимости от типа вычислений и возможностей используемой аппаратной платформы. Для программиста это означает, что больше не требуется знание специфичных особенностей архитектуры и конкретных параметров используемых вычислителей. Если требуется определить значение какого-либо параметра, то вместо него можно указать специальную конструкцию-«заглушку» и, при необходимости, определить начальное значение.

Для подбора оптимальных значений определённых подобным образом параметров

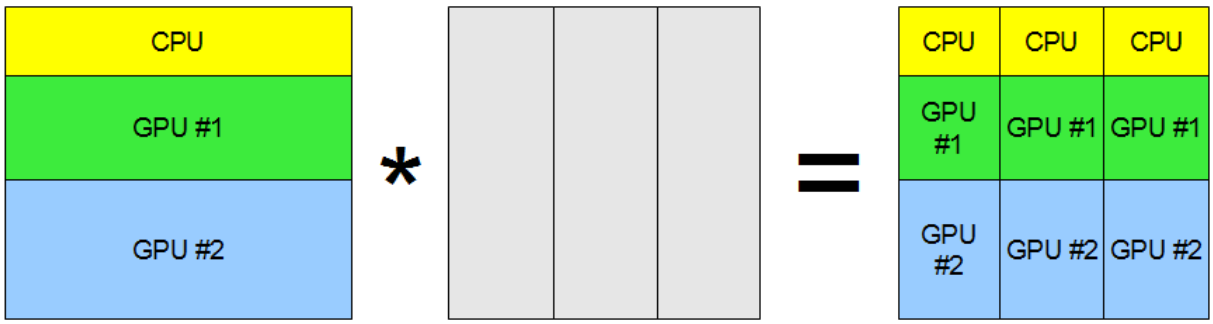
возможно два режима работы программы. В первом из них перед выполнением основных вычислений производится «обучение» на небольших входных данных. Это позволяет специальному менеджеру параметров попытаться повысить скорость работы, подбирая оптимальный вектор значений. Второй режим ориентирован на итеративные программы, в которых одно и тоже вычислительное ядро выполняется много раз. Таким образом, каждую итерацию или запуск в режиме обучения можно рассматривать как один шаг оптимизационного метода, используемого для минимизации функции времени работы, зависящей от вектора параметров. Отметим, что в настоящее время используется модификация метода покоординатного спуска, но в дальнейшем также планируется реализация как градиентного метода, так и метода на основе генетического алгоритма.

Для решения проблемы балансировки нагрузки с использованием предлагаемого подхода обрабатываемые данные разбиваются на несколько неравных частей, размеры которых будут динамическими параметрами. После обучения менеджер сможет подобрать оптимальноеразбиение, в результате чего все вычислители будут равномерно загружены. Отметим, что если экстремум функции достигается на границе (что может быть связанно с низкой производительностью какого-либо вычислителя или с особенностями копирования памяти), то соответствующий вычислитель не будет использоваться. Подобная организация программ позволяет создавать решения, которые будут достаточно эффективно работать на различных вычислительных системах и динамически «подстраиваться» под обрабатываемые данные.

Демонстрация работы

Принцип работы данного механизма лучше всего проиллюстрировать на примере перемножения матриц на системе с гетерогенной архитектурой вида CPU + GPU + GPU, в которой используются различные графические ускорители. Подобной системой может оказаться персональный компьютер, оснащённый дискретным графическим ускорителем и встроенным в чипсет или в процессор видеоядром, а также специализированная высокопроизводительная система, на которой была обновлена только часть видеоплат.

В данном случае использовалось разбиение следующего вида:



Размер каждой из полосок был параметром, значение которого варьируется от 0.0 до 1.0. Также на параметры было наложено общее ограничение - сумма значений равна 1.0. Отметим, что вторая матрица хранилась целиком в памяти каждого вычислителя, а от первой и результирующей матрицы копировались только требуемые строки.

Результат работы описанного подхода на системе, предоставленной НИВЦ МГУ и оснащённой четырёхядерным процессором Inte Core Quad Q6600 и картами NVidia Tesla C2050 и NVidia Tesla C1060, приведён в следующей таблице:

Номер итерации	CPU		GPU #1		GPU #2		Общее время, сек
	Время, сек	% разбиения	Время, сек	% разбиения	Время, сек	% разбиения	
1	24,5	33	2,4	33	2,6	33	24,5
2	17	23	2,5	38	2,7	38	17
3	9,6	13	2,6	43	2,9	43	9,6
4	2,2	3	2,67	49	2,93	48	2,93
5	5,93	8	2,62	46	2,9	46	5,93
6	3,71	5	2,64	48	2,92	47	3,71
7	2,96	4	2,65	48	2,92	48	2,96

За 7 итераций производительность программы при перемножении матриц размером 3000 на 3000 элементов повысилась более чем в 8 раз. По сравнению же с реализацией, использующей только одну видеокарту, ускорение составляет 13%, что вызвано спецификой задачи, в которой узким местом оказалось копирование памяти на графические ускорители.

Дальнейшее развитие

Основным направлением работ является адаптация текущей версии библиотеки для кластерных гетерогенных систем. Соответственно, планируется выпуск специальной версии, проводящей независимую адаптацию MPI-программы на каждом конкретном узле и регистрирующей найденные векторы оптимальных значений на узле-мастере с целью их последующего использования как начальных приближений при последующих запусках программы.

Другим направлением является более тесная интеграция с существующими технологиями параллельного программирования, такими как NVidia CUDA и OpenCL. Планируется выпуск специальных интеграционных «обёрток», встраивающих соответствующие механизмы адаптации в соответствующие SDK. С точки зрения программиста это будет означать появление дополнительных параллельных примитивов, которые будут получать вычислительные ядра и обрабатываемый массив данных, и динамически проводить распределение вычислений между графическими и центральными процессорами.