Алгоритмы и модули динамического планирования вычислительной нагрузки в системе фрагментированного программирования LuNA

Dynamic GPU-CPU load distribution algorithms and modules for LuNA fragmented programming system

Беляев Н. А.

Новосибирский государственный технический университет

ИВМиМГ СО РАН

Введение. Для решения больших задач численного моделирования и вычислительной математики активно применяются суперкомпьютеры, состоящие из большого числа узлов, соединенных высокоскоростной сетью. Узел суперкомпьютера, в свою очередь, может состоять из вычислительных устройств, имеющих различные архитектуры. Для построения суперкомпьютеров часто применяются специализированные вычислительные устройства в составе узлов, такие, как ускорители Intel Xeon Phi или графические карты Nvidia. Такие специализированные вычислительные устройства способны повысить производительность суперкомпьютера, однако их применение усложняет разработку параллельных программ, эффективно утилизирующих все доступные ресурсы суперкомпьюетра. Обеспечение эффективного использования параллельной программой всех доступных вычислительных устройств в узле суперкомпьютера является сложной задачей системного параллельного программирования. Частичная или полная автоматизация обеспечения необходимых свойств параллельных программ системами параллельного программирования позволит значительно упростить и ускорить разработку параллельных программ.

На сегодняшний день существуют средства параллельного программирования, обеспечивающие возможность вовлекать в вычисления специализированные вычислительные устройства в составе узлов суперкомпьютера. OpenCL [1] – это открытый фреймворк для создания параллелных

программ, использующих для работы вычислительные устройства, построенные на различных архитектурах. OpenCL включает в себя компилятор встроенного Си-подобного языка и исполнительную систему. Параллельная программа разбивается на множество задач, записываемых на встроенном языке OpenCL. OpenCL представляет собой низкоуровневое средство параллельного программирования. Разработчик параллельных программ программирует помимо непосредственно подпрограмм, выполняемых на специализированных вычислительных устройствах, логику их взаимодействия, передачи входных и выходных данных каждой подпрограммы и задает распределение ресурсов, используя API OpenCL. OpenACC[2] является открытым стандартом, описывающим набор директив компилятора для работы с GPU. Директивы OpenACC синтаксически схожи с аналогичными директивами в OpenMP[3]. OpenACC компилирует код, помеченный директивами в машинный код для GPU и управляет пересылками данных и запуском кода на GPU.

Система фрагментированного программирования LuNA[4] разрабатывается в ИВМиМГ СО РАН. Система фрагментированного программирования LuNA представляет собой систему параллельного программирования, ориентированную на решение больших задач численного моделирования и вычислительной математики. Система LuNA частично абстрагирует разработчика параллельных программ от решения ряда задач системного параллельного программирования. Однако система LuNA позволяет вовлекать в вычисления только CPU в составе вычислительных узлов суперкомпьютера.

Целью работы является разработка алгоритмов динамического планирования вычислительной нагрузки и реализация разработанных алгоритмов в виде модулей для системы фрагментированного программирования LuNA.

Ниже рассматриваются основные принципы системы LuNA и ее архитектура, в разделе 2 рассматривается предложенный алгоритм распределения нагрузки и его реализация в виде модуля для системы LuNA.

1. Система фрагментированного программирования LuNA

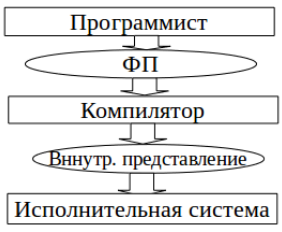

Рис. 1. Устройство системы LuNA.

Система LuNA состоит из компилятора языка LuNA и исполнитель-

ной системы, работающей в распределенной памяти. (Рис. 1). Разработчик фрагментированной программы записывает алгоритм на языке LuNA. Фрагментированная программа, записанная на языке LuNA преобразуется оптимизирующим компилятором системы LuNA во внутреннее представление фрагментированного алгоритма, которое затем может быть исполнено параллельно системой LuNA в распределенной памяти. Параллельная программа в системе LuNA представляется в виде множества фрагментов вычислений (ФВ) и фрагментов данных (ФД). ФД представляет в технологии фрагментированного программирования переменные алгоритма. ФД являются переменными единственного присваивания. Значение ФД может быть базового типа (такой, как вещественный или целочисленный) или структурированный тип. ФВ представляет собой исполняемую единицу, обрабатывающую значения своих входных ФД и вычисляющую на их основе значения своих выходных ФД. Фрагментированная программа может быть также представлена в виде двудольного ориентированного графа, вершинами которого являются ФД и ФВ, а дуги показывают информационные зависимости между ФВ. Пример фрагментирвоанной программы, представленной в виде графа, приведен на рис. 2.

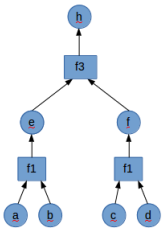

Рис. 2. Пример фрагментированной программы, представленный в виде графа.

На рисунке кружки обозначают ФД, а прямоугольники — ФВ. В данном примере системой параллельно могут быть выполнены ФВ f1 и f2 в виду отсутствия между ними информационной зависимости. Рассмотрим подробнее устройство исполнительной системы LuNA. Архитектура исполнительной системы LuNA приведена на рис. 3.

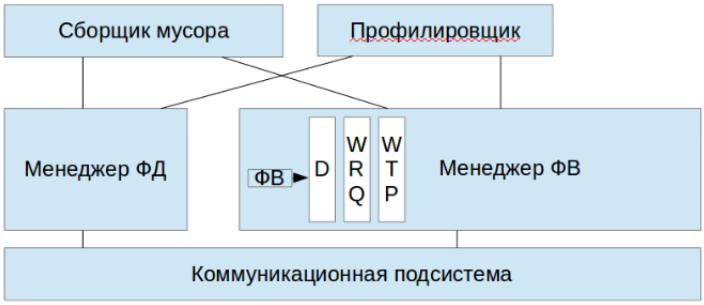

Рис. 3. Архитектура исполнительной системы LuNA.

Исполнительная система LuNA состоит из двух основных компонент: менеджера ФВ и менеджера ФД. Менеджер ФД представляет собой распределение хранилище ФД, обеспечивающее доступ системы к ФД, хранящимся в памяти узлов суперкомпьютера. Менеджер ФВ обеспечивает исполнение ФВ на узлах суперкомпьютера и представляет собой конвейер (рис. 4).

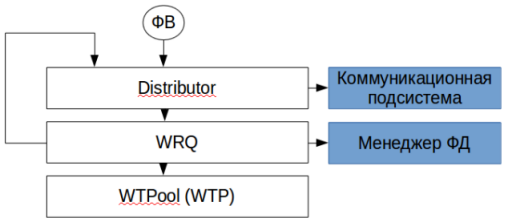

Рис. 4. Архитектура менеджера ФВ.

На стадии Distributor определяется узел суперкомпьютера, на котором ФВ будет исполнен. В случае, если целевой узел является текущим, ФВ поступает на стадию Queue, иначе происходит передача ФВ на необходимый узел с помощью коммуникационного слоя. На стадии Queue ФВ ожидает готовности всех его входных ФД. Входные ФД запрашиваются у менеджера ФД. Готовый к исполнению ФВ поступает на последнюю стадию – WTPool, представляющую собой распределенную очередь задач. ФВ помещается в очередь, откуда извлекается рабочим потоком, исполняющим ФВ.

2. Распределение нагрузки между CPU и GPU в системе LuNA
Для вовлечения GPU в вычисления с использованием системы LuNA необходимо разработать алгоритм и модуль динамического распределения нагрузки между GPU и CPU для системы LuNA. Алгоритм динамического планирования нагрузки должен назначать ФВ для вычисления на GPU или CPU так, чтобы по возможности экономить на пересылках данных между CPU и GPU. В общем случае алгоритм распределения нагрузки должен

учитывать особенности фрагментированной программы для сокращения количества пересылок между CPUи GPU. Также важно учесть наличие накладных расходов на выполнение распределения нагрузки. В ряде случаев имеет смысл применять простые алгоритмы распределения, не учитывающие особенности фрагментирвоанной программы. Такие алгоритмы применимы в случае, когда время работы фрагментирвоанной программы на одном узле суперкомпьютера невелико (составляет минуты) или, когда размеры передаваемых ФД невелики. Алгоритм распределения нагрузки должен быть реализован в виде модуля для системы LuNA. Архитектура модуля должна позволять применять для распределения нагрузки различные алгоритмы.

Далее рассматривается разработанный алгоритм распределения нагрузки для случаев, когда выгодно применять простые алгоритмы распределения нагрузки с малыми накладными расходами. Предлагается алгоритм динамического планирования нагрузки для системы LuNA. Распределение вычислительной нагрузки между CPU и GPU Nvidia внутри узла суперкомпьютера выполняется путем назначения ФВ на исполнение на CPU или GPU согласно алгоритму динамического планирования нагрузки. При назначении ФВ для исполнения на GPU входные данные автоматически копируются в память GPU. Предлагаемая архитектура менеджера ФВ с поддержкой модуля динамического планирования нагрузки приведена на рис. 5.

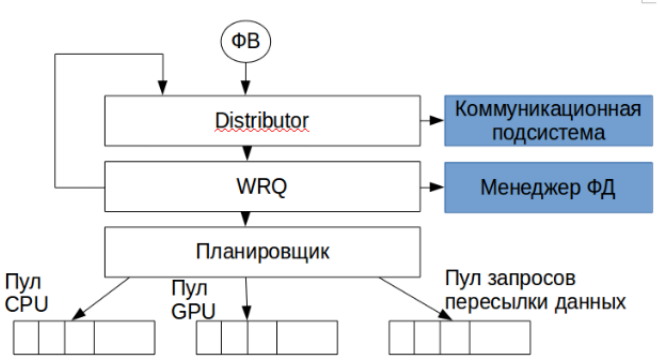

Рис. 5. Архитектура менеджера ФВ с поддержкой модуля динамического планирования нагрузки.

В архитектуру менеджера ФВ добавлен новый модуль, заменяющий пул потоков, исполняющих ФВ. Модуль состоит из планировщика, пула запросов копирования данных и пулов ФВ, назначенных для исполнения на CPU и GPU. Архитектура модуля и менеджера ФВ с поддержкой модуля распределения нагрузки позволяет применять различные алгоритмы распределения нагрузки. Модуль также является заменяемым.

Планировщик для каждого ФВ принимает решение о назначении ФВ на GPU или CPU в зависимости от текущей загрузки вычислительного

устройства и наличия реализации ФВ для исполнения на вычислительном устройстве. В случае, если в памяти вычислительного устройства находится не все множество входных ФД для назначенного ФВ, создается запрос на асинхронное копирование недостающих данных в память вычислительного устройства. Запрос затем помещается в пул запросов на копирование данных. Запросы выполняются асинхронно одним рабочим потоком. После окончания выполнения запроса, ассоциированный с ним ФВ помещается в пул ФВ, назначенных на исполнение согласно указанию планировщика. После окончания выполнения ФВ на GPU его выходные ФД остаются в памяти GPU. В дальнейшем при назначении для исполнения на GPU ФВ, входные ФД которого уже находятся в памяти GPU, их копирование не производится, что уменьшает накладные расходы на копирование данных между CPU и GPU. При необходимости ФД могут быть скопированы в оперативную память также с помощью механизма запросов на копирование данных. Память GPU освобождается в случае, если входные ФД назначенного для исполнения на GPU ФВ не помещаются в памяти GPU а также в случае, если после отработки ФВ, назначенного для исполнения на GPU, в памяти GPU не достаточно свободного места для размещения выходных ФД. При освобождении памяти, ФД, не помеченные, как удаленные сборщиком мусора, копируются в оперативную память в случае, если они отсутствуют в оперативной памяти. При этом для каждого ФД, порожденного исполнительной системой LuNA, в любой момент выполнения фрагментированной программы известен факт нахождения его копий в оперативной памяти и в памяти GPU.

Описанный алгоритм позволяет экономить на пересылках данных в случае, если ФД находятся в памяти вычислительного устройства, для выполнения на котором текущий ФВ был назначен. Алгоритм не обеспечивает оптимального распределения ресурсов, однако практически не вносит дополнительных накладных расходов на принятие решения о назначении ФВ для исполнения на вычислительном устройстве.

3. Тестирование Разработанный модуль был встроен в систему LuNA. Было проведено сравнение производительности системы со встроенным модулем и вовлечением в вычисления GPU и производительности системы без использования GPU. В качестве тестовой задачи выбрана задача параллельного умножения плотных матриц ввиду простоты реализации фрагментированного алгоритма. На рис. 6 представлены результаты тестирования. По оси абсцисс отложен размер подматрицы (ФД) x количество подматриц, по оси ординат — время выполнения.

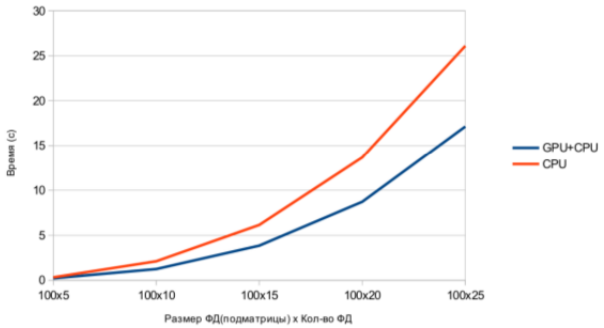

Рис. 6. Результаты тестирования.

Удалось задействовать в вычислениях, выполняемых с помощью системы LuNA, GPU, поддерживающие технологию Nvidia CUDA. Это подтверждается приростом производительности при использовании разработанного модуля распределения нагрузки между GPU и CPU.

Заключение Разработана архитектура модуля динамического планирования нагрузки для системы LuNA. Разработан алгоритм динамического распределения нагрузки между CPU и GPU для системы LuNA в соответствие с поставленными требованиями и разработанной архитектурой модуля. Модуль позволяет вовлекать в вычисления, проводимые с помощью системы LuNA GPU, построенные по технологии Nvidia CUDA. Модуль был реализован и встроен в систему LuNA. В планы входит разработка интеллектуального модуля распределения нагрузки между CPU, GPU и ускорителями Intel Xeon Phi для системы LuNA, учитывающего особенности фрагментированной программы при распределении нагрузки.

Научный руководитель

д-р тех. наук, проф. В. Э. Малышкин

Список литературы

[1] https://www.khronos.org/opencl/

[2] http://www.openacc.org/

[3] http://openmp.org/wp/

[4] Malyshkin VE, Perepelkin VA (2011) LuNA fragmented programming system, main functions and peculiarities of run-time subsystem. In: Proc. of the 11th conference on parallel computing technologies, LNCS 6873. Springer, New York, pp 53–61