РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНОЙ МОДЕЛИ ВЫЧИСЛЕНИЙ С УПРАВЛЕНИЕМ ПОТОКОМ ДАННЫХ НА КЛАСТЕРНЫХ СУПЕРКОМПЬЮТЕРАХ

Д.Н. Змеев, Арк.В. Климов, А.С. Окунев, Н.Н. Левченко

Модель вычислений с управлением потоком данных с динамически вычисляемым контекстом (МВ ПД ДВК, или просто МПД) имеет ряд хороших свойств, помогающих преодолевать проблемы, возникающие по мере повышения производительности суперкомпьютеров за пределы петафлопса [1,2,3,4,5]. Однако, основным препятствием к использованию этой модели является потребность в специально разработанном микропроцессоре [6,7]. Конечно, можно необходимые функции реализовать и программно, но в этом случае будет значительно потеряна эффективность. Для сравнения можно представить, что будет, если работу механизмов переупорядочения команд (суперскалярность) или кэш-памяти реализовать программно. Работать они будут, но пару порядков эффективности потеряют, сведя на нет все свои достоинства. Так и при реализации МПД: без аппаратной поддержки основных ее рабочих функций ожидать эффективности не приходится. Но беда в том, что микросхемы «с полки» нужные функции не поддерживают. И наоборот, поддерживают те, что для МПД не нужны (типа вышеназванных). И хотя требуемые функции, по-видимому, не намного сложнее существующих, но они другие, и еще потребуется немало усилий, чтобы довести их реализацию до сопоставимого совершенства.

В настоящее время у авторов имеется программный симулятор разрабатываемой аппартуры. Он реализована в однопроцессорном варианте и осуществляет моделирование с точностью до такта. Симулятор способен продемонстрировать потенциал возможной микросхемной реализации, но лишь на задачах очень малого размера, поскольку его произоводительность на 6-7 порядков (т.е. в миллионы раз) ниже, чем была бы у реальной многопроцессорной системы. Столь медленная реализация уже не отвечает сегодняшним потребностям.

Теперь выдвигается задача построения программной реализации модели вычислений, которая была бы в десятки тысяч раз более производительной за счет, во-первых, параллельной реализации на существующих (кластерных) суперкомпьютерах, и, во-вторых, снятия с нее требования потактового, и вообще, временного моделирования. То есть она должна обеспечивать функциональность с точки зрения пользователя с максимально возможной производительностью. Мы ориентируемся на достижение не более чем 100-кратного замедления по сравнению с гипотетической реализацией «в железе». Мы отказываемся от временного моделирования, поскольку оно вносит синхронность в процессы, которые является имманентно асинхронными, что в параллельном исполнение ведет к значительному замедлению от простоев в ожиданиях обменов. Остутствие учета времени не будет большой потерей с точки зрения оценивания адекватности моделирования, поскольку существуют и другие оценки сложности, как-то: количество выполненных узлов, количество порожденных токенов, количество токенов, переданных по сети (реальной или имитируемой). Их подсчет можно вести опционально.

Программная реализация на базе существующих кластерных суперкомпьютеров будет полезной, поскольку позволит:

пользователям — почуствовать вкус решения задач реалистичного объема с использованием данной модели вычислений (а если в задаче крупные узлы, то за счет хорошей масштабируемости можно получить реально конкурентное решение);

разработчикам — отрабатывать механизмы эффективной реализации, которые могут быть в дальнейшем перенесены в «железо»;

инвесторам — оценивать достижимые параметры целевого продукта.

Одной из ключевых функций, требующих аппаратной поддержки является сопоставляющая память токенов, работающая по принципу адресации по содержимому — т.н. ассоциативная память (АП). Другие функции — это порождение пакета (заявки на выполенние программы узла), активация вычисления программы узла, создание и посылка токена, формирование контекста (ключа), вычисление функции распределения (хеш-функции), передача токенов по назначению. С небольшими оговорками все их можно относить к накладным, тогда как содержательной частью остается только выполнение программ узлов (без операций посылки токенов).

Важной особенностью МПД является работа в парадигме раздачи, в отличие от парадигмы сбора в привычных языках и системах. Ее суть в том, что инициатива в деле передачи данных принадлежит тому узлу, который породил данные, а не тому, кто в них нуждается. Более того, производитель не нуждается (по крайней мере логически) в подтверждении получения. Это означает, что все передачи могут быть односторонними (one-way). Поэтому взаимодействие токенами в МПД — это наиболее экономный способ взаимодействия с точки знения количества передач и необходимых синхронизаций.

МПД и схема ее реализации достаточно подробно освещены в [1]. Здесь рассматриваются принципы и схема реализации модели вычислений, которую условно назовем эмулятором.

Для программирования обычно используется специальный язык – DFL. Программа на нем состоит из описаний узлов — атомарных вычислительных квантов. В заголовке узла задается перечень входов и вид контекста. Программа узла может содержать вычисления (зависящие только от входов и полей контекста) и посылки токенов вида

$$v \rightarrow N.a\{i_1,i_2,...\};$$

где v – посылаемое значение (выражение), N – имя узла назначения, a – имя входа, $i_1,\ldots$ – поля контекста (выражения). Имя узла и контекст образуют ключ, который адресует экземляр виртуального узла (ВУ).

Для работы в рамках эмулятора DFL-код транслируется в C и присоединяется к эмулятору-библиотеке линкером. Эмулятор реализуется с использованием MPI. Каждый из N_P MPI-процессов обрабатывает N_K ядер вычислительной системы, итого $K=N_P*N_K$ ядер. Каждое ядро это полнофункциональный исполнитель модели вычислений. Основной источник взаимодействий в эмуляторе это передача токенов между ядрами. Токен посылается в ядро, номер которого k ($0\le k<K$) вычисляется посредством функции распределения на основе адреса виртуального узла (ВУ), в который направлен токен. Исходя из номера ядра эмулятор определяет rank=k/N_K. Таким образом, каждый процесс должен всегда быть готов к приему токенов от любого другого процесса.

Функция распределения поставляется пользователем вместе с программой — обычно это одна формула (зависящая от полей контекста), возможно своя для каждого узла или группы узлов. Ее следует выбирать так, чтобы а) минимизировать количество обменов между ядрами и б) обеспечить относительную равномерность загрузки ядер. С точки зрения эффективности эмуляции достаточно выполнения этих свойств лишь на уровне MPI-процессов, хотя для скорости поиска желательна равномерность по ядрам и внутри процессов.

Некоторые токены могут содержать «*» в некоторых полях контекста, что понимается как посылка всем ВУ из некоторого множества. Это поле считается замаскированным и в поиске в АП не участвует. Иногда под маской возможно ненулевое значение. Тогда итоговый ключ будет дизъюнкцией сопоставляемых ключей. Чтобы такие токены не разошлись в разные ядра, должен соблюдаться принцип: от замаскированных полей функция распределения не зависит (для данного имени узла). Он может нарушаться только для глобальных токенов, которые рассылаются по всем ядрам (для них функция распределения может и не вычисляться).

Ассоциативная память (АП), в которой соединяются токены с одинаковыми ключами, может быть организована как распределенное ключевое множество с хешированием ключа для быстрого поиска. В качестве хеш-кода используется значение функции распределения — номер ядра. Ключом поиска является ключ исходного токена, в котором обнулены все поля, которые могут маскироваться в этом или любом другом встречном токене. (Будем называть его основой.) Далее можем считать, что масок и глобальных токенов нет, поскольку на описываемые принципы они почти не влияют.

По указанному хеш-кодом номеру ядра имеется свой кусочек АП, представленный для скорости поиска как бинарное дерево (например, AVL-дерево). При каждом хранимом ключе (основе) имеется фрейм, в котором хранятся ожидающие токены. На каждом входе, вообще говоря, может быть несколько токенов. Когда возникнет такая ситуация, что во фрейме есть хотя бы один токен на каждом входе, выполняется активация. Участвующие в активации токены удаляются из фрейма (если только у них не было кратности — большей 1 или бесконечной — тогда она уменьшается на 1 или остается бесконечной). При активации создается пакет со значениями входов и дизъюнкцией ключей, который ставится в очередь на исполнение. В результате выполнения узлов порождаются новые токены, которые после хеширования заносятся в буфера для передачи в другие процессы. По заполнению буфера он отправляется в виде сообщения в нужный процесс.

Основной цикл работы процесса происходит так. Начинаем, когда все входные и выходные буфера пусты. Проверяем (через IPROBE) наличие входящих сообщений и, если есть, принимаем. Если это сообщение с токенами, они заносятся во входной буфер и начинают обрабатываться. Каждый входящий токен через поиск (сначала через хеш, потом в дереве) попадает в свой фрейм, возможно порождая пакет. Созданный пакет выполняется немедленно или буферизуется. Пустые фреймы удаляются из дерева. По завершению обработки всех входных токенов и порожденных ими пакетов, непустые выходные буфера посылаются в виде сообщений. Цикл повторяется.

Завершение в норме выполняется принудительно по посылке требуемого числа выходных токенов, направленных на хост. Возможно также распознавание «тишины» по специальному алгоритму.

На первой экспериментальной версии эмулятора были опробованы некоторые простые программы, в частности решение простейшего уравнения теплопроводности по явной схеме. И хотя реализация пока далека от совершенства, результаты обнадеживают.

При наличии эмулятора описанного типа МПД может рассматриваться как альтернативная модель параллельного программирования для традиционных кластерных суперкомпьютеров. Создаваемые в ней программы обычно обладают хорошей масштабируемостью и потому, несмотря на относительно высокие накладные, при большом числе MPI-процессоров могут демонстрировать вполне конкурентную эффективность.

ЛИТЕРАТУРА:

1. Арк.В. Климов, А.С. Окунев. Потоковая модель вычислений как путь к экзафлопсу. Электрон. изд.: Труды международной суперкомпьютерной конференции «Научный сервис в сети ИНТЕРНЕТ: экзафлопсное

будущее», Новороссийск, 19-24 сентября 2011 г., [Электронный ресурс] – М.: Изд-во МГУ, 2011, с. 261-266. - URL: http://agora.guru.ru/abrau2011/pdf/261.pdf.
2. Арк.В. Климов, Н.Н. Левченко, А.С. Окунев. Преимущества потоковой модели вычислений в условиях неоднородных сетей // Информационные технологии и вычислительные системы, №2, 2012, с. 36-45.
3. Арк.В. Климов, М.И. Поликарпов. Опыт программирования одной задачи квантовой теории поля для потокового суперпроцессора ППВС «БУРАН» // XIV международная конференция «Супервычисления и математическое моделирование», тезисы, г. Саров, 1-5 октября 2012, с.105-106
4. Арк.В. Климов, Н.Н. Левченко, А.С. Окунев. Модель вычислений с управлением потоком данных как средство решения проблем больших распределенных систем // Материалы Второй Всероссийской научно-технической конференции "Суперкомпьютерные технологии" (СКТ-2012), 24-29 сентября 2012 года, с. Дивноморское, Геленджикский район.
5. Арк.В. Климов, Н.Н. Левченко, А.С. Окунев, А.Л. Стемпковский. Использование архитектуры потока данных для создания сверхвысокопроизводительных вычислительных систем // Материалы Второй Всероссийской научно-технической конференции "Суперкомпьютерные технологии" (СКТ-2012), 24-29 сентября 2012 года, с. Дивноморское, Геленджикский район.
6. В.С. Бурцев. Выбор новой системы организации выполнения высокопараллельных вычислительных процессов, примеры возможных архитектурных решений построения суперЭВМ. // В сб.: В.С. Бурцев. Параллелизм вычислительных процессов и развитие архитектуры суперЭВМ. ИВВС РАН, Москва, 1997, с. 41-78.
7. А.Л. Стемпковский, Н.Н. Левченко, А.С. Окунев, В.В. Цветков.Параллельная потоковая вычислительная система — дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов. // Информационные технологии, № 10, 2008, с. 2-7.