ПОТОКОВАЯ МОДЕЛЬ ВЫЧИСЛЕНИЙ КАК ПУТЬ К ЭКЗАФЛОПСУ

Арк.В. Климов, А.С. Окунев

Дальнейшее повышение производительности суперкомпьютеров за пределы петафлопса требует решения ряда проблем. Многие из них возникают в силу свойств традиционно используемой модели вычислений, заключающейся в представлении вычислительного процесса как совокупности взаимодействующих последовательных процессов [1] и ее разновидности – модели параллельно-последовательного программирования [2]. Есть мнение, что дальнейший рост производительности потребует перехода к иным моделям вычислений и моделям программирования. В качестве возможной альтернативной модели вычислений рассмотрим модель, основанную на управлении *потоком данных*.

Модель потока данных (ПД), или dataflow, имеет давнюю историю [3,4,5], но в основном как модель аппаратуры, позволяющая по-иному строить вычислительную систему для выполнения обычных программ. В работах по dataflow обычно даже не вводят язык этой модели в текстовом виде. Вместо этого предлагается набор графических элементов, из которых составляются схемы со связями в виде линий, по которым «движутся токены». Для программирования были предложены языки высокого уровня: Id, Val, SISAL, – по форме приближенные к обычным языкам (с функциями, блоками, условными конструкциями, циклами), но с ограничениями, состоящими прежде всего в невозможности переприсваивания. Значения, определяемые в цикле, снабжены *тегом*, или *контекстом*, содержащим номер витка цикла. Часть тега обозначает экземпляр вызова функции, часть – индекс элемента (для массива). Схема работы с контекстом, заложенная в модели, отражает ее ориентацию на классические языки программирования.

В нашей версии модели ПД предлагается забыть о циклах и массивах. Имеется огромное (общим объемом порядка 2^{64} и даже более) пространство виртуальных узлов (ВУ), обозначаемых *именем* с набором *индексов* (*полей контекста*), например X1{2,5,768}. Имя и контекст вместе составляют *адрес* ВУ. *Программа* в данной модели вычислений – это конечный набор именованных описаний узлов. В описании узла имеется заголовок, содержащий имя, формат контекста (число и типы полей), число *входов* и их типы, а также *программа узла*. Фактически используемые при вычислении ВУ являются узлами вычислительного графа. Связи между ВУ заключены в программах узлов, которые вычисляют, в зависимости от значений входов и полей контекста, некоторые новые значения и посылают их на входы других узлов. При этом перечень и адреса ВУ-получателей вычисляются программой ВУ-отправителя. Это главная особенность нашей модели ПД, которую поэтому мы называем моделью потока данных с динамически вычисляемым контекстом – ПД ДВК (по английски CEDDA – Context Evaluating Dynamic DAtaflow). В ней программа является функцией, вычисляющей получателей по отправителю и имеющейся у него информации.

Работа в модели ПД ДВК происходит следующим образом. В рабочем пуле (РП) имеется некоторое количество токенов, направленных на входы ВУ. Состав токена, как и оператор посылки токена, имеет вид:

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

где v – посылаемое значение (в программе – выражение), N – имя (типа) узла, a – имя входа, $i_1,\ldots$ – поля контекста (выражения). Некоторое неопределенное время токен «движется к цели», затем достигает ее и помещается в позицию входа ВУ. Когда у некоторого ВУ имеются в наличии токены на всех входах, ВУ *срабатывает* и формируется *пакет* – задание на выполнение программы узла, в котором заданы значения всех полей контекста и всех входов ВУ. Программа узла – обычная фон-неймановская подпрограмма. В ней могут находиться операторы посылки новых токенов. Исполненные в процессе выполнения программы узла операторы посылки создают новые токены, которые помещаются в РП. Токены, использованные при срабатывании ВУ, сразу удаляются из РП. Некоторые узлы в программе отмечены как выходные: они не имеют программы и направляемые на них токены передаются в качестве результата на хост-процессор. Исходные данные подаются из хост-процессора в виде начальных токенов на входные узлы.

Таков базовый механизм. Возможны вариации, когда специальные токены имеют специальное поведение. Например, поле контекста при отправке может быть задано звездочкой – тогда токен как бы идет на все ВУ со всевозможными значениями в этом поле. Может быть задана кратность токена (как #*n* перед стрелкой) – тогда он удалятся из РП только после *n* использований. При каждом использовании из кратности вычитается 1. Срабатывание вместе с вычитанием или удалением – атомарная операция. Есть специальные токены стирания и некоторые другие.

Рассмотрим пример разностной задачи моделирования распространения тепла в двумерной области (Рис.1). Здесь b,c,d – константы, k – номер шага по времени, i и j – координаты точки в пространстве. Входы x1 и x2 основного узла F соответствуют соседним точкам слева и справа, y1 и y2 – снизу и сверху, u – текущей точке. Узел Copy рассылает вычисленное значение по пяти направлениям. Здесь нет циклов: область задачи определяется совокупностью токенов с начальными значениями вида

$V0_{ij}$ → Copy{0,i,j}.

(имя входа для одновходового узла не обязательно), а из каждой точки границы {i,j} должен быть направлен токен с ее значением на соответствующий вход узла F каждой смежной с ней внутренней точки области:

G_{ij} ## → F.x1{*,i+1,j} .

Также на каждую точку {i,j} границы вне области следует послать токен стирания

erase ## F{*,i,j} ,

который будет стирать лишние токены, посылаемые «вовне» (символ ## означает бесконечную кратность). Для завершения на уровне k1 следует заранее послать токен «косвенность»:

@E ##→ Copy{k1,*,*},

который перенаправит все токены с узлов Copy{k1,i,j} (для всех i,j) на узлы E{k1,i,j}.

```
node F(x1,x2,u,y1,y2:real) {k,i,j};
      b*(x1+x2)+c*(y1+y2)+d*u → Copy{k+1,i,j};
node Copy(u: real){k,i,j};
       u → F.u {k,i,j},
           F.x1 {k,i+1,j},
           F.x2 {k,i-1,j},
           F.y1 {k,i,j+1},
           F.y2 {k,i,j-1};
```

Рис.1. Центральная часть программы моделирования распространения тепла в модели ПД ДВК.

В реализации предполагается, что все адресное пространство ВУ делится на примерно равные фрагменты, размещаемые на разных процессорах (ядрах). Номер процессора вычисляется как функция от адреса ВУ. Тем самым при создании токена сразу определяется, в какое ядро он должен быть передан. *Функция распределения* задается автором программы как дополнение к основному коду, которое никак не влияет на логику работы, а влияет только на производительность. Выбор функции распределения имеет целью а) обеспечить равномерность нагрузки на ядра и б) минимизировать потоки токенов между ядрами.

Проект аппаратной реализации модели ПД ДВК в виде многоядерной однокристальной системы был предложен в [6], а затем развит и обобщен до распределенной масштабируемой системы в [7]. На Рис.2 показана структурная схема системы.

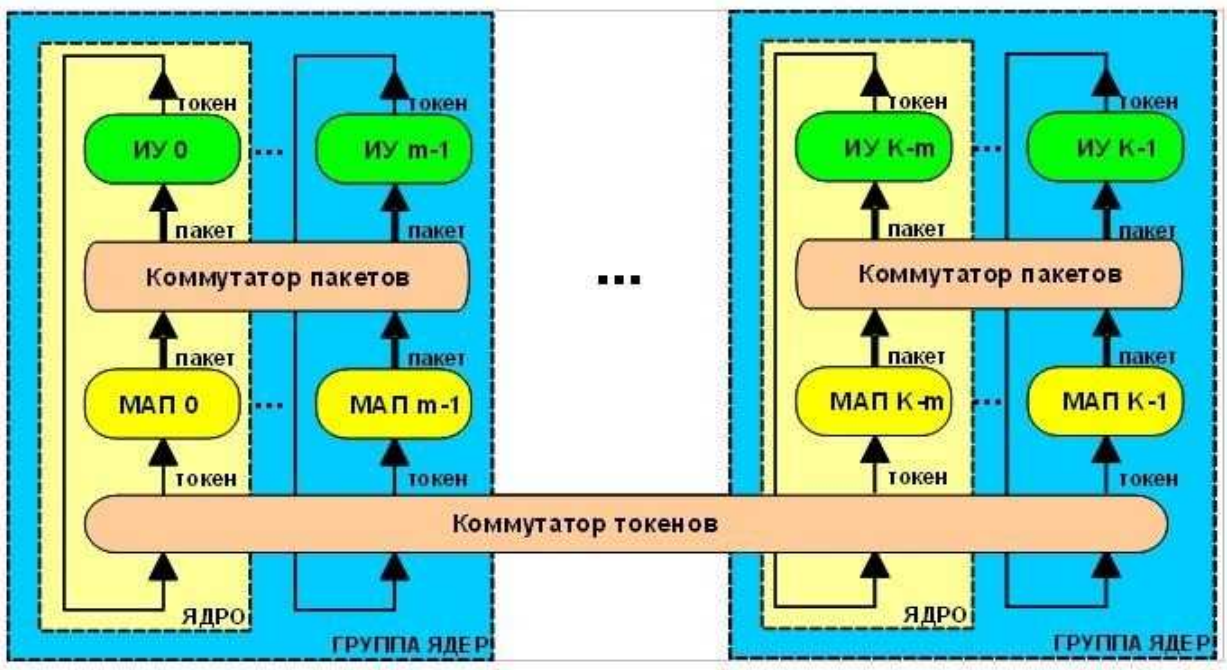

Рис. 2. Структурная схема системы (МАП – модуль ассоциативной памяти, ИУ – исполнительное устройство, m – число ядер в группе, К – общее число ядер в системе).

Между ядрами в системе передаются единицы информации – токены. Токен представляет собой структуру, содержащую данное (операнд), ключ, маску ключа, кратность и набор служебных полей. Как между ядрами в группе, так и между группами ядер действует один и тот же протокол взаимодействия, осуществляющий доставку токенов. Благодаря этому система может неограниченно масштабироваться.

Каждое ядро вычислительной системы состоит из:

- модуля ассоциативной памяти (МАП), где происходит сопоставление токенов по определенным правилам и формирование пакетов;
- исполнительного устройства (ИУ), где происходит обработка пакетов путем выполнения программы соответствующего узла, в процессе которого могут порождаться новые токены;
- блока хеш-функции, где на аппаратном уровне вычисляется функция распределения, определяющая номер целевого ядра токена.

Каждое ядро поддерживает свою локальную часть РП: осуществляет сопоставление токенов по ключам, формирует пакеты и выполняет программы узлов. Для этого и нужна ассоциативная память (АП), необходимый объем которой определяется фактически максимальным числом одновременно физически существующих виртуальных узлов в этой части РП. Виртуальный узел является физически существующим, когда на него пришел хотя бы один токен, но еще не все токены.

Реальная АП не обязательно производит физически сравнение входного ключа со всеми имеющимися ключами. Она может быть устроена как классический кэш, где частично осуществляется адресный доступ (на основе той же функции распределения, но более детальной), частично ассоциативный. Необходимая степень ассоциативности определяется степенью неоднородности используемого программой множества адресов относительно используемой функции распределения. (Отличие дисциплины работы АП от кэша лишь в реакции на отсутствие адреса: в случае кэша поиск перенаправляется во внешнюю память или внешний кэш, а в случае АП считается, что адрес отсутствует и он создается новым для нового токена.)

В целом вычислительная система, реализующая модель ПД ДВК, обладает следующими особенностями, полезными для высокопроизводительных вычислений:

- имеется ассоциативная память с развитой системой команд, на основе которой реализуется глобальная виртуальное адресное пространство (ключей токенов, или виртуальных узлов);
- система хорошо масштабируется, что позволяет реализовать возможность создания многоядерного кристалла, а в дальнейшем и высокопроизводительных систем на базе этих кристаллов;
- реализуется аппаратное выявление неявного параллелизма задачи в ходе ее решения;
- имеются аппаратно-программные средства управления параллелизмом задачи;

- имеет место асинхронность работы отдельных блоков системы;
- потоковая организация вычислительного процесса позволяет нивелировать задержки в коммуникационной сети.

Далее рассмотрим некоторые проблемы программирования, которые обостряются с ростом степени распределенности аппаратуры. Покажем, что в модели ПД ДВК их острота ниже, и они решаются проще.

Проблема «стены памяти»: доступ к удаленной памяти может занимать сотни и тысячи тактов работы процессора, которому из-за этого приходится проводить много времени в холостом ожидании. Некоторые системы пытаются ее решать путем предсказания (при компиляции или при выполнении) потребности в тех или иных данных и их упреждающей подкачки. В модели ПД ДВК эта проблема решается автоматически, во-первых, тем, что ответственность за инициативу по передаче данных от производителя к потребителю изначально возложена на производителя. Во-вторых, каждый узел срабатывает, и его программа начинает выполняться только тогда, когда все необходимые для ее работы (и завершения!) данные (входные токены) уже присутствуют локально, и потому нет необходимости приостанавливать исполняющийся тред в ожидании недостающих данных. «Ждать» приходится только токенам, находящимся в АП. Исполнительное устройство (ИУ) занято обработкой готовых к выполнению непрерываемых тредов. Главное, чтобы в задаче было достаточно параллелизма (а в модели ПД его обычно очень много).

В модели ПД ДВК аналогом проблемы «стены памяти» является проблема «стены коммуникаций», которая подобна проблеме пробок в большом городе. Причем реальной проблемой обычно является только недостаточная пропускная способность сети, на разных ее уровнях, тогда как латентность (задержка в «пустой» сети) обычно легко компенсируется наличием у задачи достаточно большого параллелизма.

Проблема необходимой однородности и синхронизации. При традиционном программировании удается добиться загрузки сотен и тысяч процессорных элементов только при выполнении большого количества однотипных действий над однотипными данными. Работа программы должна следовать схеме, показанной на Рис.3. Здесь ось времени направлена вниз, горизонтальная ось – пространственный параллелизм. Чтобы добиться хорошей загрузки, все блоки должны быть примерно равноценны и обладать естественной нумерацией, на основе которой они распределяются по ядрам «поровну». При этом каждый блок может использовать только те данные, которые были выработаны блоками «выше», причем желательно, чтобы те находились «ближе». Между уровнями требуется барьерная синхронизация, которая приводит к тому, что время уровня равно времени самого медлительного (в силу случайных причин, например, задержек в сети) блока (точнее, набора блоков, попавших на одно ядро) плюс еще чуть-чуть на выполнение барьера. Все это серьезно ограничивает класс задач, допускающих эффективное решение на суперкомпьютерах.

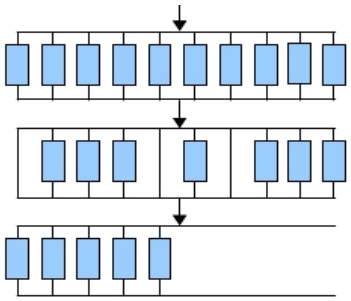

Рис. 3. Схема работы типичной хорошо распараллеливаемой программы.

В модели ПД ДВК ничего такого не требуется: каждое вычисление ждет только тех данных, которые ему необходимы, а в рамках этого условия все происходит асинхронно. Нужда в барьерных синхронизациях отпадает. Для обеспечения равномерности загрузки программисту требуется только подобрать хорошую функцию распределения, которая вычисляет номер ядра на основе адреса узла. В нашем примере функцию распределения разумно задать формулой:

$$p = zip(i, j) / d^2 \qquad 1$$

где zip – функция поразрядного скрещивания, d2 – число расчетных точек на одно ядро, p – номер ядра (окончательно будет взят остаток от деления на число ядер NP). Тогда расчетная область будет распределяться блоками размера dxd (предполагая, что d – степень 2). Этот выбор также минимизирует (при подходящем K) возникающие обмены между ядрами.

***Проблема распределения и планирования вычислений*.** Увеличение размера системы неизбежно ведет к ее большей неоднородности с точки зрения связности. Например, при одновременном попарном взаимодействии между всеми ядрами его время будет существенно зависеть от того, как размещены пары: внутри узлов (плат) или в разных стойках. (Если, конечно, сеть не реализует полную бисекцию. Но полная бисекция для $N_P > 10^6$ ядер дорого стоит.) А значит, программистам придется учитывать эту неоднородность, и тратить больше усилий (и своих и аппаратных) на распределение вычислений, которое будет лучше использовать локальность задачи на разных уровнях. И если при традиционном программировании (MPI, shmem, UPC,...) для решения данной проблемы вероятно потребуется серьезно усложнять код (разбиение на блоки разных уровней, tiling и т.п.), то в модели ПД ДВК все сведется к выбору функции распределения, причем в типовых случаях (как в нашем примере) функция zip будет обеспечивать хорошую локальность на всех уровнях, предполагая, что узлы с близкими номерами, как правило, близки и в смысле топологии сети. Более тонкая адаптация к топологии таких сетей как 3D-тор потребует использования многомерных функций распределения. Подчеркнем еще раз, что в модели ПД ДВК переход от одного способа распределения к другому

не затрагивает основной код программы и поэтому не требует переотладки.

Планирование вычислений призвано решать проблему временн***о***й локальности. В традиционных системах она проявляется как проблема эффективности использования кеша. Одним из методов ее решения является tiling. В сущности, это смена порядка обхода области, то есть определенное изменение (и усложнение) кода. В нашей модели ПД ДВК эта же проблема проявляется как проблема устранения избыточного параллелизма. Мы решаем ее путем предоставления программисту возможности указывать дополнительно *функцию распределения по времени*, которая вычисляет *номер этапа*, что также не требует внесения изменений в основной код. Смысл в том, что чем больше значение этой функции, тем позже ожидается активация данного виртуального узла. Специальный механизм обеспечивает приоритет срабатываниям узлов с меньшим номером этапа. Токены, направляемые на более поздние этапы, откладываются в дополнительной внешней памяти и подкачиваются по мере активизации новых этапов. Эксперименты показали существенное снижение требуемого объема АП на некоторых задачах.

Проблема чувствительности к задержкам в сети. Иногда не удается добиться хорошего эффекта от распараллеливания даже при наличии достаточного параллелизма и при небольшом объеме передач через сеть — когда все упирается в латентность сети. Так, в нашем примере, если задержка передачи будет заметно превосходить время, затрачиваемое каждым процессором на вычисления, время одной итерации будет определяться задержкой и не будет более уменьшаться с ростом числа процессоров. Но мы можем избавиться от излишней чувствительности задачи к задержкам, просто заменив формулу (1) на формулу

$$p = zip(i+k,\ j+k)/d^2 \qquad 2$$

Здесь k – номер шага, или время. На Рис.4 показано соответствующее «косое» разбиение пространства-времени на процессоры для одного пространственного измерения (для двух все сложнее, но эффект тоже есть, хотя и слабее), ось k направлена вверх. Область завершенных узлов (нижняя зеленая) автоматически стала пилообразной. Результирующий эффект будет таким, как если бы задержки в сети сократились в d/2 раз (предполагая, что пропускная способность сети еще не стала проблемой). В традиционных моделях программирования аналогичный эффект может быть достигнут лишь ценой значительного усложнения кода.

Проблема автоматического распараллеливания. Для модели ПД ДВК удается производить автоматически распараллеливание для вычислительных ядер, имеющих структуру регулярных гнезд циклов, в том числе таких, с которыми традиционные распараллеливатели не справляются [8,9,10]. Это связано с тем, что значительную часть работы по распараллеливанию берет на себя аппаратура, а на компилятор возлагается только задача перевода в потоковую модель вычислений.

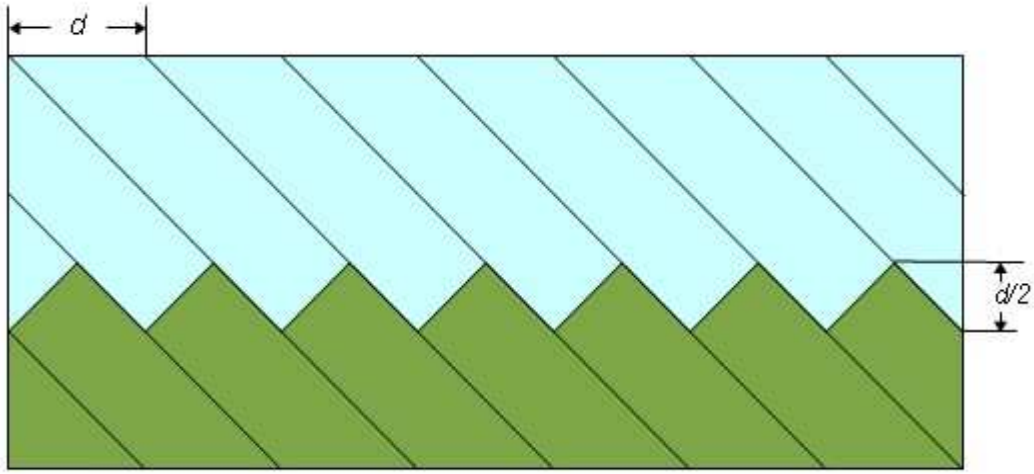

Рис.4. «Косое» распределение вычислительного пространства по формуле (2).

Заключение. По нашему убеждению представленная модель вычислений имеет много хороших свойств, которые делают ее значимым кандидатом на роль модели и технологии параллельного программирования будущего. Раньше, пока старые модели справлялись с текущими запросами, у нее не было шанса на успех, поскольку ее внедрение потребовало бы полного пересмотра архитектур вычислительных систем и принципов их программирования. Но с дальнейшим увеличением размеров и производительности суперкомпьютеров у новой модели шанс появляется, поскольку старые модели вычислений могут не справиться с новыми запросами.

ЛИТЕРАТУРА:

1. Ч. Хоар. Взаимодействующие последовательные процессы. М.: Мир, 1989, 264 с.
2. Е. Валях. Последовательно-параллельные вычисления. М.:Мир, 1985. 456 с.
3. J.R.W. Glauert, J.R. Gurd, C.C. Kirkham. Evolution of a Dataflow Architecture. In: Concurrent Languages

in Distributed Systems: Hardware Supported Implementation. Eds: G.L. Reijns, E.L. Dagless. North Holland Publishing Company. January 1985, pp. 1-18.
4. J.R. Gurd, C.C. Kirkham, I. Watson. The Manchester Prototype Dataflow Computer. Communications of the ACM, vol.28 no.1, January 1985, pp. 34-52.
5. G. Papadopoulos. Implementation of a General Purpose Dataflow Multiprocessor. PhD thesis, Massachusetts Institute of Technology, 1998.
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, С.А.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.
8. Арк.В. Климов. Использование деревьев выбора для описания состояний в распараллеливающем компиляторе. // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность. Труды Всероссийской суперкомпьютерной конференции, М.: Изд-во МГУ, 2009, с.238-240.
9. Арк.В. Климов. Трансляция последовательной программы в потоковый язык как способ распараллели-вания. Материалы Международной научно-технической конференции «Суперкомпьютерные техно-логии: разработка, программирование, применение», Дивноморское, 27 сент.-2 окт. 2010, с 246-250.
10. Арк.В. Климов, Н.Н. Левченко, С.А. Окунев, А.Л. Стемпковский. Автоматическое распараллеливание последовательных программ для гибридной системы с ускорителем на основе потока данных. Сборник трудов Международной конференции «Параллельные вычислительные технологии» ПаВТ'2011, МГУ, Москва, 28 марта — 1 апреля 2011.