Преимущества потоковой модели вычислений в условиях неоднородных сетей

А.В. Климов, Н.Н. Левченко, А.С. Окунев

Аннотация. С увеличением мощности суперкомпьютеры становятся все менее однородными, менее симметричными. В зависимости от расстояния между узлами времена передачи сообщений между узлами могут отличаться в десятки раз. Чтобы уменьшить соответствующие потери, обычно приходится учитывать эти различия «руками» при программировании. В предлагаемой версии модели вычислений, основанной на управлении потоком данных, эти и некоторые другие проблемы решаются автоматически, что способствует снижению трудоемкости разработки программ и повышению их переносимости. В статье эффект автоматической динамической адаптации объясняется и демонстрируется на примере простой задачи.

Ключевые слова: модель вычислений с управлением потоком данных, неоднородность доступа, стена памяти, асинхронное программирование, динамическая адаптация.

Введение

С увеличением мощности суперкомпьютеры становятся все менее однородными и симметричными. Возросшая структурная иерархия оказывает заметное влияние на коммуникационные возможности. В зависимости от расстояния между узлами (которое следует понимать как меру объема наименьшего структурного компонента системы, содержащего оба узла) времена передачи сообщений между узлами (задержки) могут отличаться в десятки раз. С увеличением расстояния, как правило, падает также пропускная способность коммуникационной сети в пересчете на один узел.

Обычно программы для суперкомпьютеров пишутся в расчете на двухуровневую модель вычислителя, в которой имеются узлы с быстрым доступом к локальной памяти и большим временем передачи данных другим узлам (или доступа к памяти другого узла), причем это время считается одинаковым для всех пар узлов. И когда в реальности эти времена сильно различаются, такие программы работают плохо – так, как если бы времена и скорости всех передач сравнялись с наихудшими. Возникает нетривиальная задача адаптировать программы к таким неоднородностям, что, однако, может отрицательно сказаться на переносимости программ. И уж совсем плохо, когда для оптимальной работы на разных уровнях приходится применять разные алгоритмы.

Одна из причин возникающих сложностей состоит в том, что при распараллеливании мы изначально стремимся явно спланировать, что с чем в нашей задаче будет выполняться параллельно и что за чем будет следовать. При этом мы используем глобальные барьеры, эффективные коллективные операции и повышаем производительность за счет того, что делаем это все лучше. И хорошо еще, если у нас один-два уровня неоднородностей (например, CPU–GPU). Но дальше на этом пути сложность программирования стремительно растет.

Хотелось бы, чтобы адаптация к различным неоднородностям происходила автоматически. Из вышесказанного следует, что для этого целесообразно отказаться от идеи планировать

параллелизм заранее и весомую часть работы по распараллеливанию выполнять динамически, без лишних синхронизаций. Одной из известных моделей вычислений с указанным свойством является модель, основанная на управлении потоком данных. Мы предлагаем нашу версию потоковой модели вычислений, для которой в ИППМ РАН разрабатывается схема и модель аппаратной реализации. В этой статье на простом примере покажем, как в предлагаемой модели происходит автоматическая адаптация к неоднородностям коммуникационной среды. Будут рассмотрены и другие проблемы, связанные с неоднородностью и иерархичностью вычислительной системы, и пути их решения в нашей модели вычислений. Но сначала опишем нашу модель вычислений с управлением потоком данных, учитывая, что она заметно отличается от классической, известной под таким же названием (dataflow) уже более 40 лет.

Классическая модель dataflow возникла как модель аппаратуры, позволяющая по-иному строить вычислительную систему для выполнения обычных программ на обычных или слегка видоизмененных языках. В работах по dataflow [3-5] даже не предлагалась текстовая форма языка программирования в этой модели. Взамен обычно предлагался 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. В реализации значения, определяемые в цикле, снабжены тегом, или контекстом, содержащим номер витка цикла. Часть тега обозначает экземпляр вызова функции, часть – индекс элемента (для массива). В целом схема работы с контекстом встроена в систему и слабо контролируется программистом, отражая ориентацию на обычные языки программирования.

Описание модели вычислений

Наша версия модели вычислений с управлением потоком данных не ставит целью поддержать традиционную модель программирования[1]. В ней нет традиционных циклов и массивов. Вместо них имеется огромное (в реализации – до 264) пространство виртуальных узлов (ВУ), идентифицируемых *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*. Реально используемые 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 в виде *токенов*. При этом набор выдаваемых токенов, включая как данные, так и адреса ВУ-получателей, формируется программой ВУ-отправителя. Это главная особенность нашей модели потока данных (ПД), поэтому мы называем ее моделью потока данных с динамически вычисляемым контекстом – ПД ДВК. В ней программа является функцией, вычисляющей набор выходных токенов (со всеми их атрибутами) по узлу-отправителю и данным, содержащимся во входных токенах этого узла.

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

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

где v – посылаемое значение (в тексте программы – выражение), N – имя (типа) узла, а – имя входа, $i_1,i_2,\ldots$ – значения полей контекста (выражения). Некоторое неопределенное время токен «движется к цели», затем он достигает ее и помещается в позицию входа определенного 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. Про-

[1] Хотя для ограниченного класса программ – так называемого линейного, или аффинного – такая поддержка возможна и предлагается.

```
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 распространения тепла в модели ПД ДВК

грамма узла похожа на обычную фоннеймановскую подпрограмму. 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, обычно 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 идет на все ВУ как бы со всевозможными значениями в этом поле. Может быть задана кратность токена (#*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. Одновходовый узел Copy рассылает вычисленное значение по пяти направлениям. Здесь нет циклов: область задачи определяется совокупностью токенов с начальными значениями вида

$$V0_{ij} \rightarrow Copy\{0,i,j\},$$

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:

$$G_{ij}\ \#\# \rightarrow F.x1\{*,i+1,j\}.$$

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

$$\textbf{erase}\ \#\#\ F\{*,i,j\}\ ,$$

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

$$@E\ \#\# \rightarrow Copy\{k1,*,*\},$$

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

Реализация модели

Может показаться, что данная модель вычислений предполагает наличие узкого горла – единого хранилища токенов, где встречаются токены с общим именем и контекстом. Но это совсем не так. В целях эффективности реализации 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, причем оно никак не влияет на логику работы, а влияет только

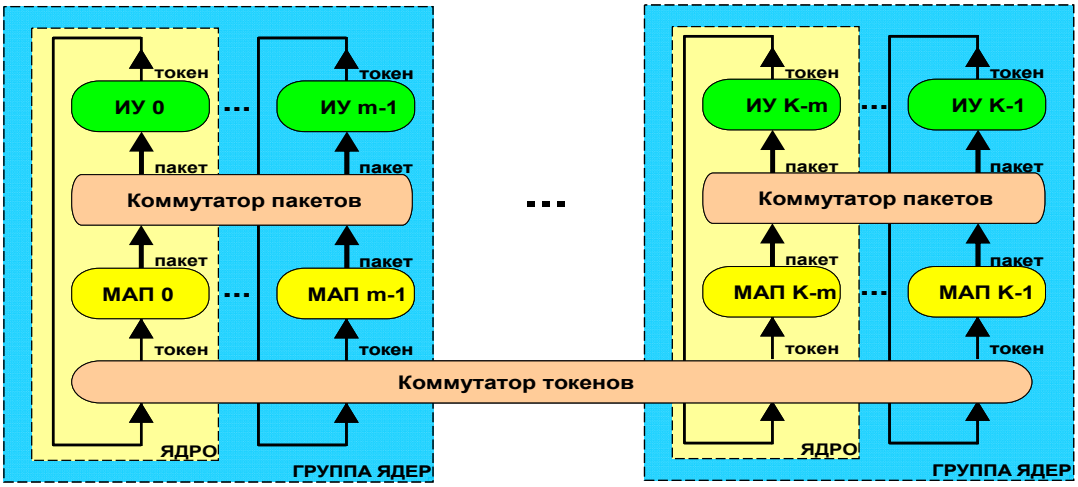

Рис. 2. Структурная схема системы

МАП — модуль ассоциативной памяти, ИУ — исполнительное устройство, m — число ядер в группе, K — общее число ядер в системе

на производительность. Выбор функции распределения имеет следующие цели:

- обеспечить равномерность нагрузки на ядра;
- минимизировать потоки токенов между ядрами;
- снизить чувствительность программы к задержкам в сети.

Сама модель вычислений вносит в реализацию определенные накладные расходы, к которым можно отнести следующие:

- организация ассоциативного доступа;
- меры по предотвращению переполнений;
- организация динамической памяти для многовходовых виртуальных узлов;
- накопление токенов в виртуальном узле;
- выполнение проверок условия готовности при каждом приходе токена;
- формирование пакета из набора токенов;
- активация пакета;
- формирование токена;
- вычисление функции распределения;
- пересылки токенов, в том числе с использованием broadcast и multicast.

Если эти функции вычислять программно, используя обычные процессоры, применяемые в суперкомпьютерах, то на фоне мелкозернистости программы (определяемой размером программ узлов) эти накладные расходы будут весьма велики. Но поскольку это конкретные функции и их перечень ограничен, есть смысл поддержать их аппаратно. Единственная из перечисленных функций, требующая программного управления, – это вычисление функций распределения, которые задаются пользователем-программистом. Но сам класс этих функций весьма специфичен и может быть положен на программируемые логические схемы.

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.

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, их накопление и формирование пакетов;

- исполнительного устройства (ИУ), где происходит обработка пакетов путем выполнения программы соответствующего узла, в процессе которого могут порождаться новые токены;
- блока хеширования, 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 становится физически существующим, когда на него придет хотя бы один токен, но еще не все необходимые для его исполнения токены. Тогда в АП появляется ключ с адресом этого узла, и в динамической памяти отводится место для накопления токенов (данных) для этого узла. Когда придет последний токен из числа необходимых, будет сформирован пакет (возможно, в этом же месте), который будет поставлен в очередь на исполнение в ИУ.

Реальная АП одного ядра 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.

Новые токены формируются в ИУ специальными командами. Эти операции также должны быть поддержаны на аппаратном уровне, поскольку сопряжены с очень специфическими действиями, которые было бы накладно выполнять программно. Роль этих команд подобна роли операций записи-чтения по глобальному адресу в системах типа PGAS. Отличие в том, что здесь используются только односторонние посылки типа записи (нет двусторонних посылок типа чтения) и токен несет код-команду, определяющую действие, которое необходимо сделать в целевом ядре (с указанием узла, который, возможно, будет активирован). На выходе из блока хеширования стоит блок сравнения номера целевого ядра с номером своего ядра, который при совпадении перенаправляет токен сразу в свой МАП, минуя глобальный коммутатор токенов.

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 глобальное виртуальное адресное пространство (ключей токенов, или виртуальных узлов);
- система хорошо масштабируется, что дает 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.

Основной недостаток данного проекта заключается в том, что для его реализации требуется разработка специального «железа» в виде многоядерных кристаллов нового типа, а также специальной коммуникационной среды, с нового типа интерфейсами и т.п. Иначе, в силу специфичности описанных выше операций, может быть потеряно 1–2 порядка производительности, что сделает результат бессмысленным. Тем не менее мы полагаем, что реализация данной модели вычислений на существующей элементной базе тоже имеет смысл как прототип, демонстрирующий возможности новой архитектуры на различных задачах. А для отдельных задач, где велик удельных вес вычислений в узлах, возможен и реальный выигрыш.

Реализация данной модели на обычных кластерных системах потребует эффективной программной реализации следующих функциональных компонентов:

- ассоциативная память в виде хеш-таблиц, возможно многоуровневых;
- библиотека передачи сообщений, ориентированная на случайный поток односторонних коротких сообщений (имеющая внутри средства агрегирования, сброса по тайм-ауту, буферизации, broad- и multi-casting);
- динамическая память (для состояний виртуальных узлов и больших элементов данных);
- многопоточный (многотредовый) исполнитель пакетов (возможно, объединенный с блоком формирования пакетов);
- генератор эффективного кода формирования токенов (в составе компилятора входного языка DFL), в том числе кода вычисления функций распределения;
- генератор эффективного кода обработчиков токенов и формирователей пакетов (или вызова программ узлов).

Проблемы масштабирования

Теперь обратимся к рассмотрению проблем выполнения параллельных программ на больших суперЭВМ с иерархической структурой. 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 (точнее, их разработчики) пытаются решать эту проблему путем предсказания (при компиляции или при выполнении) потребности в тех или иных данных для их упреждающей подкачки или откачки. Другие системы стремятся поддержать одновременно сотни и тысячи легких тредов, малая часть которых займет работой процессор, пока остальные ждут прихода результатов чтения из памяти. 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, то есть передачи только односторонние, не требующие ответа, как в операции чтения. Во-вторых, каждый узел активируется, и его программа начинает выполняться только тогда, когда все необходимые для ее работы (и полного завершения!) данные уже присутствуют локально, и потому не будет необходимости приостанавливать исполняющийся тред в ожидании недостающих данных. «Ждать» приходится только токенам, находящимся в АП, пока их набор неполон. 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. Она подразделяется на две независимые друг от друга проблемы: проблема латентностного барьера, вызванного ограничением (снизу) на время передач (латентность), и полосового барьера, вызванного ограничением (сверху) пропускной способности сети. Причем последнее выражается либо в гигабайтах в секунду, либо в числе сообщений в секунду. Все эти барьеры проявляются в том, что задача перестает масштабироваться, начиная с некоторого числа процессоров. Лантентностный барьер может компенсироваться наличием у задачи достаточно большого параллелизма – но при условии, что либо программа, либо сама модель вычислений позволяют совмещать одни вычисления с передачей данных для других. В модели ПД ДВК это происходит практически без участия программиста. Полосовые барьеры составляют реальную проблему при любой модели вычислений и должны решаться путем реального снятия этих ограничений. И действительно, в последние годы по этой части наблюдается заметный прогресс.

Проблема ***необходимой однородности и синхронизации.*** При традиционной модели программирования [2] 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, расположенными «выше» (то есть раньше), причем желательно, чтобы те находились пространственно «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 некоторое время на выполнение барьера. Все это серьезно ограничивает класс задач, допускающих эффективное решение на суперкомпьютерах.

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 по большому счету 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, d^2 – число расчетных точек на одно ядро, p – номер ядра (окончательно будет взят остаток от деления на число ядер N_P). Тогда расчетная область будет распределяться блоками размера $d \times d$ (имеется в виду, что d – степень двойки). 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 большого количества ядер стоит дорого). 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. При традиционном программировании (в моделях MPI, shmem, UPC, и т.п.) для решения данной проблемы, как правило, требуется серьезно усложнять код, используя разбиение на блоки разных уровней, tiling и т.д. А в модели ПД ДВК все сводится к выбору функции распределения, причем в типовых случаях, когда имеется многомерное пространство с локальным взаимодействием (как в нашем примере), функция zip будет обеспечивать хорошую локальность одновременно на всех уровнях. Более тонкая адаптация к топологии таких сетей как 3D-тор потребует использования многомерных функций распределения. Отметим, что в модели ПД ДВК переход от одного способа распределения к другому не затрагивает основной код программы и поэтому не требует перепрограммирования и переотладки.

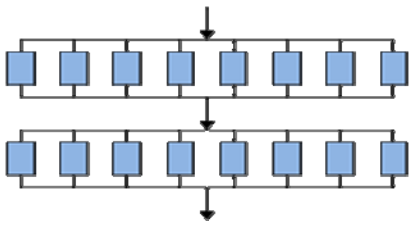

Рис. 3. Схема работы типичной хорошо распараллеливаемой программы

Планирование вычислений призвано решать проблему локальности по времени. В традиционных системах она проявляется как проблема эффективности использования кэша. 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 (Рис. 4). Но мы можем избавиться от излишней чувствительности задачи к задержкам, просто заменив формулу (1) на формулу

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

Здесь k – номер шага, или время. На Рис. 5 показано соответствующее «скошенное» 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 автоматически стала пилообразной. Результирующий эффект будет таким же, как если бы задержки в сети сократились в d/2 раз (предполагается, что пропускная способность сети еще не стала проблемой). В традиционных моделях программирования аналогичный эффект может быть достигнут лишь ценой значительного усложнения кода (особенно для двумерных и трехмерных задач).

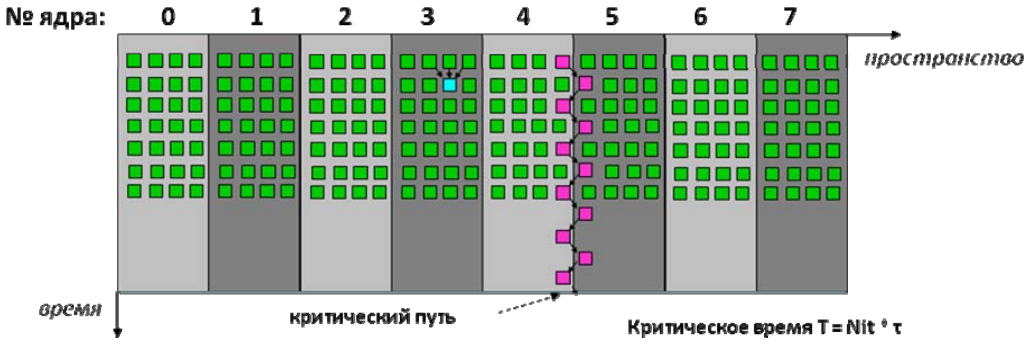

Рис. 4. Влияние задержки передачи между ядрами на эффективность при прямом распределении

Время вычисления одной итерации ограничено снизу величиной (самой долгой) задержки

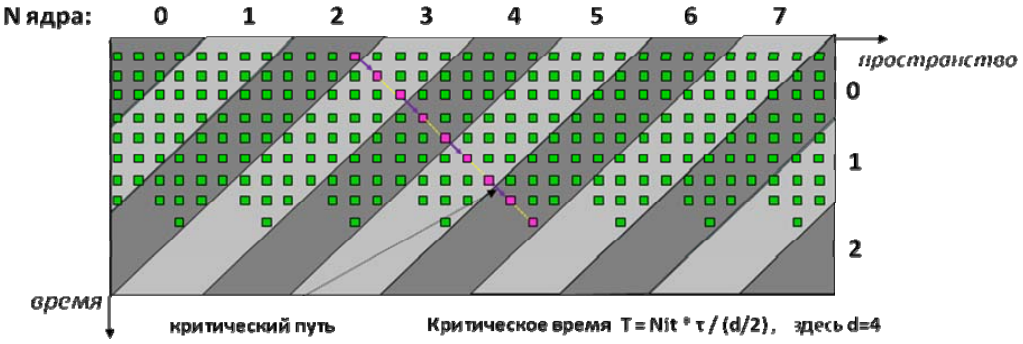

Рис. 5. Влияние задержки передачи между ядрами на эффективность при скошенном распределении

Величиной задержки ограничено снизу время вычисления d/2 (здесь 2x) итераций, где d — ширина области расчетного поля, приходящейся на одно ядро

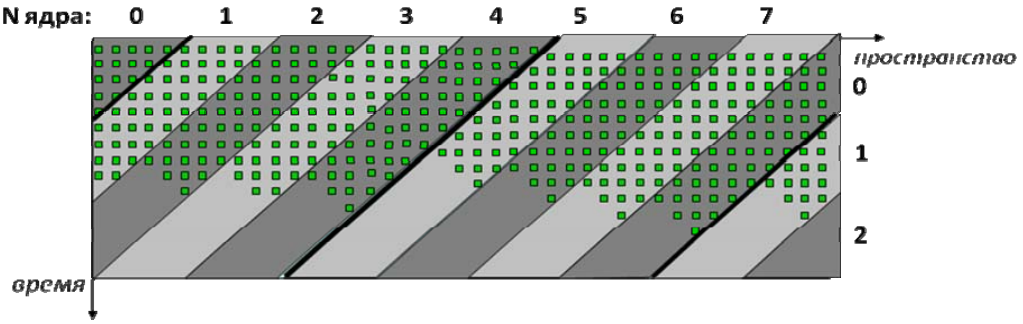

Рис. 6. Адаптация в модели ПД ДВК к различиям в задержках

Линия фронта автоматически становится пилой с зубьями разного размера, компенсируя различия в задержках величиной отставания

Но самое интересное произойдет в ситуации, когда задержки между парами соседних ядер будут различными (Рис. 6). Тогда величина отставания правой области от левой для каждой пары соседних ядер будет своей, и соответственно будут меняться размеры зубьев. Но общий темп работы по всему фронту будет попрежнему максимально возможным, при учете объема вычислений и/или пропускной способности каналов, но не задержек.

Проблема автоматического распараллеливания. Для модели ПД ДВК удается автоматически распараллеливать вычислительные коды, имеющие вид регулярных гнезд циклов, в том числе такие, с которыми традиционные распараллеливатели не справляются [8-10]. 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.

Заключение

Представленная модель вычислений демонстрирует хорошие свойства, обеспечивающие автоматическое решение ряда проблем, возникающих при масштабировании задач на больших системах с неоднородной связностью. Можно говорить [11] о перспективности применения этой модели вычислений в будущих экзафлопсных системах.

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, 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.
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 распараллеливания. Материалы Международной научно-технической конференции «Суперкомпьютерные технологии: разработка, программирование, применение», Дивноморское, 27 сент.–2 окт. 2010 г., с. 246–250.
10. Арк.В. Климов, Н.Н. Левченко, А.С. Окунев, А.Л. Стемпковский. Автоматическое распараллеливание последовательных программ для гибридной системы с ускорителем на основе потока данных. Параллельные вычислительные технологии (ПаВТ'2011): труды международной научной конференции (Москва, 28 марта–1 апреля 2011 г.) [Электронный ресурс] – Челябинск: Издательский центр ЮУрГУ, 2011, с. 211–218 – URL: http://omega.sp.susu.ac.ru/books/conference/PaVT2011.

11. Арк.В. Климов, А.С. Окунев. Потоковая модель вычислений как путь к экзафлопсу. Труды Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: экзафлопсное будущее» (19–24 сентября 2011 г., г. Новороссийск), электронное издание, с. 261–266.

Климов Аркадий Валентинович. Старший научный сотрудник Института проблем проектирования в микроэлектронике РАН. Окончил Московский государственный университет в 1976 году. Автор более 20 научных работ. Область научных интересов: языки программирования и модели вычислений, семантика языков программирования, методы анализа, преобразования и синтеза программ, нестандартные вычислительные модели и архитектуры, параллельные вычисления и алгоритмы, языки параллельного программирования, архитектуры параллельных ЭВМ.
E-mail: klimov@burcom.ru.

Левченко Николай Николаевич. Старший научный сотрудник Института проблем проектирования в микроэлектронике РАН. Окончил Московский авиационно-технологический институт (технический университет) им.К.Э. Циолковского (МАТИ) в 2001 году. Кандидат технических наук. Автор более 60 научных работ, включая монографию и патент на изобретение. Область научных интересов: параллельные вычисления, параллельное программирование, модель вычислений управляемых потоком данных, использование ассоциативной памяти в высокопроизводительных вычислительных системах. E-mail: nick@ippm.ru.

Окунев Анатолий Семенович. Старший научный сотрудник Института проблем проектирования в микроэлектронике РАН. Окончил Московское высшее техническое училище им. Н.Э. Баумана (МВТУ) в 1974 году. Кандидат технических наук. Автор более 60 научных работ, включая монографию и патент на изобретение. Область научных интересов: архитектуры высокопроизводительных вычислительных систем, модель вычислений с управлением потоком данных, применение различных САПР для реализации архитектурных решений, применение принципов ассоциативности при разработке различных аппаратных узлов. E-mail: oku@ippm.ru.