УДК 004.2+004.4
DOI: 10.7587/prin.13.471-482

В. В. Корнеев, д-р техн. наук, проф. гл. науч. сотр., korv@rdi-kvant.ru,
ФГУП «Научно-исследовательский институт "Квант"», Москва

Маршрутизация в коммуникационной среде вычислительной системы с распределенной разделяемой памятью и синхронизацией на базе FE-битов

Рассмотрена реализация распределенной разделяемой памяти с синхронизацией на базе FE-битов. Предложена арифметическая маршрутизация в коммуникационной среде вычислительной системы с распределенной разделяемой памятью, устраняющая взаимовлияние параллельных программ, исполняемых в разных подсистемах, и снижающая энергозатратность коммутаций в целом.

Ключевые слова: *архитектура вычислительной системы с распределенной разделяемой памятью, модель параллельного программирования, координатная адресация, энергоэффективная маршрутизация, арифметическая маршрутизация*

Введение

Существующие суперкомпьютеры ориентированы на достижение высокой производительности при проведении больших вычислений, например, вычислений с большими плотными матрицами. Однако их производительность падает до неприемлемого уровня при работе с большими данными. К их числу относятся, например, данные, представленные большими графами, в том числе потоками таких данных [1, 2], решения по выявлению фактов из больших потоков данных, поступающих из большого числа разнообразных источников.

Для работы с большими данными с приемлемой производительностью необходимы новые архитектурные подходы. Их поиск ведется среди как специализированных архитектур, так и расширения возможностей существующих мультитредовых архитектур. Так, для решения задач с большими графами предложена специализированная архитектура [3, 4] на базе алгоритмов библиотеки GraphBLAS [5], ориентированная на работу с большими разреженными матрицами, используемыми для представления графов. В предлагаемой архитектуре элементы разреженных матриц и элементы результирующей матрицы распределяются по процессорным элементам. В каждом процессорном элементе вычисляются частичные результаты из размещенных в них элементов исходных матриц. Эти результаты передаются в соответствующие процессорные элементы, в которых формируются элементы результирующей матрицы. В процессорных элементах используется аппаратный ускоритель — систолическая сортировка.

Более универсальная мультитредовая EMU-архитектура предложена в работе [6]. Ее особенностью является множество мультитредовых процессорных элементов, каждый из которых имеет "узкий" байтовый доступ к собственному блоку некешируемой памяти, что увеличивает пропускную способность памяти за счет большого числа блоков и обмена только необходимым числом байтов. Общее адресное пространство распределено по процессорным элементам, и тред, обращающийся к слову памяти, адрес которого в другом процессорном элементе, мигрирует в этот элемент. Наряду с этим используется также технология PCI по образованию разделяемой памяти, доступной всем процессорным элементам, что делает возможным размещение в ней программ, данных и констант, общих для всех тредов.

В работе [7] предложена архитектура PIUMA (*programmable integrated unified memory architecture*), которая обобщает EMU-архитектуру [6], добавляя в процессорном элементе к множеству мультитредовых ядер специализированные ускорители

(*memory offload engines*): канал прямого доступа к памяти (DMA) с возможностями копирования, сборки, рассылки (*copy*, *scatter*, *gather*) данных между блоками памяти, а также ускорители для работы с очередями (*queue engines*), ускорители-синхронизаторы (*collective engines*), обеспечивающие выполнение атомарных операций и барьерную синхронизацию. В отличие от EMU-архитектуры, распределение тредов по ядрам управляется программно без автоматической миграции тредов к обрабатываемым данным.

Хотя существуют отдельные оценки производительности рассмотренных архитектур, например, на задаче умножения разреженной матрицы на плотный вектор [7, 8], в целом неизвестно о реализации систем с этими архитектурами и, соответственно, об их возможности решать актуальные задачи с большими данными в приемлемое время. Вместе с тем во всех архитектурах делается упор на повышение эффективности работы с разделяемой памятью. Следует также отметить, что предлагаемая разделяемая память реализуется на базе традиционных коммуникационных сред, мало подходящих для интенсивных обращений к большим данным, в том числе в силу их энергозатратности.

Существующие суперкомпьютеры демонстрируют очевидный предел по потребляемой энергии, что делает проблематичным их экстенсивное развитие при сохранении используемых архитектурно-технических решений. Суперкомпьютер Frontier потребляет более 20 МВт при проведении вычислений с большими плотными матрицами, достигая производительности 1,102 Экзафлоп/с на 8 730 112 вычислительных ядрах [9]. Архитектурно-технические решения, принятые в суперкомпьютере Frontier, представляют достигнутый передний край, что подтверждается первым местом в соответствующем списке энергоэффективных компьютеров [10].

Коммуникационная среда Slingshot [11] вносит весомый вклад в достижение энергоэффективности суперкомпьютера Frontier. В Slingshot функции сети Ethernet реализуются без энергозатратных типовых решений этой сети с использованием контекстно-адресуемой памяти. В ней аппаратно встроены алгоритмы поиска наименее нагруженных маршрутов, обеспечения соглашения об уровне обслуживания, введения классов пакетов и другие алгоритмы, ранее опробованные в программных продуктах управления коммуникационными сетями суперкомпьютеров.

В настоящее время идет интенсивное исследование способов построения энергоэффективных отказоустойчивых коммуникационных сред с высокой пропускной способностью и малой латентностью. Например, в работе [12] рассмотрены различные алгоритмы маршрутизации, в том числе отказоустойчивой, устраняющей влияние сетевых отказов на исполнение прикладных программ. В большинстве коммуникационных сред современных суперкомпьютеров для маршрутизации пакетов используются таблицы, размещаемые для ускорения поиска соответствия в блоках энергозатратной ассоциативной памяти, объем и число которых также возрастают с увеличением числа вычислительных модулей. По этой причине интенсивно исследуются возможности создания арифметической маршрутизации [13], при которой маршрут задается самим адресом и определяется путем достаточно простых вычислений.

В целом для коммуникационной среды определяющим фактором эффективности служит граф связей — топология и алгоритм маршрутизации. Все остальные алгоритмы, направленные на устранение конфликтов, "укорочение" хвоста распределения задержек (*tail of latency distribution*), обеспечения качества обслуживания и отказоустойчивости, применимы, как правило, для всех топологий и маршрутизаций.

Часто используемой топологией служит Dragonfly, рекомендуемая в качестве основной в коммуникационной среде Slingshot, но в работе [11] отмечено, что существуют другие топологии с меньшей длиной среднего пути, но для них надо использовать более сложные и длительно выполняемые энергозатратные маршрутизации. В работе [14] показано преимущество топологии с меньшей длиной среднего пути по сравнению с другими топологиями, включая Dragonfly, при выполнении коллективных операций MPI и ряда тестовых программ, таких как NAS Parallel Benchmarks (NPB version 3.3.1 on MPI) и Graph 500.

В настоящей статье рассмотрена архитектура, базирующаяся на парадигме использования всего возможного параллелизма обработки [15—18]. Пользователь должен только указывать, какие вычисления можно проводить параллельными потоками над общей разделяемой памятью, сообразуясь только с выбранным алгоритмом. Такой подход позволяет создать максимальный поток обращений к памяти, присущий алгоритму. При необходимости прочитать ячейку общей памяти одним потоком, и только потом записать в нее новое значение другим потоком, пользователь полагает, что механизм разрешения конфликта реализуется аппаратными средствами управления доступом к памяти. В целом предлагаемая архитектура направлена на решение тех же задач, что и архитек-

туры EMU и PIUMA. Однако она использует для синхронизации тредов и реализации атомарных операций "умные" смарт-контроллеры блоков разделяемой памяти.

Для большого потока обращений к распределенной разделяемой памяти необходима энергоэффективная маршрутизация. В настоящей статье предложена арифметическая маршрутизация, которая применима в любых коммуникационных средах, в том числе с графами межмашинных связей Dragonfly и Л(N, v, 7) с числом вершин N и степенями вершин v. При диаметре 3 графы Л(N, v, 7) имеют минимально возможную длину среднего пути [15, 16]. Предложен алгоритм маршрутизации, обеспечивающий отказоустойчивое функционирование на базе возможности выбора альтернативных маршрутов.

Основы архитектуры для работы с большими данными

Экзафлопсные компьютеры строятся из вычислительных модулей (ВМ), каждый из которых имеет совокупность универсальных и специализированных вычислительных ядер, блок управления памятью (*Memory Management Unit* — MMU), смарт-контроллеры блоков памяти, сами блоки памяти и сетевые контроллеры коммуникационной среды, объединяющей все вычислительные модули [15, 17, 18]. Число смарт-контроллеров, собственно блоков памяти и сетевых контроллеров определяется исходя из необходимости обеспечения пропускной способности, минимизирующей простой ядер ВМ в ожидании завершения доступа к данным.

Распределенная разделяемая память суперкомпьютера состоит из совокупности блоков памяти, размещенных в каждом ВМ. Блоки распределенной разделяемой памяти, размещенные в ВМ, будем называть близкими для процессорных ядер этого ВМ, все остальные — удаленными. Число блоков памяти в каждом ВМ, с одной стороны, должно быть как можно больше, что обеспечивает увеличение доли обращений к близким блокам памяти. С другой стороны, следует иметь в виду, что их число ограничивается сложностью управления и конструкцией ВМ.

При инициации суперкомпьютера выполняется конфигурация глобального адресного пространства путем распределения адресного пространства по блокам памяти. Это распределение фиксируется в блоках управления памятью MMU, основной функцией которых является направление запросов к контроллерам блоков памяти с соответствующими адресами.

Каждое ядро при выполнении текущим тредом команды доступа к разделяемой памяти по чтению после выдачи обращения к памяти приостанавливает этот тред до получения ответа от памяти.

Обращение треда к памяти по чтению или по записи помещается в очередь необслуженных запросов к памяти в блоке управления памятью MMU. Далее MMU определяет, куда идет обращение — к близкому или удаленному блоку памяти другого ВМ.

В случае обращения к близкому блоку памяти это обращение посылается в соответствующий смарт-контроллер памяти.

Для сокращения трафика и ускорения работы с памятью может быть реализована модель программирования с локальными чтениями и локальными или удаленными записями [19]. Так описано, например, в работах [6, 20].

При обращении к близкому блоку по записи соответствующий запрос удаляется из очереди необслуженных запросов к памяти в блоке управления памятью MMU.

При обращении по чтению запрос из очереди необслуженных запросов в MMU удаляется после получения данных из блока памяти, передачи этих данных ждущему треду и перевода ждущего треда в состояние готовых к исполнению.

В случае обращения к удаленному блоку памяти по записи формируется обращение к сетевому контроллеру, который должен переслать в другой ВМ обращение соответствующему удаленному блоку памяти, удалив после этого запрос из своей очереди необслуженных запросов. Адресуемый ВМ, в котором размещен блок памяти, в который должна быть проведена запись данных, получив соответствующий адрес и сами данные по сети формирует запрос по записи в очереди необслуженных запросов MMU этого ВМ. Далее этот запрос передается в соответствующий смарт-контроллер памяти для выполнения и удаляется из очереди необслуженных запросов MMU. То есть с момента попадания запроса на запись в очередь необслуженных запросов MMU его исполнение не отличается от исполнения запроса на запись в близкий блок памяти.

Языком параллельного программирования, реализующим модель PRAM (*Parallel Random Access Model*), может служить расширение языка C [21] и Cilk [22]. Для порождения и завершения асинхронных тредов в Cilk используются три дополнительные функции: cilk_spawn, cilk_for, cilk_sync. Эти функции позволяют, соответственно:

- породить новый тред, исполняющий код, непосредственно следующий за cilk_spawn;

- выполнить порождение в цикле for совокупности тредов;
- завершить исполняющий тред и перейти к следующему оператору, если завершены все порожденные соответствующей функцией cilk_spawn треды.

При порождении тредов создается контекст треда, достаточный чтобы выполнить последующее завершение всех порожденных тредов. Собственно этот контекст используется в смарт-контроллере при выполнении соответствующей функции cilk_sync. Контекст треда, наряду с требуемыми ресурсами, обязательно включает адрес слова разделяемой памяти, в котором хранится число завершенных тредов. В контексте также есть переменная, содержащая число порожденных тредов. Слово памяти, в котором хранится число завершенных тредов, обнуляется при порождении тредов и увеличивается при достижении тредом завершения cilk_sync. При достижении значения, равного числу порожденных тредов, выполняется переход к следующему за cilk_sync оператору.

Если тред порождает тред или совокупность тредов, то каждый из них наследует контекст порождающего треда, расширенный указателями на этот тред и рядом параметров. Естественно, все порожденные треды должны быть завершены до порождающего. У порожденных тредов в контексте используется собственное слово для подсчета завершенных тредов. Уровень вложенности тредов ограничивается принятой структурой контекста.

Следует отметить, что в Cray XMT [23] расширение языка C для использования синхронизации на базе FE-битов реализовано как введение синхронизирующих переменных x$ и аппаратных функций (*generic functions*) для выполнения операций чтения и записи. Среди них отметим:

- purge x$ — присвоение FE-биту x$ значения empty;
- writeqr x$, *g* — запись в x$ значения *g*, если значение FE-бита x$ равно *q*, или ожидание записи, пока значение FE-бита не станет *q*; после записи значение FE-бита становится равным *r*;
- readqr x$ — чтение значения x$, если значение FE-бита x$ равно *q*, или ожидание чтения пока значение FE-бита не станет *q*; после чтения значение FE-бита становится равным *r*.

Смарт-контроллер памяти для каждого запроса независимо может работать в обычном режиме или в расширенном режиме с учетом механизма FE-битов.

Смарт-контроллер содержит память FE-битов слов памяти блоков, к которым он обеспечивает доступ. Размещение FE-битов слов в памяти, отдельной от самого блока памяти, имеет ряд преимуществ. Во-первых, позволяет исключить обращения к памяти при несоответствующем требуемому состоянии FE-бита слова памяти, во-вторых, позволяет выполнять операции с FE-битами без реального обращения к блокам памяти, и, в-третьих, позволяет использовать традиционный подход к построению блоков памяти и функционированию без использования FE-битов.

В расширенном режиме смарт-контроллер реализует задаваемую модель PRAM: PRAM-EREW (*exclusive-read exclusive-write*); PRAM-CREW (*concurrent-read exclusive-write*); PRAM-CRCW (*concurrent-read concurrent-write*) [24]. Однако все они, кроме PRAM-EREW требуют достижения обращения к слову блока памяти всеми порожденными тредами, что может повлечь неприемлемую потерю производительности (неявное введение барьера). Вопрос использования двух других моделей требует дополнительного исследования.

Если FE-бит не имеет требуемого значения, то обращение к памяти помещается в очередь ожидания. Элементы этой очереди содержат собственно обращение к памяти (команду чтения/записи, адрес слова, значения FE-битов до выполнения обращения и по его завершении, служебную информацию для работы в расширенном режиме, а также указатель на незавершенную команду MMU). Изменение состояния FE-бита слова инициирует обработку очереди ожидания с реализацией требуемой операции.

Если FE-бит имеет требуемое значение, то контроллер блока памяти выполняет требуемое чтение или запись и формирует заданное значение FE-бита.

Не вдаваясь в детали реализации контроллера памяти, отметим, что после завершения обращения к слову памяти должны выполняться поиск в очереди ожидания элементов, соответствующих этому слову памяти, и проверка возможности выполнения соответствующего обращения к памяти. Если таковое возможно, то обращение пускается на выполнение и соответствующий элемент удаляется из очереди. Естественно, описанные действия для своего ускорения требуют ассоциативного поиска в очереди ожидания.

Атомарные операции compare-and-swap (CAS), fetch-and-add, test-and-set выполняются как операции чтения с дополнительными действиями, для реализации которых используется арифметико-логическое устройство смарт-контроллера, необходимое также для cilk_sync. Такая реализация устраняет проблемы с обеспечением атомарности, исключая возможность прерывания хода выполнения действий, составляющих операцию.

Это обстоятельство исключает необходимость использования вместо CAS пары load-reserved (lr) и store-conditional (sc) [25].

Распределенный характер обращений в память служит гарантией высокой производительности, пропорциональной числу используемых смарт-контроллеров и блоков памяти вычислительных модулей.

Энергоэффективная маршрутизация

При использовании порядка 10^7 и более ВМ для исполнения программ с интенсивными обменами данными коммуникационная среда превращается в существенного потребителя энергии. Это вызывается, в том числе тем обстоятельством, что программы пользователей именуют процессы, приписывая им логические номера 0, 1, ..., $n-1$, где n — число используемых ВМ, и для установления соответствия между адресацией по логическим номерам процессов и физическими адресами ВМ, в которых эти процессы протекают, используются таблицы. Объем памяти, занимаемой ими, пропорционален квадрату числа ВМ в системе, так как таблица должна быть в каждом ВМ.

В большинстве коммуникационных сред современных суперкомпьютеров пакеты маршрутизируются таблицами, размещаемыми для ускорения поиска в блоках энергозатратной ассоциативной памяти, объем и число которых также возрастают с увеличением числа ВМ. По этой причине интенсивно исследуются возможности создания арифметической маршрутизации [13].

Среди алгоритмов арифметической адресации известна координатная адресация, которая применяется, в том числе и в вычислительных системах с Л(N, v, 4)-графом [15, 16], или многомерными кубическими структурами в качестве графов межмодульных связей, в которых адрес ВМ задается координатами по каждому направлению. При передаче элемента данных сравниваются значения координат адреса назначения и ВМ, в котором выполняется сравнение. Если значения всех координат равны, адресат достигнут. Среди направлений передачи с несовпавшими координатами выбирается направление, ведущее к уменьшению рассогласования. Таким образом обеспечивается эффективная адресация. Она используется в ряде суперкомпьютеров, например, фирмы CRAY.

Для вычислительных систем с Л(N, v, g)-графом межмодульных связей, где N — число вершин; v — степени вершин; g — обхват графа, предложена Д(z, m) — адресация [16], при которой адрес ВМ_i, $i \in \{0, \ldots, n-1\}$, также задается вектором $A_i = A_{i0}, \ldots, A_{i\,m-1}$. Каждый A_{ij}, $j = 0, \ldots, m-1$, принимает значение из множества $\{0, 1, \ldots, 2^z - 1\}$, $m \in \{1, 2, \ldots\}$, $z \in \{1, 2, \ldots\}$, n — число ВМ подсистемы, $mz \geqslant \lceil \log_2 n \rceil$, $\lceil x \rceil$ — наименьшее целое такое, что $x \leqslant \lceil x \rceil$. ВМ_i и ВМ_k, $i, k \in \{0, 1, \ldots, n-1\}$, принадлежат подсистеме $R_r(A_{i0}, \ldots, A_{i\,r-1})$ яруса r, если $A_{i\,r} \neq A_{kr}$, и $A_{ij} = A_{kj}$ для всех $j < r$, $r = 1, 2, \ldots, m$. $R_0()$ — подсистема яруса 0 (вся подсистема из n ВМ), $R_m(A_{i0}, \ldots, A_{im-1})$ — подсистема яруса m, состоящая из одного ВМ_i. Адресация ВМ должна удовлетворять следующим свойствам:

- ВМ каждой подсистемы $R_r(A_{i0}, \ldots, A_{ir-1})$ яруса r, $r = 1, 2, \ldots, m$, должны вместе с линиями связи между ними отображаться в связный подграф графа межмодульных связей;
- $R_0 \supseteq R_1(A_{i0}) \supseteq R_2(A_{i0}, A_{i1}) \supseteq \ldots \supseteq R_m(A_{i0}, \ldots, A_{im-1})$;
- $R_r(A_{i0}, \ldots, A_{ir-1}) = \bigcup_{j=0}^{d} R_{r+1}(A_{i0}, \ldots, A_{ir-1}, j)$,

$d = 2^z - 1$;

- $R_{r+1}(A_{i0}, \ldots, A_{ir-1}, j) \cap R_{r+1}(A_{i0}, \ldots, A_{ir-1}, k) = \varnothing$, $j \neq k$.

При Д(z, m)-адресации в каждом ВМ_i, $i \in \{0,1, \ldots, n-1\}$, должно храниться в путевой таблице только $m2^z$ номеров $p_1(0)$, $p_1(1)$, ..., $p_1(2^z-1)$, $p_2(0)$, $p_2(1)$, ..., $p_2(2^z-1)$, ..., $p_m(0)$, $p_m(1)$, ..., $p_m(2^z-1)$ выходных полюсов ВМ_i, принадлежащих кратчайшим путям из ВМ_i в соответственно подсистемы $R_1(0)$, $R_1(1)$, ..., $R_1(2^z-1)$, $R_2(A_{i0}, 0)$, ..., $R_2(A_{i0}, 2^z-1)$, ..., $R_m(A_{i0}, \ldots, A_{im-2}, 2^z-1)$. Кратчайший путь до подсистемы — это путь до ближайшего ВМ этой подсистемы.

Алгоритм маршрутизации сравнивает адресата пакета A_k с адресом ВМ, в котором он находится, и определяет старший, начиная с 0, не совпадающий разряд j, $j \in \{0, 1, \ldots, m-1\}$. Далее из путевой таблицы берется значение, соответствующее A_{kj}, которое задает номер линии связи для передачи пакета. Если адресат пакета и адрес ВМ, в котором он находится, совпадают, пакет доставлен. Как видно, отличие от координатной адресации заключается в том, что ищется старший не совпадающий разряд, а не любой. Кроме того, необходимо нахождение в путевой таблице номера линии связи, а не прямое его определение номером разряда рассогласования.

Представленная маршрутизация на основе Д(z, m)-адресации требует при программировании таблицы соответствия номеров и адресов в каждом ВМ, так как программирование ведется в терминах номеров процессов, а передача пакетов сообщений между процессами выполняется на основании адресов ВМ. Представим алгоритм маршрутизации, в котором используется Д(1, m)-

адресация с адресами, задаваемыми номерами ВМ в двоичном представлении m разрядами.

Создадим Д(1, m)-адресацию на выделенной связной подсистеме из N ВМ, значение m определится в ходе построения адресации. Итак, будем полагать, что есть алгоритм, строящий связную подсистему из виртуальных ВМ, с производительностью выше заданного порога и линиями связи между ВМ, с пропускной способностью выше заданного порога, что определяется соглашением о качестве облуживания.

Пусть имеем граф с N вершинами и ребрами, помеченными их пропускной способностью. Находим для этого графа паросочетание вершин (*matching*) с максимальной суммой меток выбранных ребер [23]. Пусть для графа на рис. 1, $N = 10$, получится паросочетание, приведенное на рис. 2. Вершины, инцидентные выбранным ребрам, соответствуют двухвершинным подсистемам нижнего уровня, а каждая вершина, не инцидентная выбранным ребрам, образует одновершинную подсистему нижнего уровня. В целом выбор делается так, чтобы получить максимальную суммарную пропускную способность линий связи подсистем.

Далее двухвершинные подсистемы стягиваются в одну вершину с сохранением инцидентных им ребер (рис. 3).

Находим для этого графа паросочетание вершин (*matching*) с максимальной суммой выбранных ребер. И повторяем последовательность этих действий, пока не получим двухвершинный граф. Соответствующие шаги алгоритма иллюстрируются на рис. 4—6.

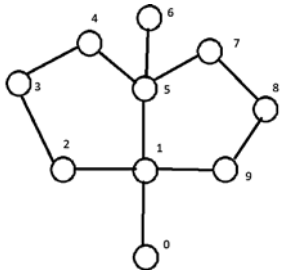

Рис. 1. Граф межмодульных связей выделенной подсистемы

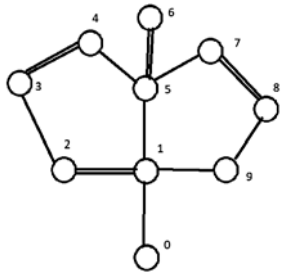

Рис. 2. Сформированные паросочетания на шаге 1 алгоритма

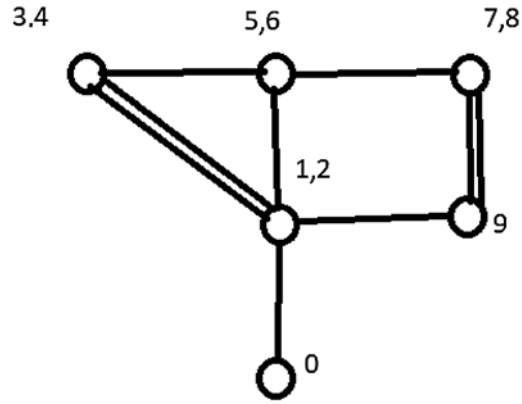

Рис. 3. Сформированные паросочетания на шаге 2 алгоритма

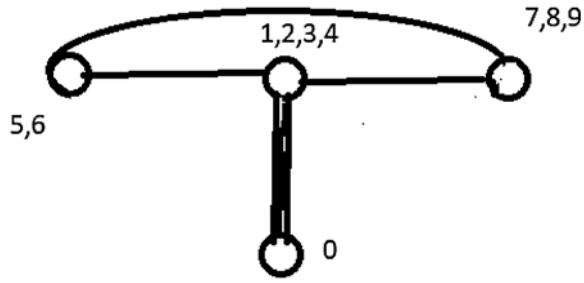

Рис. 4. Сформированные паросочетания на шаге 3 алгоритма

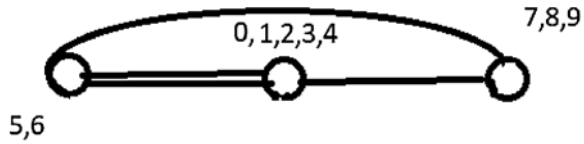

Рис. 5. Сформированные паросочетания на шаге 4 алгоритма

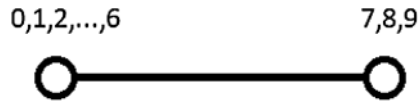

Рис. 6. Сформированные паросочетания на шаге 5 алгоритма

Число сделанных шагов определяет параметр m. В рассматриваемом примере их 5. Таким образом создана Д(1, 5)-адресация, приведенная на рис. 7.

При задании адресов одноэлементному подмножеству всегда приписывается 0, что обеспечивает обязательное присутствие номера 0 с адресом 0...0 с нулями во всех m разрядах адреса.

Далее в каждом ВМ формируются путевые таблицы $T[i]$, $i = 0, ..., m - 1$. По завершении элемент $T[i]$, $i = 0, ..., m - 1$:

1) если его значение больше 0, то задает номер связи, по которой пакет должен быть передан из ВМ, чей адрес отличается в разряде i и совпадает во всех предшествующих 0, 1, ..., $i - 1$ разрядах с адресатом пакета;

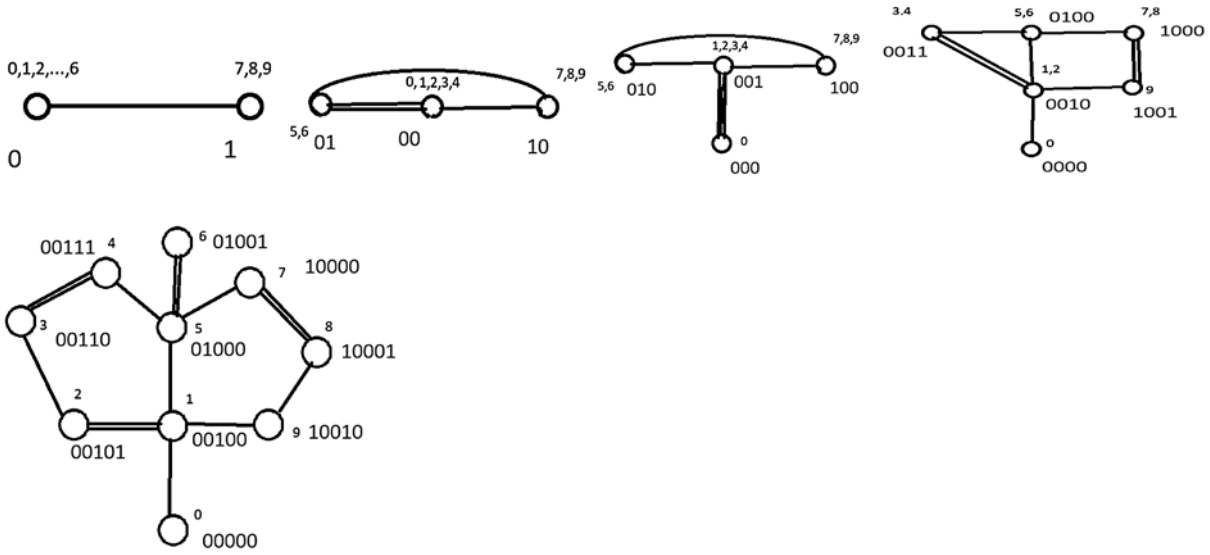

Рис. 7. Сформированная Д(1, 5)-адресация

2) если имеет ненулевое отрицательное значение, то его модуль задает адрес перенумерованного ВМ, и должен быть записан в пакет, как адресат пакета, после чего пакет заново маршрутизируется;

3) если имеет нулевое отрицательное значение, то задает отсутствие такого адреса.

Формирование путевых таблиц начинается с заполнения их нулевыми отрицательными значениями. Это необходимо для выделения несуществующих адресов.

Также формируются вспомогательные таблицы $H[i, j]$, $i = 0, ..., m - 1$, $j = 0, ..., v - 1$, содержащие минимальное расстояние (число передач) до соответствующей адресной подсистемы, исходя из ВМ по связи $j + 1$. Исходно — максимально возможное расстояние.

Во всех ВМ генерируются сообщения с полями:

- номер ВМ;
- адрес ВМ;
- количество передач (исходно равно 1);
- максимальное количество передач;
- номер связи, по которой пришло (исходно равен 0).

Сообщение посылается по всем линиям связи, кроме той, по которой поступило в ВМ. Если сообщение сгенерировано в ВМ, номер связи 0, сообщение посылается по всем линиям связи.

Получив сообщение, ВМ сравнивает свой адрес с адресом в сообщении. Находится старший разряд i, $i \in \{0, ..., m - 1\}$, несовпадения. Сравнивается значение в таблице $H[i, j - 1]$, где j равно номеру связи, $j \in \{1, 2, ..., v\}$, по которой сообщение пришло в ВМ, и число передач сообщения. Если число передач меньше значения в таблице $H[i, j - 1]$, то в $H[i, j - 1]$ заносится число передач из сообщения. Тем самым сколько бы сообщений не прошло, даже одно, будет сформирован участок маршрута к адресной подсистеме $A_0...A_i$, $i \in \{0, ..., m - 1\}$. Если пройдут все сообщения, то $H[i, j - 1]$ будет содержать длину кратчайшего маршрута из рассматриваемого ВМ до адресной подсистемы $A_0...A_i$, $i \in \{0, ..., m - 1\}$, включающего связь с номером j. Если такого маршрута не окажется, то его длина будет максимальной, т. е. исходно заданной.

Кроме того, адрес в сообщении сравнивается с n — числом ВМ в подсистеме. Если адрес больше $n - 1$, то он запоминается в списке перенумерумых ВМ. В каждом ВМ список сортируется по возрастанию, таким образом, все ВМ сформируют одинаковые списки и определится число перенумеруемых ВМ. Так, для адресации, показанной на рис. 7, в список попадут ВМ с адресами 10000, 10001, 10010.

Если не превышено максимальное число передач сообщения, оно транслируется далее по всем линиям связи, кроме той, по которой поступило в ВМ.

После завершения формирования вспомогательных таблиц $H[i, j]$, $i = 0, ..., m - 1$, $j = 0, ..., v - 1$, создаются путевые таблицы $T[i]$, $i = 0, ..., m - 1$. В каждом ВМ выбирается в каждой строке i, $i = 0, ..., m - 1$, столбец таблицы $H[i, j]$, $i = 0, ..., m - 1$, $j = 0, ..., v - 1$, с наименьшим, не равным максимальному значению расстоянием до адресуемой подсистемы, и соответствующий номер столбца становится значением $T[i]$, $i = 0, ..., m - 1$. Если вся строка i, $i = 0, ..., m - 1$, таблицы имеет исходные максимальные значения, то $T[i]$, получает значение -0.

В каждом ВМ становится возможным определить номера ВМ, адреса которых будут не совпа-

дать с их номерами. Если адрес ВМ меньше n, то анализируются элементы путевой таблицы. Если обнаруживается $T[i]$, $i \in \{0, ..., m-1\}$, равный -0, то номера, начинающиеся на $A_0A_1...A_{i-1}$, кроме собственно номера ВМ, определяются как отсутствующие. Так, для адресации, показанной на рис. 7, в ВМ с адресом 00000 $T[3]$ и $T[4]$ равны -0, что обнаруживает отсутствие номеров 00001, 00010, 00011.

Далее каждый ВМ по восходящему и нисходящему корневым деревьям с корнем 0 ... 0 передает обнаруженные отсутствующие номера и принимает эти номера, сортируя их по возрастанию. Так как число таких номеров известно, то процесс в каждом ВМ завершается их принятием. Тем самым в каждом ВМ формируется соответствие отсутствующих номеров и перенумерумых ВМ. Для адресации, показанной на рис. 7, 00001—10000, 00010—10001, 00011—10010.

Формирование адресации завершается в каждом ВМ выделением соответствующего ему подсписка переадресации отсутствующих адресов и сопоставленных перенумерумых ВМ.

Алгоритм маршрутизации модифицируется. Если обнаруживается $T[i]$, $i \in \{0, ..., m-1\}$, равный -0, то из подсписка переадресации отсутствующих адресов выбирается номер, сопоставленный адресату сообщения. Далее этот номер заносится в сообщение в качестве его адресата. Полученное сообщение подвергается маршрутизации для достижения перенумерованного ВМ.

Представленные алгоритмы формирования адресации и маршрутизации гарантируют пользователю предоставление для выполнения параллельной программы ВМ с номерами 0, 1, ..., $n-1$. Если автоматическая переадресация в алгоритме маршрутизации признается пользователем излишней, он может проводить подмену адресов в своей программе, используя данные из соответствующих таблиц.

Для обеспечения отказоустойчивости и обхода перегруженных участков маршрута алгоритм маршрутизации может использовать данные таблиц $H[i, j]$, $i = 0, ..., m-1$, $j = 0, ..., v-1$, содержащие сведения об альтернативных маршрутах до адресуемых подсистем.

Следует отметить, что на одном и том же графе связей подсистемы, в зависимости от выбранных паросочетаний вершин, может быть сформирована адресация с меньшим числом уровней. Так, для подсистемы, изображенной на рис. 1, может быть сформирована адресация, показанная на рис. 8.

Примечательно, что при меньшем значении $m = 4$ понадобится только одна переадресация: адрес 0101 заменяется на 1010. Эта адресация получается, если на шаге 1 выбраны паросочетания (0, 1), (3, 4), (5, 6), (7, 8), на шаге 2 — ((3, 4), 2), ((0, 1), (5, 6)), ((7, 8), 9), на шаге 3 — (((0, 1), (5, 6)), ((3, 4), 2))) и, наконец, на шаге 4 — ((((3, 4), 2), ((0, 1), (5, 6))), ((7, 8), 9)).

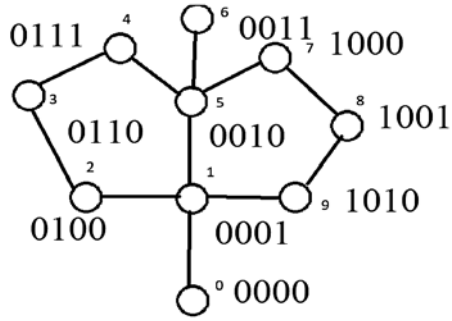

Рис. 8. Сформированная Д(1, 4)-адресация

Формирование подсистем с заданной адресацией

Рассмотренный алгоритм маршрутизации вписывается в архитектуру и организацию функционирования суперкомпьютера с разделяемой памятью с синхронизацией на базе FE-битов слов памяти [15—18]. Исходно весь суперкомпьютер рассматривается как совокупность всех ресурсов [26]. Для исполнения каждой программы выделяется подсистема, состоящая из затребованного числа виртуальных ВМ, соединяющих их линий и других ресурсов. Кратко, со ссылкой на более подробное описание, представим децентрализованный распределенный параллельный алгоритм построения подсистем, несколько модифицированный по сравнению с представленным в работе [16].

Построение подсистемы в суперкомпьютере с Л(N, v, g)-графом межмашинных связей инициируется из любого ВМ. В ВМ формируется сообщение с полями:

- тип сообщения "распространение";
- число n ВМ, которое требуется включить в строящуюся подсистему этим сообщением; $n \in \{1, ..., N\}$;
- тег подсистемы;
- соглашение о качестве обслуживания;
- номер линии связи, по которой сообщение поступило, изначально устанавливается 0.

Будем полагать, что в каждом ВМ имеется локальная очередь для сообщений с тегом подсистемы, помещаемых в нее по поступлении по линии связи или из самого ВМ, с внесением в поле "номер линии связи" номера линии связи, по которой сообщение поступило. Кроме того, имеется совокупность локальных переменных ВМ, определенных далее по мере использования.

Итак, в ВМ запускается программа построения подсистемы, в очереди которой находится сформированное сообщение типа "распространение".

По получении сообщения типа "распространение" определяется, порожден или нет виртуальный ВМ подсистемы с тегом сообщения. Если не порожден, то выполняются действия по порождению.

Если при обработке сообщения типа "распространение" локальная проверка устанавливает возможность выполнения соглашения о качестве обслуживания, то порождается виртуальный ВМ с ресурсами, выделенными по соглашению о качестве обслуживания. Номер линии связи, по которой сообщение поступило, записывается в локальную переменную ВМ up (путь к $ВМ_0$). Если номер линии связи, по которой сообщение поступило, равен 0, то ВМ получает в этой подсистеме локальный номер 0. В иных случаях номер — максимально возможное число. Поле "количество ВМ" уменьшается на 1, его значением становится $n - 1$. В порожденном виртуальном ВМ формируется локальный вектор q_p, (q_p исходно равен 1), $p = 1, ..., v$, где p — номер линии связи. Если номер s линии связи, по которой сообщение поступило, больше 0, то $q_s \leftarrow 0$.

Если после выполнения перечисленных выше действий по порождению ВМ поле "количество ВМ" имеет значение $n = 0$, то тип сообщения "распространение" заменяется на "завершение". В поле "количество ВМ" заносится 1. Количество ВМ поддерева, корнем которого он служит [15, 16], устанавливается равным 1 в локальной переменной τ, $\tau \leftarrow 1$. Сообщение типа "завершение" отправляется по линии связи, по которой сообщение поступило (в случае 0 в ВМ, инициировавший построение подсистемы).

Если $n - 1 \geqslant 1$, то τ получает значение n, $\tau \leftarrow n$, и $n - 1$ делится между открытыми линиями связи этого ВМ с $q_p = 1$, $p = 1, ..., v$. Для передачи по каждой открытой линии связи на основе полученного формируются сообщения типа "распространение", в которых указано выделенное линии количество ВМ. Это количество ВМ также суммируется в q_p, где p — номер очередной рассматриваемой открытой линии связи.

Если ВМ уже есть, и пришло сообщение типа "распространение" на продолжение построения подсистемы из этого ВМ в силу того, что где-то не включено в подсистему запрошенное количество ВМ, то анализируется q_p, $p = 1, ..., v$, этого ВМ. Если все значения $q_p = 0$, $p = 1, ..., v$, то тип сообщения меняется на "тупик", и оно передается в ВМ из которого поступило. Если хотя бы одно q_p, $p = 1, ..., v$, этого ВМ больше 1, то обработка сообщения откладывается до момента, пока все q_p, $p = 1, ..., v$, не получат значения 1 или 0. То есть в ВМ ожидается ответ на все посланные сообщения типа "распространение". По получении последнего ответа и формировании актуального вектора q_p, $p = 1, ..., v$, этого ВМ локальная переменная $\tau \leftarrow \tau + n$. Количество n делится между открытыми линиями связи этого ВМ с $q_p = 1$, $p = 1, ..., v$. Для передачи по каждой открытой линии связи на основе полученного формируются сообщения типа "распространение", в которых указано выделенное линии количество ВМ. Это количество ВМ также суммируется в q_p, p — номер рассматриваемой линии связи.

Если устанавливается отсутствие возможности выполнения соглашения, то формируется сообщение с типом "ресурсный отказ", отправляемое по линии связи, по которой сообщение поступило (в случае 0 ВМ, инициировавший построение подсистемы, получает отказ).

При поступлении в ВМ сообщения типа "ресурсный отказ" номер s линии, по которой оно поступило, определяет $q_s \leftarrow q_s - n$, переменная $\tau \leftarrow \tau - n$, где n — значение поля "количество ВМ" сообщения. Если $q_s = 1$, то q_s обнуляется $q_s \leftarrow 0$. Тем самым линия связи закрывается. Тип сообщения меняется на "распространение".

При поступлении в ВМ сообщения типа "завершение" номер s линии, по которой оно поступило, определяет $q_s \leftarrow q_s - n$, где n — значение поля "количество ВМ" сообщения. Если все q_p, $p = 1, ..., v$, имеют после этого значения 1 или 0, то получены подтверждения на включение в подсистему запрошенного количества ВМ. Анализируется значение локальной переменной up. Если up не равно 0, то создается сообщение типа "завершение", в котором поле "количество ВМ" равно локальной переменной τ, то сообщение передается по линии, определяемой локальной переменной up.

Если значение up равно 0, то достигнут корневой $ВМ_0$. Построение подсистемы завершено. В каждом ВМ сформированы локальные переменные, необходимые для задания номеров ВМ и маршрутизации по восходящему и нисходящему корневым деревьям [15, 16]:

- локальная переменная τ задает количество ВМ корневого покрывающего поддерева, корнем которого служит этот ВМ, подсистемы с корнем $ВМ_0$;
- локальная переменная up служит направлением к корню;
- компоненты локального вектора q_p, $p = 1, ..., v$, равные 1, задают направления от корня.

При инициации суперкомпьютера строится подсистема, включающая все ВМ, на которых задается Д(1, m)-адресация. Номер ВМ служит префиксом адресов памяти блоков памяти, размещен-

ных в этом ВМ. Длина префикса $m \geqslant \lceil \log_2 N \rceil$. При использовании N порядка 10^7, $m \geqslant 24$.

Таким образом, адрес разделяемой памяти суперкомпьютера состоит, по крайней мере, из трех полей: номер ВМ, номер блока памяти ВМ, адрес в блоке памяти. При этом номер ВМ определяется в ходе формирования адресации.

Построение Д(1, m)-адресации выполняется на ВМ подсистемы децентрализованным распределенным параллельным алгоритмом с использованием алгоритма нахождения паросочетаний [23] в режиме эмуляции графа межмодульных связей.

Заключение

Представленная в статье организация работы с распределенной разделяемой памятью направлена на поддержку повышения производительности параллельного программирования и снижение сложности и энергозатратности исполнения программ. Для исполнения параллельной программы формируется подсистема из виртуальных вычислительных модулей с номерами в интервале 0, ..., $n - 1$, где n — число запрошенных программой ВМ. Для синхронизации легких тредов программ используются FE-биты слов разделяемой памяти. Смарт-контроллеры содержат отдельную память FE-битов слов блоков памяти и для каждого запроса независимо могут работать в обычном режиме или в расширенном режиме с учетом механизма FE-битов.

Предложенная арифметическая маршрутизация при использовании топологии с минимальным средним расстоянием устраняет взаимовлияние параллельных программ, исполняемых в разных подсистемах, и снижает энергозатратность коммутаций в целом.

Список литературы

1. **Kogge P. M.** Graph Analytics: Complexity, Scalability, and Architectures // 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017. P. 1039—1047. DOI: 10.1109/IPDPSW.2017.176.

2. **Harrod W.** Advanced Graphical Intelligence Logical Computing Environment (AGILE) // ISC 2022 IXPUG Workshop. June 2, 2022. Hamburg Germany. URL: https://www.iarpa.gov/images/PropsersDayPDFs/AGILE/AGILE_-_ISC_2022_Harrod_Updated.pdf

3. **Song W. S., Gleyzer V., Lomakin A., Kepner J.** Novel graph processor architecture, prototype system, and results // 2016 IEEE High Performance Extreme Computing Conference (HPEC), 2016. P. 1—7.

4. **Song W. S.** Processor for large graph algorithm computations and matrix operations. United States Patent US 8751556B2.

5. **Kepner J., Aaltonen P., Bader D.** et al. Mathematical Foundations of the GraphBLAS. arXiv:1606.05790v2.

6. **Dysart T., Kogge P. M., Deneroff M.** et al. Highly Scalable Near Memory Processing with Migrating Threads on the Emu System Architecture // 6th Workshop on Irregular Applications: Architecture and Algorithms (IA3). 2016. P. 2—9. DOI: 10.1109/IA3.2016.007.

7. **Aananthakrishnan S., Ahmed N. K., Cave V.** et al. PIUMA: programmable integrated unified memory architecture. arXiv:2010.06277.

8. **Belviranli M. E., Lee S., Vetter J. S.** Designing Algorithms for the EMU Migrating-threads-based Architecture // 2018 IEEE High Performance Extreme Computing Conference (HPEC), 2018. P. 1—7. DOI: 10.1109/HPEC.2018.8547571.

9. **The 59th** edition of the TOP500. URL: https://www.top500.org/lists/top500/2022/06/highs/

10. **GREEN500** LIST — JUNE 2022. URL: https://www.top500.org/lists/green500/list/2022/06/

11. **De Sensi D., Di Girolamo S., McMahon K. H.** et al. An In-Depth Analysis of the Slingshot Interconnect. arXiv:2008.08886v1 [cs.DC] 20 Aug 2020.

12. **Vigneras P., Quintin J. N.** The BXI routing architecture for exascale supercomputer // The Journal of Supercomputing. 2016. Vol. 72, No. 12. P. 4418—4437. DOI: 10.1007/s11227-016-1755-2.

13. **Concatto C., Pascual J. A., Navaridas J.** et al. A CAM-Free Exascalable HPC Router for Low-Energy Communications // Springer International Publishing AG, part of Springer Nature 2018/ M. Berekovic et al. (Eds.). ARCS 2018. LNCS 10793. P. 99—111. DOI: 10.1007/978-3-319-77610-1_8.

14. **Deng Y., Guo M., Ramos A. F., Huang X., Xu Zh., Liu W.** Optimal Low-Latency Network Topologies for Cluster Performance Enhancement. arXiv:1904.00513v1 [cs.NI].

15. **Корнеев В. В.** Параллельное программирование // Программная инженерия. 2022. Том 13, № 1. С. 3—16. DOI: 10.17587/prin.13.3-16.

16. **Корнеев В. В.** Архитектура вычислительных систем с программируемой структурой. Новосибирск: Наука, 1985. 166 с.

17. **Корнеев В. В.** Модель программирования и архитектура экзафлопсного суперкомпьютера // Открытые системы. СУБД. 2014. № 10. С. 20—22.

18. **Елизаров С. Г., Лукьянченко Г. А., Корнеев В. В.** Технология параллельного программирования экзафлопсных компьютеров // Программная инженерия. 2015. № 7. С. 3—10.

19. **Hoffmann H., Wentzlaff D., Agarwal A.** Remote Store Programming: A Memory Model for Embedded Multicore // Proceedings of the 5th international conference on High Performance Embedded Architectures and Compilers, HiPEAC 2010, Jan 2010. P. 3—17. DOI: 10.1007/978-3-642-11515-8_3.

20. **Davidson S., Taylor M. B., Dreslinski R. G.** et al. The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips // IEEE Micro March/April 2018. Vol. 38, No. 2. P. 30-41. DOI: 10.1109/MM.2018.022071133.

21. **Wen X., Vishkin U.** FPGA-Based Prototype of a PRAM-On-Chip Processor // Proceedings of the 5th conference on Computing frontiers. New York, NY, USA: ACM, 2008. P. 55—66. DOI: 10.1145/1366230.1366240.

22. **Blumofe R. D., Joerg C. F., Kuszmaul B. C.** et al. Cilk: an efcient multithreaded runtime system // Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming, ser. PPOPP'95. New York, NY, USA: ACM, 1995. P. 207—216. URL: DOI: 10.1145/209936.209958.

23. **Feo J.** Dataflow on Cray XMT. URL: https://www.youtube.com/watch?v=5nj1Ql0Eo5k&t=1266s

24. **Akl S. G.** Design and analysis of parallel algorithms. Prentice Hall, 1989. 412 p.

25. **Patterson D., Hennessy J. L.** Computer Organization and Design RISC-V Edition: The Hardware Software Interface. Morgan Kaufmann, 2017. 696 p.

26. **Николаев Д. С., Корнеев В. В.** Использование механизмов контейнерной виртуализации в высокопроизводительных вычислительных комплексах с системой планирования заданий SLURM // Программная инженерия. 2017. Том 8, № 4. С. 147—160. DOI: 10.17587/prin.8.147-160.

Routing in a Communication Fabric of a Computing System with Distributed Shared Memory and Synchronization based on FE¯Bits

V. V. Korneev, korv@rdi-kvant.ru,
FSUE R&D Institute "Kvant", Moscow, 125438, Russian Federation

Corresponding author:

Victor V. Korneev, Principal Researcher,
FSUE R&D Institute "Kvant", Moscow, 125438, Russian Federation
E-mail: korv@rdi-kvant.ru

Received on September 13, 2022
Accepted on September 29, 2022

This article discusses an architecture based on the paradigm of using all possible processes parallelism. The user should only specify which calculations can be performed in parallel threads over shared memory, conforming only to the selected algorithm. This allows you to create the maximum flow of memory accesses inherent in the algorithm. If necessary, read, and only then write a new value instead to the corresponding shared memory cell, the user believes that the conflict resolution mechanism is implemented by hardware memory access control. In general, the proposed architecture is aimed at solving the same problems as the EMU and PIUMA architectures, but uses "smart" controllers of shared memory blocks to synchronize threads and implement atomic operations.

For a large flow of accesses to distributed shared memory, energy-efficient routing is necessary. This paper proposes arithmetic routing, which is applicable in any communication fabrics, including with graphs of Dragonfly and graphs with the minimum possible length of the middle path and with the same number of vertices N and degrees of vertices v. An addressing and routing algorithm is proposed that provides energy-efficient access to distributed shared memory. Routing enables fault-tolerant operation based on the choice of alternative routes.

Keywords: *architecture of a computing system with distributed shared memory, parallel programming model, coordinate addressing, energy-efficient routing, arithmetic routing*

For citation:

Korneev V. V. Routing in a Communication Fabric of a Computing System with Distributed Shared Memory and Synchronization based on FE-Bits, *Programmnaya Ingeneria,* 2022, vol. 13, no. 10, pp. 471—482.

DOI: 10.17587/prin.13.471-482

References

1. **Kogge P. M.** Graph Analytics: Complexity, Scalability, and Architectures, *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW),* 2017, pp. 1039—1047. DOI: 10.1109/IPDPSW.2017.176.

2. **Harrod W.** Advanced Graphical Intelligence Logical Computing Environment (AGILE). ISC 2022 IXPUG Workshop. June 2, 2022. Hamburg Germany, available at: https://www.iarpa.gov/images/PropsersDayPDFs/AGILE/AGILE_-_ISC_2022_Harrod_Updated.pdf

3. **Song W. S., Gleyzer V., Lomakin A., Kepner J.** Novel graph processor architecture, prototype system, and results, *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, 2016, pp. 1—7.

4. **Song W. S.** Processor for large graph algorthm computations and matrix operations. United States Patent US 8751556B2.

5. **Kepner J., Aaltonen P., Bader D.** et al. Mathematical Foundations of the GraphBLAS. arXiv:1606.05790v2.

6. **Dysart T., Kogge P. M., Deneroff M.** et al. Highly Scalable Near Memory Processing with Migrating Threads on the Emu System Architecture, *6th Workshop on Irregular Applications: Architecture and Algorithms (IA3),* 2016, pp. 2—9. DOI: 10.1109/IA3.2016.007.

7. **Aananthakrishnan S., Ahmed N. K., Cave V.** et al. PIUMA: programmable integrated unified memory architecture. arXiv:2010.06277.

8. **Belviranli M. E., Lee S., Vetter J. S.** Designing Algorithms for the EMU Migrating-threads-based Architecture, *2018 IEEE High Performance Extreme Computing Conference (HPEC),* 2018, pp. 1—7, DOI: 10.1109/HPEC.2018.8547571.

9. **The 59th** edition of the TOP500, available at: https://www.top500.org/lists/top500/2022/06/highs/

10. **GREEN500** LIST — JUNE 2022, available at: https://www.top500.org/lists/green500/list/2022/06/

11. **De Sensi D., Di Girolamo S., McMahon K. H.** et al. An In-Depth Analysis of the Slingshot Interconnect. arXiv:2008.08886v1 [cs.DC] 20 Aug 2020.

12. **Vigneras P., Quintin J. N.** The BXI routing architecture for exascale supercomputer, *The Journal of Supercomputing,* 2016, vol. 72, no. 12. pp. 4418—4437. DOI: 10.1007/s11227-016-1755-2.

13. **Concatto C., Pascual J. A., Navaridas J.** et al. A CAM-Free Exascalable HPC Router for Low-Energy Communications, *Springer International Publishing AG, part of Springer Nature 2018/* M. Berekovic et al. (Eds.), ARCS 2018, LNCS 10793, pp. 99—111. DOI: 10.1007/978-3-319-77610-1_8.

14. **Deng Y., Guo M., Ramos A. F.** et al. Optimal Low-Latency Network Topologies for Cluster Performance Enhancement. arXiv:1904.00513v1 [cs.NI].

15. **Korneev V. V.** Parallel programming, *Programmnaya Ingeneria*, 2022, vol. 13, no. 1, pp. 3—16. DOI: 10.17587/prin.13.3-16 (in Russian).

16. **Korneev V. V.** *Architecture of computer systems with a programmable structure*, Novosibirsk, Nauka, 1985, 166 p. (in Russian).

17. **Korneev V. V.** Programming model and architecture of an exaflops supercomputer, *Otkrytye sistemy, SUBD*, 2014, no. 10, pp. 20—22 (in Russian).

18. **Elizarov S. G., Lukyanchenko G. A., Korneev V. V.** Technology of parallel programming of exaflops computers, *Programmnaya Ingeneria*, 2015, no. 7, pp. 3—10 (in Russian).

19. **Hoffmann H., Wentzlaff D., Agarwal A.** Remote Store Programming: A Memory Model for Embedded Multicore. *Proceedings of the 5th international conference on High Performance Embedded Architectures and Compilers, HiPEAC,* 2010, Jan 2010, pp. 3—17. DOI:10.1007/978-3-642-11515-8_3.

20. **Davidson S., Taylor M. B., Dreslinski R. G.** et al. The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips, *IEEE Micro,* 2018, vol. 38, no. 2, pp. 30—41. DOI: 10.1109/MM.2018.022071133.

21. **Wen X., Vishkin U.** FPGA-Based Prototype of a PRAM-On-Chip Processor, Proceedings of the 5th conference on Computing frontiers, New York, NY, USA, ACM, 2008 pp. 55—66. DOI: 10.1145/1366230.1366240.

22. **Blumofe R. D., Joerg C. F., Kuszmaul B. C.** et al. Cilk: an efcient multithreaded runtime system, *Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming, ser. PPOPP'95,* New York, NY, USA, ACM, 1995, pp. 207—216. URL: http://doi.acm.org/10.1145/209936.209958.

23. **Feo J.** Dataflow on Cray XMT. URL: https://www.youtube.com/watch?v=5nj1Ql0Eo5k&t=1266s

24. **Akl S. G.** *Design and analysis of parallel algorithms,* Prentice Hall, 1989, 412 p.

25. **Patterson D., Hennessy J. L.** *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, Morgan Kaufmann, 2017, 696 p.

26. **Nikolaev D. S., Korneev V. V.** The use of container virtualization mechanisms in high-performance computing complexes with the SLURM task scheduling system, *Programmnaya Ingeneria*, 2017, vol. 8, no. 4, pp. 147—160. DOI: 10.17587/prin.8.147-160 (in Russian).

ИНФОРМАЦИЯ