Результаты масштабирования бенчмарка NPB UA на тысячи ядер суперкомпьютера Blue Gene/P с помощью PGAS-расширения OpenMP*

А.А. Корж

В статье рассмотрено распараллеливание бенчмарка Unstructured Adaptive из пакета NAS Parallel Benchmarks в парадигме PGAS, дополняющей парадигму OpenMP, для машин с распределенной памятью. Рассматривается реализация данной парадигмы на суперкомпьютере IBM Blue Gene/P. Приводятся результаты исследования производительности рассмотренного бенчмарка. На 2048 ядрах системы Blue Gene/P установленной в Московском Университете получены результаты, превосходящие ранее известные результаты для OpenMP-версии на машинах с общей памятью.

1. Введение

Будущая серия машин Cray Baker, разрабатываемая в рамках военной программы DARPA HPCS, ожидается на рынке во втором квартале 2010 года. Оставаясь в рамках традиционной MPP-архитектуры (десятки тысяч узлов, в каждом из которых по два процессора AMD Opteron, узлы соединены коммуникационной сетью с топологией 3D-тор) Baker будет использовать принципиально новую коммуникационную сеть с кодовым названием Gemini. Одним из существенных отличий Gemini от интерконнекта Seastar2+ машин серии Cray XT будет являться значительно более высокий темп передачи коротких сообщений (Message Rate) и аппаратная поддержка парадигмы PGAS. Все это значит, что сеть обеспечит эффективную передачу десятков миллионов сообщений в секунду. Другой проект, разрабатываемый фирмой IBM в рамках DARPA HPCS – PERCS. В рамках этого проекта планируется представить суперкомпьютер IBM Blue Waters в 2011 году, который также будет использовать процессоры со стандартной суперскалярной архитектурой POWER7, соединенные между собой коммуникационной сетью с пропускной способностью 400 Гбит/с на узел. Среди новшеств также значится аппаратная поддержка парадигмы общей памяти PGAS. Все это говорит о том, что вскоре PGAS получит большую популярность и имеет возможность потеснить парадигму MPI в высокопроизводительных вычислениях.

С другой стороны, более 99 процентов используемого на современных суперкомпьютерах кода написано с применением библиотеки MPI (на языках Fortran и C) и опыт показывает, что переходить на другие парадигмы и языки программирования пользователи не хотят, не видя существенных преимуществ той или иной парадигмы.

Целью данной работы – показать пример задачи, для которой использование парадигмы PGAS сможет дать значительные преимущества относительно реализации с использованием стандартных MPI/OpenMP. Предположительно, такую задачу следует выбирать из класса задач, имеющих нерегулярный шаблон доступа к данным, то есть мелкую гранулярность обращений, низкую пространственную и временную локализацию обращений к памяти. В частности, расчеты на нерегулярных адаптивных сетках относятся к данному классу. По этой причине автором была выбрана модельная задача UA (Unstructured Adaptive) из известного пакета бенчмарков NASA NAS Parallel Benchmark (NPB). Впервые бенчмарк NPB UA был добавлен в версию 3.1 пакета NPB с целью исследования производительности суперкомпьютеров на задачах с нерегулярным динамически изменяемым шаблоном доступа.

В бенчмарке NPB UA решается задача Дирихле уравнения теплопереноса в трехмерной кубической области на нерегулярной декартовой сетке. Источник тепла представляет собой шар, движущийся с постоянной скоростью. Самым существенным, с точки зрения шаблона доступа к памяти, является тот факт, что для решения задачи используется нерегулярная сетка. Причем каждые несколько шагов происходит адаптация сетки: на областях с большим градиентом тем-

* Работа выполнена при поддержке РФФИ, грант № 09-07-13596-офи_ц.

пературы сетка измельчается, с малым — укрупняется. Для решения задачи применяется спектральный метод конечных элементов (SEM) с применением метода конечных мортаров. Подробное описание применяемого численного метода и его преимуществ можно прочесть у авторов бенчмарка в [2].

До сих пор существовало лишь две известные реализации данного бенчмарка — последовательная версия и OpenMP-версия (NPB-OMP UA). MPI-версии, показывающей сколь-нибудь хороший результат на стандартных Infiniband-кластерах, так никому и не удалось представить. В связи с этим, разработанная автором версия, которая может работать на стратегическом суперкомпьютере IBM Blue Gene/P без аппаратной поддержки общей памяти представляет существенный интерес. С другой стороны, бенчмарк представляет собой реальную программу на Fortran77, суммарный объем кода составляет 8000 строк без учета комментариев. Поэтому, тот факт, что на распараллеливание было потрачено около двух недель, доказывает крайне высокую продуктивность программирования в стиле PGAS.

2. PGAS/OMP-версия NPB-UA

2.1 PGAS-расширение OpenMP

Модели параллельного программирования можно разделить на два класса, в зависимости от того, на коммуникациях какого типа они основаны: односторонних (обращения к удалённой памяти) либо двусторонних (передача сообщений). В двусторонних коммуникациях (используемых в библиотеке MPI версии 1) активное участие принимают две стороны: одна отправляет записываемое слово, а вторая — ждёт прихода слова, после чего копирует полученное слово из буфера приёма в нужную ячейку памяти. Адрес, куда необходимо записать слово в памяти второго узла, указывается самим получателем. В односторонних коммуникациях активное участие принимает лишь инициатор: при записи он отсылает записываемое слово, которое, достигнув по сети адресата, напрямую записывается в память второго узла (при этом процессорное время на ожидание и запись вторым узлом не тратится). Адрес, куда записать слово в памяти второго узла, указывается отправителем.

Система программирования SHMEM (от shared memory — общая память) была разработана фирмой Cray более 15 лет назад, как интерфейс односторонних коммуникаций, способный стать эффективной альтернативой и дополнением к MPI и PVM. Интерфейс SHMEM поддерживается всеми MPP-системами фирмы Cray (Cray T3E, Cray XT3/4/5/6), Silicon Graphics (SGI Altix), интерконнектами Quadrics (QsNetIII). Также библиотека SHMEM (с некоторыми дополнениями) реализована в системе МВС-Экспресс (разработана под руководством А. О. Лациса).

По сути SHMEM реализует простейший вариант программирования в стиле PGAS (Partitioned global address space). У каждого узла есть локальная память; каждому узлу также доступна удалённая память: узел может напрямую обращаться к локальной памяти любого узла системы. Поскольку обращения к удалённой памяти происходят через коммуникационную сеть, время их выполнения заметно больше, а темп — меньше, чем у обращений к локальной памяти. Ожидать выполнения каждой одиночной операции крайне дорого, поэтому требуется, чтобы программист явно выделял обращения к нелокальным ячейкам памяти.

В отличие от других PGAS-языков, например, UPC, SHMEM *заставляет* программиста явно выделять внешние обращения с помощью функций, при этом дальнейшая группировка обращений и оптимизация выполняются аппаратно. Здесь можно добавить сравнение с парадигмой общей памяти OpenMP, в которой программисту следует разрезать вычисления на части, не заботясь о распределении памяти. Но учесть различие в цене доступа к памяти NUMA систем, в особенности систем без кэш-когерентной общей памяти, данная парадигма не в силах. Именно поэтому поддержка OpenMP не смогла быть реализована эффективно на системах с распределенной памятью, хотя безуспешные попытки и предпринимались (Intel Cluster OpenMP и ScaleMP vSMP). Парадигма PGAS расширяет парадигму общей памяти OpenMP тем, что программисту надо не только распределить вычисления, но также распределить данные, а при распределении вычислений учесть то, как были распределены данные, что приводит к более эффективно работающему коду, учитывающему локальность распределения данных.

Основу SHMEM составляют две операции: `shmem_put` — запись в память удалённого узла и `shmem_get` — чтение из памяти удалённого узла. Синхронизация происходит с помощью встроенной функции `shmem_barrier_all`. Возможность напрямую обращаться к удалённой памяти даёт большинство преимуществ работы с «общей памятью», не накладывая никаких дополнительных ограничений на то, как память распределена физически.

Родной реализации SHMEM на суперкомпьютере IBM Blue Gene/P нет. Однако используемый в суперкомпьютере Blue Gene/P интерфейс DCMF [1] среднего уровня содержит функции `DCMF_Put, DCMF_Get, DCMF_Send` и другие, на основе которых довольно легко реализовать интерфейс SHMEM. Несколько нюансов содержится в реализации отвечающей парадигме SHMEM синхронизации с помощью `shmem_barrier_all`, которая не только выполняет синхронизацию процессов, но и гарантирует получение всех отправленных другими процессорами операций `shmem_put`. Однако эти вопросы остаются за рамками данной статьи.

Сравнения ради укажем, что в сделанной реализации темп выдачи сообщений `shmem_put` размером в 8 байт составляет 1 млн/с, при использовании одного ядра и 2 млн/с при использовании двух и четырех ядер на одном узле. Одной из причин такой низкой производительности является использование режима DMA, который эффективен для передачи длинных сообщений, но не оптимален в случае коротких сообщений. В частности, текущая реализация использует технику callback-ов для получения подтверждений, о том, что аппаратура DMA прочла все посланные сообщения. Исследуются возможности более эффективной реализации SHMEM с помощью интерфейса нижнего уровня SPI. Также перспективным представляется техника агрегации сообщений, которая впрочем, неприменима при экстремальных уровнях масштабирования.

Для сравнения, использование макета М3 специализированной сети, разработанной в НИЦЭВТ под руководством автора дает темп выдачи 15 млн/с, система МВС-Экспресс [3], разработанная под руководством А.О. Лациса имеет темп выдачи 12 млн/с. Однако следует учесть, что узлы Blue Gene/P имеют в 4 раза меньшую частоту, и в 8 раз меньшую производительность, чем стандартные x86-узлы, при этом доступное их количество составляет сотни-тысячи узлов.

2.2 Описание структуры программы NPB UA

С программистской точки зрения один временной шаг бенчмарка NPB UA состоит из следующих этапов: 1) продвижение конвекционной части уравнения явным методом Рунге-Кутта 4-го порядка 2) продвижение диффузионной части уравнения 3) изменение сетки, если прошло заданное число шагов. Основным по трудоемкости является продвижение диффузионной части, так как оно включает решение систем линейных уравнений с помощью метода сопряженных градиентов с предобуславливанием. Например, на процессоре PowerPC 450, используемом в узле суперкомпьютера Blue Gene/P, время счета на последовательной версии на задаче класса C распределяется следующим образом:

Адаптация сетки	0.40 %
Конвекция	24.4 %
Прочие вычисления	3.80 %
400 операций `scatter/gather`	1.90 %
Диффузия	69.5 %

Всего задача класса C содержит 4900 операций `gather/scatter` (4200 из которых содержится в продвижении диффузионной части), а это примерно 23 % времени счета.

2.3 Метод конечных мортаров и его распараллеливание

Использование метода MEM (Mortar Elements Method), описанного в [3], и перенумерация элементов согласно мортоновской нумерации каждый раз после изменения сетки значительно облегчают процесс распараллеливания. В этой вариации метода конечных элементов каждый элемент покрывается сеткой из $(N+1)^3$ точек. Из-за нерегулярности сетки, точки находящиеся на границе смежных элементов могут не совпадать друг с другом. Для решения проблемы переноса температуры смежных элементов применяется два множества точек. Первое и основное

множество точек называется точками коллокации и состоит из всех точек дискретизации каждого элемента, таким образом, их общее число составляет $N_{elt}\ (N+1)^3$.

Очевидно, что часть точек на границе элементов может содержаться в двух или трех соседних элементах, при этом это будут разные точки коллокации. Второе множество, является подмножеством точек коллокации, не содержит совпадающих точек и образует сетку. Очевидно, что точки коллокации, лежащие во внутренностях элементов идентичны точкам сетки не лежащим на границе. Отличаются же точки коллокации и точки сетки только теми точками, которые лежат на границе элементов, такие точки будем называть точками согласования. Изначально сетка содержит один элемент, совпадающий с кубической областью. В начале теста и каждый пятый шаг происходит адаптация сетки: в зависимости от градиента температуры на каждом элементе, он может быть измельчен на восемь новых кубических элементов или, наоборот, восемь кубиков одного размера могут быть объединены в один. При этом накладывается ограничение, что смежные элементы имеют либо одинаковый размер, либо отличаются размером стороны в два раза.

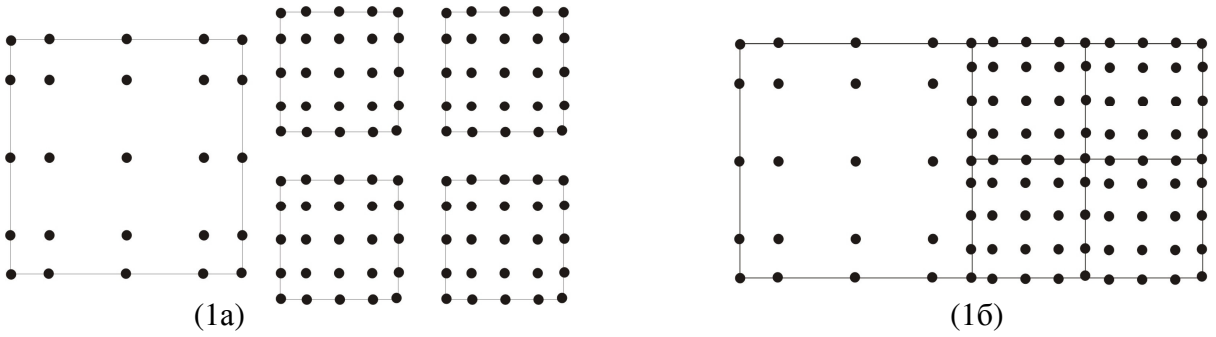


Рис. 1 Точки коллокации (1а) и точки сетки (1б) на грани элемента

Все массивы, с которыми оперирует исходная программа, разделяются на три группы:

- массивы, содержащие информацию об элементах и точках сетки
- массивы с плавающей точкой, индексированные точками коллокации (tx-массивы)
- массивы с плавающей точкой, индексированные точками согласования (tmor-массивы)

С точки зрения программиста, почти все вычисления производятся в циклах с индексом, пробегающим или множество элементов или множество точек согласования. При этом каждая итерация цикла оперирует соответственно или с точками коллокации данного элемента или с точками согласования. Поэтому, достаточно распределить точки согласования и точки коллокации между процессорами, и соответственно распределятся все итерации таких циклов, причем между итерациями циклов нет зависимости по данным. Самыми нетривиальными с точки зрения распараллеливания операциями являются операции `gather` и `scatter`. Эти операции отображают решение из точек коллокации в точки согласования и обратно, причем они выполняются на каждой итерации метода сопряженного градиента.

Для первоначальной версии было принято решение не распараллеливать процедуру адаптацию сетки. Учитывая тот факт, что она занимает 0.4 процента времени, масштабирование такого варианта будет теоретически ограничено коэффициентом 250. Также увеличивается суммарное потребление памяти, так как массивы, содержащие информацию об элементах и точках сетки, будут продублированы на каждом узле. Также, при переходе от параллельной части потребуется дополнительно выполнить коллективную операцию `all2all` для того, чтобы собрать данные по температуре с прошлой итерации на каждом узле перед адаптацией сетки, которая выполняется каждым узлом с одними и теми же данными независимо. Это также негативно отразится на масштабировании задачи.

2.4 Операция gather

Упрощенно, код, выполняемый операцией `gather`, переносящей вычисленное значение температуры с точек согласования на граничные точки коллокации, выглядит следующим образом:

```
do i=1,nelt
 do j=1,125
  tx(i,j) = a(j,1)*tmor(idmo(i,f(j,1))) + a(j,2)*tmor(idmo(i,g(j,1)))+...
 end do
end do
```

В данном фрагменте `nelt` — число элементов сетки N_{elt}, 125 = $(N+1)^3$ — число точек коллокации в каждом элементе, массив `tx` индексируется точками коллокации, массивы `a`, `f` и `g` являются константами, `idmo` - индекс вектор, индексируемый точками коллокации, массив `tmor` индексируется точками согласования.

Именно в данном цикле содержится существенная нерегулярность метода, так как соответствующие точки коллокации относящиеся к одному элементу сетки разбросаны случайным образом по массиву. Проиллюстрировать нерегулярность можно следующим графиком (см. Рис. 2), на котором изображен шаблон коммуникаций для маленького класса задачи W (число элементов — 600, число точек согласования — 30000).

На графике прекрасно видно, как шаблон нерегулярности меняется каждые 5 шагов. С другой стороны, можно отметить, что шаблон доступа к элементам не является полностью случайным: довольно большое количество обращений сконцентрировано на двух «прямых».

Отметим, что нижняя «толстая» прямая соответствует вершинам элементов, а диагональная «прямая» — точкам на гранях и ребрах элементов. Точки, которые выбиваются из этих «прямых» соответствуют граничным точкам элементов, соседних с данным. Исходя из сделанного анализа, осуществлялось распределение данных по процессорам.

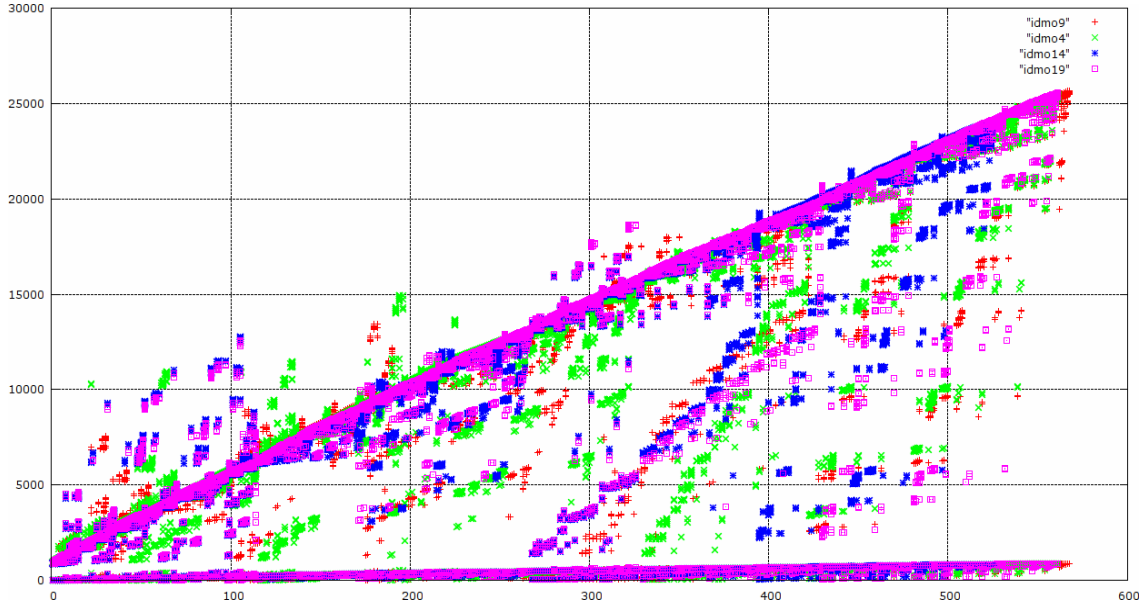


Рис. 2 – Соответствие элементов сетки (по оси абсцисс) точкам согласования (по оси ординат) в операциях Gather/Scatter в NPB UA (class W) для различных временных шагов.

2.5 Распределение данных

Выбор стратегии распределения данных имел своей целью следующее: обеспечение максимальной локализации обращений к данным, то есть уменьшение числа удаленных обращений к памяти (RMA или Remote Memory Accesses), обеспечение равного количества удаленных обращений к памяти от разных узлов для того, чтобы избежать простоя тысяч процессоров, ожидающих последнего на барьере. Причем важно не только количество исходящих обращений, но и количество входящих обращений, чтобы отдельные узлы не стали узким местом системы.

Были приняты следующие решения по распределению данных: элементы сетки и массивы, индексируемые точками коллокации, распределяются блочно по узлам распределенной системы. Таким образом, обращение к данным точек коллокации во всех циклах будет осуществляться последовательно и регулярно, как это происходит и в последовательной версии теста. Это относится к массивам `tx`, `idmo` из приведенного упрощенного кода операции `gather`.

Все нелокальные обращения происходят к массивам, индексированным точками согласования. Адреса всех таких обращений записаны в огромном индекс-векторе `idmo`, более того, все обращения к точкам сетки осуществляются только через данный индекс-вектор. Рассматривалось три варианта распределения: блочный, циклический и «двублочный».

Блочный вариант распределения точек согласования по узлам аналогичен распределению точек коллокации и является самым простым вариантом. Преимуществами данного метода является его простота реализации и некоторый учет шаблона обращений — большая часть обращений становится локальной, так как почти все обращения на диагональной линии становятся локальными. Недостатком данного распределения является значительный дисбаланс обращений от разных узлов: например, в первые узлы адресовано больше обращений, чем в другие узлы, из-за того, что первыми в нумерации точек согласования идут вершины (нижняя «толстая» прямая на графике).

Циклический вариант распределения (номер процессора определяется как остаток от деления индекса элемента на общее число процессоров) лишен недостатка дисбаланса входящих сообщений, но при этом полностью не учитывает локальность обращений — почти все обращения становятся удаленными, что увеличивает нагрузку на сеть, приводя к тому, что значительная часть времени уходит на коммуникации.

«Двублочный» вариант учитывает характер обращений к точкам согласования и особенность их нумерации, которая заключается в том, что первые N_{vertex} элементов содержат вершины всех элементов, а остальные элементы содержат точки на ребрах и гранях. Поэтому вершины и остальные точки независимо друг от друга разбиваются блочным образом, а i-узел локально хранит i-й блок вершин и i-й блок остальных точек. Этот вариант максимально сохраняет локальность обращений, но при этом сбалансирован по числу исходящих и входящих RMA.

Модификация «двублочного» варианта распределения точек согласования заключалась в том, что в i-м узле находились точки i-го блока вершин и (N-i)-го блока остальных точек, где N — число процессоров. Этот вариант в эксперименте показал лучшую производительность.

Для предоставления возможности гибкой реализации любого распределения точек согласования, в программу автором была встроена возможность выбора любого заданного пользователем распределения точек согласования. Достаточно реализовать лишь следующие функции: `redistributemor` — осуществляет вычисление констант распределения, `islocalmor` — определяет по глобальному индексу, является ли точка согласования локальной для данного процессора, `pemor` и `localmor` — определяют номер процессора и локальный индекс по глобальному индексу точки согласования.

Следует отметить, что каждые 5 шагов распределение будет изменяться, поскольку после адаптации сетки количество элементов изменится. Однако в силу гибкости кода это не составит никакой проблемы. Единственно, при распараллеливании адаптации сетки добавится несколько нелокальных обращений, так как часть элементов переедет с одних процессоров на другие. Но пока, мы решаем эту проблему с помощью сбора операцией `all2all` всех точек коллокации (точнее значения температуры на них) и выполнения адаптации сетки локально.

Следует отметить, что в таких языках, как UPC и системах с аппаратной поддержкой общей памяти (таких как Cray T3E/X1/X2/XMT) такой гибкости в распределении массивов по узлам не имеется. Как правило, аппаратно поддерживается лишь блочное, блочно-циклическое и циклическое распределение массивов по узлам, причем, как размер массива, так и шаг распределений может быть лишь степенью двойки и отсутствует возможность изменять распределение сегментов в ходе работы программы. Поэтому при реализации данной задачи придется считать распределение программно, что сделает всю это громоздкую поддержку аппаратной виртуальной сегментно-страничной памяти попросту ненужной. С другой стороны не удастся воспользоваться продуктивностью языков PGAS-класса, таких как UPC, также не обладающих гибкостью при распределении данных по узлам, что показывает достаточность таких средств как Cray SHMEM и CAF (Co-Array Fortran).

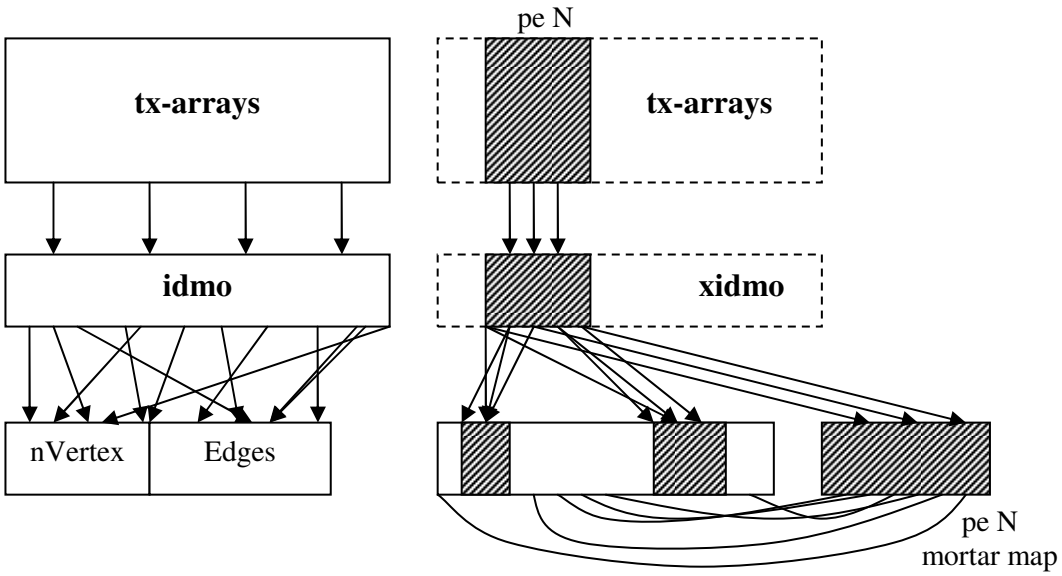
2.6. Пересчет индекс вектора

Как уже было указано, каждый процессор имеет свою локальную порцию массива `tx` и индекс-вектора `idmo`. Индекс-вектор `idmo` управляет внешними обращениями к ячейкам уда-

ленной памяти: каждая ячейка данного массива содержит индекс точки согласования, используемой при вычислении некоторого элемента массива `tx`.

Отметим, что количество элементов индекс-вектора несколько больше, чем количество элементов массива `tx`. Это отражает тот факт, что для вычисления значения температуры на точке коллокации используется не одна, а несколько соседних точек согласования. Подобное упрощение сделано лишь для простоты изложения и никак не влияет на предложенную схему распараллеливания.

Основной идеей является предлагаемый пересчет индекс-вектора с целью разделить локальные обращения и удаленные обращения, которые мы хотим выполнить перед общим циклом, реализуя парадигму DAE (Decoupled Access Execute). Для этого, каждый раз при изменении сетки, после формирования индекс-вектора (точнее той его части, что будет использоваться на локальном процессоре) мы выполняем ряд действий, которые призваны обнаружить все элементы, которые данный процессор собирается получить с других процессоров, и сохранить отсортированный список уникальных элементов в специальном массиве. Таким образом, формируется список точек согласования, находящихся на других узлах, значения которых необходимы для вычислений на локальном процессоре. Данный список будем называть картой доступа. Преимуществом нашего подхода является составление списка только из уникальных индексов, так как к каждому элементу в среднем происходит 6 обращений, а выполняя лишь одно удаленное чтение, мы значительно экономим на коммуникациях. С другой стороны, составление карты доступа приводит накладным расходам, однако следует учесть, что время подготовки карты доступа уменьшается с ростом числа узлов. К тому же, формирование карты происходит только один раз при изменении сетки, а используется она в каждой операции `gather/scatter`. Сортировка и выявление уникальных индексов выполняется с помощью отдельного кода, написанного на C++ и использующего стандартный шаблон `hash_table`.



а) б)

Рис. 3 – Доступ по индекс-вектору в последовательной версии (а) и доступ по измененному индекс-вектору к локальным и удаленным данным (1б) в PGAS-версии бенчмарка.

Далее карта доступа используется в начале каждой операции `gather`: в специальный массив `morloc` с помощью операции `shmem_get` подкачиваются значения температуры в точках согласования, находящихся на других процессорах, причем доступ к данному массиву на локальном процессоре выполняется регулярно и последовательно. После завершения функции синхронизации, гарантирующей получение всех запрошенных значений, выполняется главный цикл, который не содержит операций доступа к удаленным ячейкам памяти. С этой целью главный цикл был лишь незначительно модифицирован, все обращения к индекс вектору `idmo` были заменены на обращения к модифицированному индекс-вектору `xidmo`, который указыва-

ет на локальные элементы tmor-массива и на подкаченные с помощью построенной карты доступа элементы вспомогательного массива.

С учетом вышеизложенного, упрощенный код распараллеленной версии gather выглядит следующим образом:

```
      call shmem_barrier_all
c.....request outstanding tmor accesses to morloc array
      do i = 1,mormap_size
         pe = pemor(mormap(i))
         call shmem_double_g(morloc(i),tmor(localmor(mormap(i))),pe)
      end do
c.....gets synchronization
      call shmem_sync_gets

      do ie=startelt,endelt
         do j=1,125
          tx(i,j) = a(j,1)*readmor(xidmo(i,f(j,1)))
     &              + a(j,2)+readmor(xidmo(i,g(j,1))) + ...
       end do
      end do
```

Функция readmor заменяет все обращения к массиву tmor и содержит следующий код

```
      if(ig.gt.endmor) then
        readmor = morloc(ig-endmor)
       else
        readmor = tmor(ig)
       end if
```

Стандартным приемом программирования в стиле PGAS является замена подкачки с помощью операций shmem_get на посылку узлом-владельцем элементов с помощью операции shmem_put. На множестве реальных задач этот прием дает значительный выигрыш, так как снижает нагрузку на сеть, из-за того, что операция get посылает по сети два пакета - запрос и ответ, а операция put на многих системах реализована посылкой пакета только в одну сторону. Но для возможности такой замены каждому узлу придется составить обратную карту доступа, которая содержит номера узлов и номера ячеек в массиве morloc где ожидает данные удаленный узел. Есть два варианта составления такой обратной карты - вычислять ее полностью локально, или составить с помощью пересылок каждым узлом тех ячеек, которые содержатся в прямой карте доступа. Очевидно, что первый вариант не масштабируется, поэтому даже на небольшом числе узлов сильно проигрывает второму, который и был использован.

Таким образом, операция составления одной карты по коммуникациям эквивалентна одной операции gather. Эксперименты показали преимущество версии с put над версией с get в два раза, как и ожидалось. Другим плюсом стало увеличенная переносимость программы - для ее запуска на новой архитектуре теперь достаточно реализовать всего две функции shmem_put и shmem_barrier_all, если не считать функции инициализации (shmem_init, shmem_finalize, shmem_alloc, shmem_mype, shmem_numpes).

Альтернативой парадигме DAE при реализации операции gather могла бы стать реализация, которая прозрачно заменяет каждое чтение удаленного элемента массива операций shmem_get, однако из-за большой задержки на доступ к каждому элементу (несколько микросекунд) скорость выполнения цикла упала бы на порядок. Это было бы не так страшно, если бы вычислитель состоял из множества низкоскоростных ядер или тредов, обеспечивающих толерантность к коммуникационной задержке. Однако тренд в области HPC заключается в том, что современные суперкомпьютеры используют процессоры традиционной архитектуры с высокой производительностью одного треда, так как такие проекты, как Cray XMT не доказали своей применимости на широком классе задач, и остались узкоспециализированными системами, не применяемые в том числе и по сугубо экономическим причинам. Также, из-за отсутствия кэширования удаленной памяти в подобных nccNUMA системах, обращения к одной и той же удаленной ячейке выполнялись бы несколько раз, увеличивая расходы на коммуникации. Все это говорит о нецелесообразности аппаратной реализации nccNUMA систем, так как для реаль-

ных задач достаточно эффективной реализации иллюзии общей памяти на уровне эффективной реализации библиотеки односторонних коммуникаций, такой как Cray SHMEM.

2.7 Распараллеливание scatter

Операция `scatter` является обратной операцией к операции `gather`, и ее распараллеливание полностью аналогично, так как используется тот же индекс вектор `idmo`, а упрощенно код выглядит следующим образом:

```
do i=1,nelt
 do j=1,125
   tmor(idmo(i,f(j,1))) =   tmor(idmo(i,f(j,1))) + tx(i,j)*a(i,j)
   tmor(idmo(i,g(j,1))) =  tmor(idmo(i,g(j,1))) + tx(i,j)*b(i,j)
 end do
end do
```

Шаблон доступа scatter отличается от gather тем, что нужно выполнять изменение удаленных ячеек, а не только их чтение. Это вызвано тем, что значение температуры на одной точке согласования получается путем суммирования значений полученных в разных итерациях цикла, которые будут выполняться потенциально на разных процессорах. В OpenMP-версии такие конфликты по данным улаживаются с помощью блокировок `omp_lock` или секций `OMP_ATOMIC`. Это, в частности, одна из причин плохой масштабируемости OpenMP версии бенчмарка. В многоузловой версии также потребуются атомарное сложение с плавающей точкой, но поскольку поддержки таких операций на большинстве систем нет, придется эмулировать их через операции удаленной записи программно.

Использовалась довольно простая схема: в первой фазе каждый процессор выполняет все атомарные операции сложения с плавающей точкой, причем вместо ячеек находящихся на других узлах использует соответствующие ячейки массива `morloc`, предварительно обнуленные. После окончания всех итераций, содержимое массива `morloc` посылается с помощью операций `shmem_put` в специальный вспомогательный массив. После приема всех сообщений от других узлов, во второй фазе каждый узел выполняет локальные операции сложения полученных значений от различных узлов с соответствующими ячейками tmor-массива. Отметим, что на первой фазе используется прямая карта доступа, в то время как на второй итерации используется обратная карта. Также отметим, что количество коммуникаций сокращено в то же количество раз, что и для операции `gather`. Более того, объем коммуникаций в операциях `scatter` и `gather` абсолютно одинаковый. Различным является только направление посылок.

На Blue Gene/P можно было использовать возможность передачи активных сообщений с помощью функции `DCMF_Send`, однако это сделало бы программу непереносимой на другие системы, где такой функциональности нет. В дальнейшем планируются исследования эффективности применения `DCMF_Send` для выполнения произвольных атомарных операций.

2.8 Коллективные операции

Единственными коллективными операциями, которые потребовались при реализации бенчмарка UA, оказались `shmem_allsum_double` - суммирования по одному числу с плавающей точкой двойной точности от каждого узла, доставляя результат на каждый узел и `shmem_all2all` - для сборки вектора температуры на всех узлах перед выполнением адаптации сетки. Обе эти операции на Blue Gene/P были реализованы через специализированную сеть для коллективных операций. На других системах эти операции были реализованы через функцию `shmem_put` программно.

2.9 Гибридный режим PGAS/OpenMP

Заявленное в названии статьи расширение OpenMP заключается в том, что предложенная к использованию библиотека и парадигма программирования SHMEM подразумевает обмен ме-

жду различными узлами системы, а параллелизм же внутри одного узла может быть использован более эффективно с помощью парадигмы общей памяти OpenMP.

С точки зрения программы это означает, что распараллеливалась изначально не последовательная версия, а OpenMP версия бенчмарка, таким образом, все циклы будут распараллелены с помощью OpenMP автоматически. При этом требуется не вызывать функций shmem из параллельных секций (можно вызывать функции `shmem_put` и `shmem_get`, если обеспечить их выполнение в критических секциях).

К сожалению, даже оригинальная версия NPB-OMP дает на одном узле Blue Gene/P ускорение в 1.22 раза на четырех ядрах одного узла относительно производительности одного ядра, что свидетельствует о не самой лучшей реализации критических секций и семафоров OpenMP на данной архитектуре. В связи с этим, гибридная версия давала всего лишь 10 процентный прирост относительно чистой SHMEM-версии, использовавшей одно ядро на узел.

В то же время, например, при использовании 128 узлов и SHMEM-версии, VN режим дает ускорение 2.26 раза относительно SMP-режима. Единственной системой, где гибридный параллелизм дал ощутимый прирост, оказалась система МВС-Экспресс.

3. Результаты

На графике приводятся результаты, полученные на стратегическом суперкомпьютере IBM Blue Gene/P в зависимости от суммарно использованного количества ядер. Исследовались три режима работы комплекса Blue Gene/P: режим SMP при котором используется одно ядро на узле, режим DUAL - два ядра на узел, режим VN - четыре ядра в узле. Для запуска были доступны размеры от 128 до 1024 узлов выделяемых под задачу. Меньшее число узлов запустить на установленном в МГУ суперкомпьютере Blue Gene/P нельзя из-за малого количества IO-узлов системы.

Запускался класс C задачи, который описывается следующими параметрами: число элементов 33500 (число точек коллокации 4 млн.), число точек согласования 1.26 млн., максимальная глубина измельчения каждого элемента - 8 шагов. Объем памяти, занимаемой последовательной версией - 508 МБ. Параллельная версия требует 166 МБ на статические массивы (целочисленная информация о сетке) и 150 МБ PGAS-памяти на каждом узле. Основной потребитель памяти - массивы для программной реализации атомарных операций.

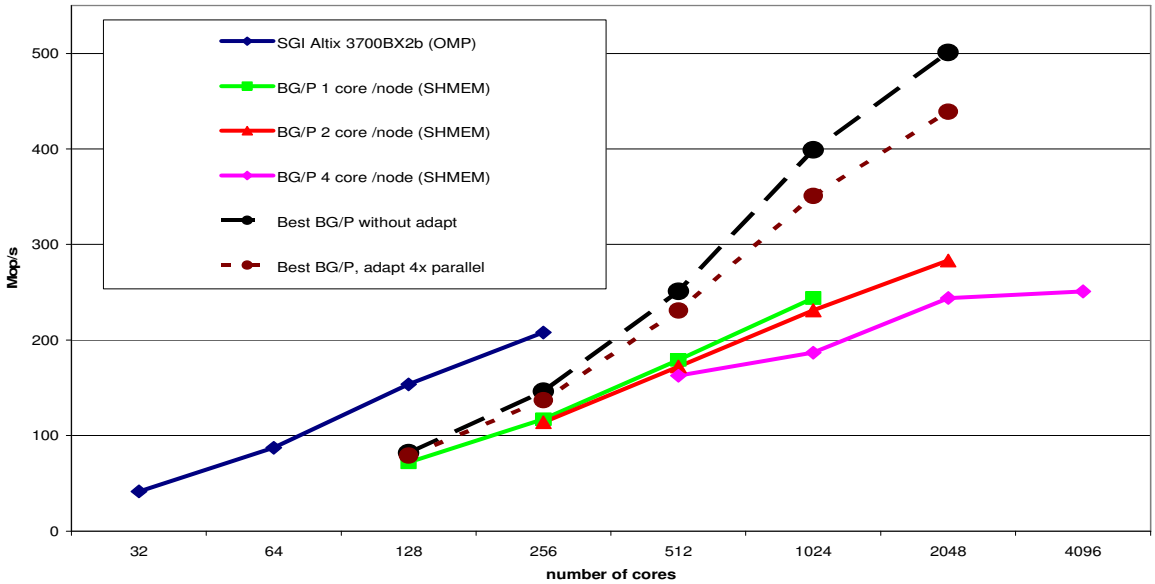


Рис.4 Абсолютная производительность NPB-PGAS UA (class C) на суперкомпьютерах IBM Blue Gene/P и SGI Altix 3700bx2.

Для сравнения приведены максимальные цифры производительности, полученные от NAS Advanced Supercomputing Division, полученные на OpenMP-версии на сравнительно старой ма-

шине с общей памятью SGI Altix 3700BX2b, 512 Itanium2, 1.6GHZ, 9MB L3, NUMALink 4. Максимальная производительность на 256 ядрах Itanium2 достигает 207 млн. операций в секунду. Под условной операцией в этом тесте понимается одна итерация метода сопряженных градиентов или интегрирование методом Рунге-Кутта. Такая единица измерения выбрана из-за неудобства оценки производительности операциями с плавающей точкой, так как ни scatter ни gather таких операций не содержат, и сравнение производительности запусков в FLOP/s при различном числе итераций метода CG было бы бессмысленным. В будущем, было бы интересно сравниться с результатами, полученными на новом поколении машин SGI Altix UV.

Из графика видно, что SMP и DUAL режимы близки по производительности при одинаковом количестве используемых ядер, в то время как VN режим несколько "проседает". Это согласуется с информацией о масштабировании SHMEM приведенной в разделе 2.1.

Максимальная производительность в 283 млн. операций в секунду достигается на 1024 узлах в режиме DUAL. На 4096 производительность падает из-за уже упоминавшегося проседания VN-режима, и из-за ограниченности задачи: в этом случае на один процесс приходится всего менее 8 элементов сетки и 315 точек согласования, а это слишком мелкий уровень гранулярности даже для такой машины как Blue Gene/P.

Следует отметить, что на большом количестве ядер значительную часть времени начинает занимать процедура адаптации сетки, которая автором в данное время распараллелена не была. Например, на 128 ядрах адаптация занимает 14 секунд из 122, а на 2048 ядрах процедура адаптации занимает 14 секунд из 31 общего времени счета задачи. На 1024 ядрах абсолютное ускорение достигает 100 раз, без учета времени адаптации получаемое ускорение составляет 171.

В дальнейшем планируется распараллелить адаптацию или на все узлы системы, или что было бы гораздо проще на 4 ядра одного узла. То есть каждый узел выполнял бы адаптацию независимо, но несколько ядер одного треда работали бы совместно. Поэтому на рисунке пунктиром обозначены две кривые, одна показывающая результат при нулевом времени адаптации, другая при 4х кратном ускорении времени адаптации, что показывает достижимость уровня производительности в 450-500 млн. операций в секунду на классе C.

3.2 Другие архитектуры, на которых была запущена NPB-PGAS UA

Кроме суперкомпьютера IBM Blue Gene/P, PGAS-версия бенчмарка была запущена на экспериментальном кластере из 6 узлов, на базе разработанного в НИЦЭВТ макета Ангара-М2 отечественной коммуникационной сети на базе ПЛИС, на экспериментальной системе МВС-Экспресс. На обычных SMP узлах и на vSMP системе запускалась OpenMP версия бенчмарка.

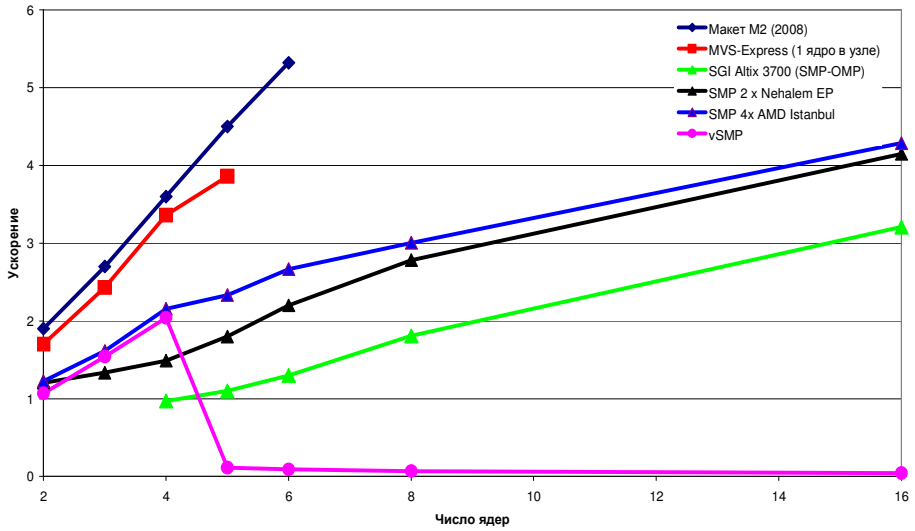


Рис.5 Ускорение на тесте NPB UA от числа используемых ядер.

Резкое падение производительности на vSMP-системе связано с тем, что до 4-х ядер задача работала на одном узле (с четырёхядерным процессором Intel Xeon QC), при большем же числе ядер были задействованы несколько узлов, связанных коммуникационной сетью Infiniband. При использовании нескольких узлов появляются частые промахи по виртуальным адресам, вызывающие исключения (page fault). В результате промаха на чтение одного слова (8 Б) в локальную физическую память подкачивается целая страница (4 КБ); таким образом, при нерегулярных обращениях к памяти накладные расходы на передачи по сети существенно превышают долю вычислений. В целом видно полное превосходство SHMEM-версии на системах, которые разрабатывались специально под SHMEM относительно OpenMP-версии.

4. Заключение

В заключение можно отметить, что применение парадигмы PGAS показало как высокую продуктивность, так и высокую эффективность даже на не имеющих аппаратной поддержки общей памяти стратегических машинах текущего поколения, таких как IBM Blue Gene/P. Будущие поколения суперкомпьютеров разработанных в рамках программы DARPA HPCS будут иметь еще большую производительность на данном классе PGAS-приложений.

Автору удалось на 2048 ядрах системы Blue Gene/P показать на 37 процентов лучший результат, чем максимально известный результат, который показывает OpenMP-версия на 256 ядрах SGI Altix 3700BX2B. Ожидаемый результат следующей версии, включающей распараллеленную версию адаптации сетки, превысит на 2048 ядрах результат Altix в более чем два раза.

Планируются следующие направления работ в дальнейшем: распараллелить адаптацию сетки, как в рамках одного узла, так и между всеми узлами, за счет этого и другой реализации атомарных с плавающей точкой радикально уменьшить потребление памяти и запустить класс D на Blue Gene/P, для чего потребуется расширить библиотеку SHMEM активными операциями. Также рассматриваются варианты ускорения реализации SHMEM на BG/P за счет экономии на подтверждениях отправки DCMF, а также реализовать агрегацию сообщений прозрачную для прикладного программиста. Также с помощью системы визуализации [4] планируется визуализировать работу метода на BG/P. Хорошо было бы провести измерения на стратегических суперкомпьютерах Cray серии XT5/XT6 (Baker в 2010 году) и на SGI Altix UV.

Автор выражает благодарности А.О. Лацису за полемически заостренную форму постановки вопросов и предоставленный доступ к коммуникационной системе МВС-Экспресс, факультету ВМиК МГУ за предоставленный доступ к единственному в России стратегическому суперкомпьютеру, изготовленному одной из двух суперкомпьютерных в мире компаний, H. Jin Haoqiang из NASA Advanced Supercomputing Division за предоставленные данные по производительности NPB-OMP UA на SGI Altix, компании Cray Inc. за разработку в 1991 гениальной и простой библиотеки Cray SHMEM.

Литература

1. Kumar, S., et al, The deep computing messaging framework: generalized scalable message passing on the blue gene/P supercomputer. // 22nd Annual international Conference on Supercomputing, June 07 - 12, 2008, Island of Kos, Greece. Proceedings, ACM. 2008. P. 94-103.
2. Moffett Field, H. Feng, R.F. Van der Wijngaart, R. Biswas Unstructured adaptive (UA) NAS Parallel Benchmark. NASA Ames Research Center, CA, 2004.
3. Лацис А.О. Вычислительная система МВС-Экспресс
URL: http://www.kiam.ru/MVS/research/mvs_express.html
4. Dzhosan O.V., Popova N.N., Korzh A.A. Hierarchical Visualization System for High Performance Computing // 14th International conference on Parallel Computing, September 1-4, 2009, Lyon, France, Proceedings.