

РАСПАРАЛЛЕЛИВАНИЕ ЗАДАЧ С НЕРЕГУЛЯРНЫМ ДОСТУПОМ К ПАМЯТИ С ПОМОЩЬЮ РАСШИРЕННОЙ БИБЛИОТЕКИ SHMEM НА СУПЕРКОМПЬЮТЕРЕ BLUE GENE/P

А.А. Корж

Аннотация. Описывается библиотека программирования SHMEM и модель параллельного программирования характерная для задач использующих эту библиотеку. Описываются предлагаемые расширения библиотеки SHMEM с помощью нестандартных схем синхронизации и активных сообщений. Рассматриваются детали распараллеливания бенчмарка NASA NPB UA с помощью данной библиотеки реализованной автором на суперкомпьютере Blue Gene/P. Приводятся полученные результаты, проводится сравнение с реализацией SHMEM поверх библиотеки MPI-2.

Введение.

Одним из важных свойств суперкомпьютеров является продуктивность их программирования, так как именно от продуктивности зависит возможность использования суперкомпьютеров для решения той или иной задачи. В последние годы признается, что именно сложность параллельного программирования и нехватка опытных программистов является сдерживающим фактором широкого распространения высокопроизводительных вычислений в России. Причем основную сложность представляет собой не сколько процесс написания и отладки корректно работающей параллельной программы, сколько оптимизация этой программы. Проблемой здесь является необходимость учитывать множество нюансов используемой архитектуры суперкомпьютера, то есть программисту приходилось бы «опускаться» на уровень железа.

В настоящее время стандартом параллельного программирования де-факто является модель программирования Message-Passing. Широкое распространение многоядерных процессоров и многосокетных узлов привело лишь к тому, что модели MPI гибридным образом добавили использование OpenMP. Это усложнило и без того нелегкий труд написания эффективных параллельных программ. Появление же графических ускорителей внесло еще большую сумятицу, расширив стандартные модели программирования возможностью ускорения отдельных частей программы с помощью новых инструментальных средств.

Попытки разработать альтернативные модели программирования, такие как PGAS, представленные языками UPC, CAF, Titanium, Global Arrays и перспективными Chapel, Fortress и X10 или модель программирования Message-Driven представленную языком программирования Charm++ оглушительного успеха по разным причинам не имели. Также можно упомянуть оригинальные отечественные разработки: T-система, основанная на теории параллельной редукции графов и DVM-система, значительно упростившая написание широкого класса задач, по сравнению с MPI. Для полноты картины остаются редко используемая модель конвейерных вычислений и мультитредовая модель вычислений, широко подхваченная в последнее время в связи с широким распространением графических ускорителей. При этом основная проблема параллельного программирования — сложность оптимизации была не только не решена, но наоборот усугублена, что и привело к состоянию, когда используется в основном MPI и OpenMP а остальные парадигмы представляют лишь научный интерес.

В связи с этим предлагается по-новому взглянуть на забытую библиотеку SHMEM, показавшую в [1] возможность ее эффективного применения на тысячах ядер современного суперкомпьютера. Преимуществами данной библиотеки являлась как простота программирования «растущая» из модели общей памяти, так и высокая эффективность вызванная односторонностью коммуникаций. При этом проблема сложности оптимизации представляется частично решенной, так как сводится в основном к управлению локальностью данных.

Библиотека SHMEM.

Модели параллельного программирования можно разделить на два класса, в зависимости от того, на коммуникациях какого типа они основаны: односторонних (обращения к удаленной 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 Research 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, Cray XE6), Silicon Graphics (SGI Altix), интерконнектами Quadrics (QsNetIII). Также библиотека SHMEM (с некоторыми дополнениями) реализована в системе MVS-Экспресс

(разработана под руководством А. О. Ладиса) и в коммуникационной сети разработанной под руководством автора в НИЦЭВТ. Кроме того данная библиотека была реализована автором на суперкомпьютере Blue Gene/P через библиотеку нижнего уровня Deep Computing Messaging Framework.

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.

Преимущество SHMEM относительно MPI с точки зрения производительности сказывается особенно сильно в области задач работающих с небольшими блоками данных, хаотично разбросанными по огромным массивам распределенной памяти. Это можно увидеть, рассмотрев отношения APEX-поверхностей для SHMEM и MPI у различных систем: Cray X1E, IBM Blue Gene /P. Причем следует учесть, что текст программы APEX-SHMEM значительно более прост, чем вариант программы написанной на MPI. SHMEM вариант просто подкачивает данные с помощью операции `get`, в то время как MPI версия посылает запросы и обрабатывает приходящие запросы извне. Большая эффективность библиотеки SHMEM для коротких сообщений означает возможность эффективного использования мелкозернистого параллелизма, а следовательно, означает лучшую сильную масштабируемость, так как при большом числе узлов объемы данных приходящиеся на один узел становятся небольшими. Таким образом, при прочих равных тот же алгоритм, распараллеленный с помощью SHMEM, будет масштабироваться на большее число узлов, чем программа написанная с помощью MPI.

На рисунках 1 и 2 изображены диаграммы показывающие отношения APEX-поверхности для SHMEM относительно MPI для векторной машины Cray X1 и для массивно параллельной системы IBM Blue Gene/P.

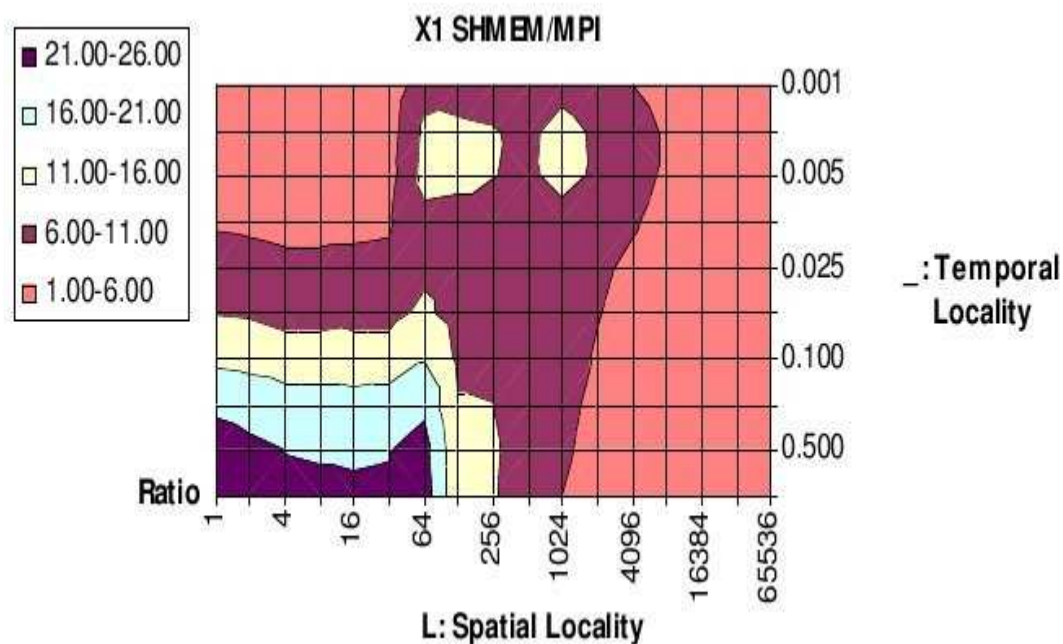


Рис. 1. Отношение SHMEM версии APEX-поверхности относительно MPI версии для 256 узлов Cray X1 в зависимости от параметров временной и пространственной локализации.

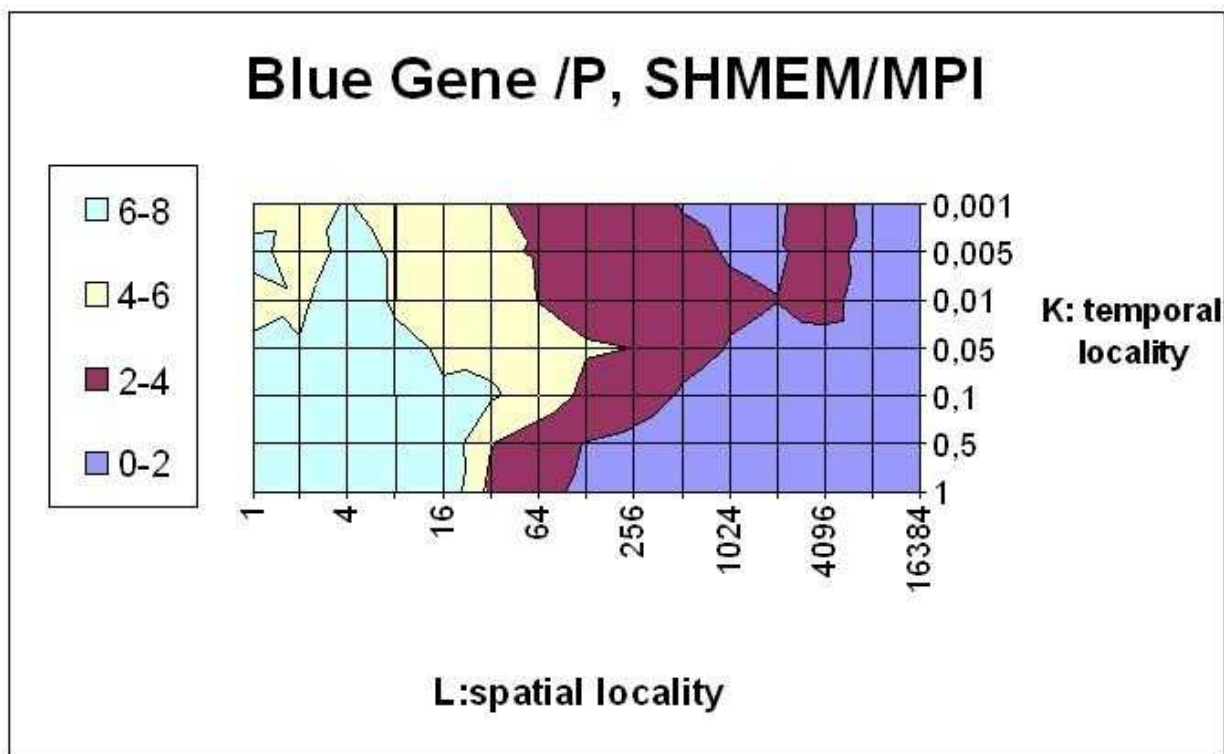


Рис. 2. Отношение SHMEM версии APEX-поверхности относительно MPI версии для 128 узлов IBM Blue Gene /P в зависимости от параметров временной и пространственной локализации

Распараллеливание NASA NPB UA.

При распараллеливании бенчмарка NASA NPB UA в [1] поначалу использовались только функции, входившие в стандартную библиотеку SHMEM. Основным подходом, применявшимся при распараллеливании NPB UA, был подход перехода от OpenMP версии к SHMEM. Таким образом, итерации OpenMP-параллельных циклов распределялись между узлами блочным образом. Все массивы также распределялись между узлами.

При этом ставилась цель минимизации обращений к данным других узлов. Зачастую для этого было достаточно простого блочного распределения массивов между узлами. Большая часть итераций параллельных циклов при этом содержала обращения к элементам массивов индексированных индексом равным номеру итерации цикла. Условно циклы можно поделить на несколько типов, в зависимости от зависимостей итераций от данных.

Первый тип — полностью параллельные по данным циклы. Это самый простейший тип и он проблем не вызывал — каждый узел выполнял свой блок итераций цикла и оперировал при этом только с локально расположенными данными.

Второй тип — циклы, итерации которых читают только локально расположенные данные, но могут писать в ячейки массивов расположенные на других узлах. В данном типе циклов записи потенциально удаленных ячеек заменялись соответствующими неблокирующими вызовами функции `shmem_put`. После окончания всех итераций цикла все узлы выполняли коллективный вызов `shmem_barrier_all`.

Третий тип циклов — циклы итерации которых содержали и записи и чтения данных находящихся на других узлах. Записи аналогично выполнялись с помощью функции `shmem_put`, а чтения выполнялись с помощью функции `shmem_get`. Причем сразу после вызова `get` ставился вызов `shmem_fence`, который дожидался выполнения чтения. Этот подход неплохо работал в случае редких удаленных чтений, но был неприемлем в случае большого количества удаленных чтений. Причина этому — простой ядра во время ожидания каждого удаленного чтения, таким образом, время простоя каждого узла было прямо пропорционально количеству выполняемых им удаленных чтений. В частности в операции `Gather` присутствует большое число удаленных чтений. Поэтому для эффективного распараллеливания таких циклов требуется другой подход. Основной идеей является выполнение всех необходимых удаленных чтений группой в предварительном цикле подкачки. После чего работает основной цикл, который уже не содержит зависимостей по данным от удаленных узлов — все необходимые данные находятся на локальном узле. Заметим, что коммуникационный шаблон цикла преднакачки повторяет шаблон теста `RandomAccess`.

Еще одной оптимизацией здесь является замена операций `get` на операции `put` в цикле подкачки. Идея заключается в том, что `get` состоит из двух посылок: запроса и ответа. Если пачка операций `get` повторяется несколько раз с одними и теми же адресами источника и назначения, то можно один раз запомнить эти адреса на узле, хранящем данные, а потом лишь посылать сами данные с помощью операции `put` состоящей из одной посылки. Это повлечет за собой ускорение работы цикла преднакачки до двух раз.

Расширения библиотеки SHMEM.

В качестве расширений предлагаются следующее: активные сообщения, более тонкая синхронизация групп односторонних операций и двухадресные чтения. Все эти операции востребованы по итогам реальной работы распараллеливания псевдореальной задачи `NPB UA`. Часть этих операций реализована на макете коммуникационной сети разрабатываемой в НИЦЭВТ и на суперкомпьютере стратегического назначения `IBM Blue Gene/P`.

Активные сообщения представляют собой любые функции задаваемые пользователем, посылаемые на другой узел одной посылкой. Важным свойством является атомарность выполнения обработчиков на одном узле. Простейшим примером такого сообщения является атомарный инкремент одной ячейки массива, хотя, очевидно, и операция записи может быть реализована как активное сообщение. Требуются данные операции, если на какой либо итерации происходит большое количество сгруппированных операций с одним и тем же нелокальным индексом — тогда оказывается проще и эффективней послать одно активное сообщение на тот узел, где и находятся все данные, чем выполнять большое количество операций `get` и `put`. При реализации существует два варианта выполнения данных операций узлом. В одном случае выделяется специальный тред (так называемый `remote agent`), в другом случае операции выполняются внутри вызовов библиотеки (пассивный прогресс). Для симметрично сгруппированных операций, как это происходит в цикле преднакачки оказалось достаточно второго подхода, таким образом, на суперкомпьютере `BlueGene/P` посылка выполняется с помощью функции `DCMF_Send` а прием с помощью вызова функции `DCMF_Messenger_advance`. С точки зрения прикладного программиста — доступно две функции `shmem_register_handler` и `shmem_send`. Первая регистрирует обработчик и является коллективной функцией. Вторая отправляет сообщение, в котором указываются данные и номер обработчика, присвоенный первой функцией. Выполнение направленных данному узлу активных сообщений гарантируется вызовом `shmem_barrier_all` точно так же, как это происходит в случае с операциями `put`.

Расширенные возможности синхронизации различных групп операций предназначены для придания большей гибкости при синхронизации различных операций записи. В стандартной библиотеке `shmem` для синхронизации используется две функции `shmem_fence` и `shmem_barrier_all`, которые дожидаются завершения выполнения всех запущенных операций по сети. Иногда требуется дождаться завершения не всех запущенных операций, а только некоторого подмножества. Еще одна проблема возникает, если мы хотим для экономии коммуникаций использовать записи, которые не подтверждаются ответным пакетом по сети. В таком случае, реализация операции `shmem_fence` становится очень дорогой, а синхронизация с помощью `shmem_barrier_all` даже если и может быть реализована эффективно, неудобна прикладному программисту, так как это коллективная операция и должна быть вызвана всеми без исключения узлами.

Для преодоления этих проблем предполагается введение нового понятия сегмента. Сегмент является локальным понятием, содержит в себе область памяти, выделенную с помощью `shmem_alloc` и счетчик входящих сообщений. Операции `shmem_put_seg` аналогичны функциям `shmem_put`, нужно лишь указать номер сегмента, предварительно зарегистрированный с помощью коллективной функции `shmem_register_seg`. Данной функцией каждый узел передает число ожидаемых сообщений. После этого каждый узел может отправить свои сообщения другим узлам и вызвать функцию `shmem_sync_seg` ожидающую получение заранее указанного числа сообщений. Данная функция не является коллективной и реализуется эффективно, даже в случае использования пакетов без подтверждений, позволяя таким образом сэкономить до половины трафика передаваемого по коммуникационной сети.

Двухадресные чтения отражают ситуацию, при которой приходится запрашивать с одного узла значение лишь для того, чтобы записать его в ячейку на другом узле. В таком случае практически было бы организовать посылку ответа на запрос чтения сразу в нужную ячейку на другом узле. Один из вариантов реализации таких чтений — через активные сообщения. Более эффективна организация на аппаратном уровне, без участия процессора, однако, это не получается реализовать через стандартные интерфейсы нижнего уровня. На суперкомпьютере Blue Gene/P как показали исследования, лишь интерфейс самого нижнего уровня SPI поддерживает выполнение трюков такого рода. Планируются дальнейшие исследования возможности эффективной реализации и удобного использования таких функций в прикладных программах.

Реализация SHMEM на Blue Gene/P.

Родной реализации SHMEM на суперкомпьютере IBM Blue Gene/P нет. Однако используемый в суперкомпьютере Blue Gene/P интерфейс DCMF [2] 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 M3 специализированной сети, разработанной в НИЦЭВТ под руководством автора дает темп выдачи 32 млн/с, система МВС-Экспресс [3], разработанная под руководством А.О. Лациса имеет темп выдачи 24 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.

Результаты.

После практического полного распараллеливания бенчмарка NPB UA с помощью расширенной описанными выше функциями были получены следующие результаты. В [1] приведены результаты распараллеливания бенчмарка притом, что часть программы была оставлена последовательной. Эта часть — процедура изменения сетки. С одной стороны распараллеливание адаптации имело большую трудоемкость, с другой стороны ее доля во времени последовательной программы была менее одного процента. Однако при распараллеливании на восемь тысяч ядер из-за закона Амдала требовалось распараллеливание этой части. Сложность распараллеливания процедуры адаптации сетки была вызвана тем, что в циклах имелось множество зависимостей по данным, а с другой стороны именно в процессе выполнения этой процедуры происходит полная перенумерация элементов сетки, а так же изменяется функция распределения в связи с изменением числа элементов. При распараллеливании этой части и понадобились описанные выше расширения библиотеки SHMEM.

Таблица 1. Результаты на NPB UA Class C

	serial	32	64	128	256	512	1K	2K	4K	8K
Altix OpenMP	5.9	42	87	154	207					
BG/P DCMF	2.5	34	50	82	147	264	437	670	777	659
BG/P MPI-2	2.5	14	18	27	46	70	76	40	12	2

Таблица 2. Результаты на NPB UA Class D

	64	128	256	512	1K	2K	4K	8K

BG/P
SHMEM 83 126 190 345 620 1160 1880 2204
DCMF

Автор выражает благодарность А.С.Фролову за отладку MPI и SHMEM версий теста APEX-поверхности.

Работа выполнена при финансовой поддержке РФФИ грант 09-07-13596 офи_ц.

ЛИТЕРАТУРА:

1. А.А. Корж Результаты масштабирования бенчмарка NPВ UA на тысячи ядер суперкомпьютера Blue Gene/P с помощью PGAS-расширения OpenMP // Вычислительные методы и программирование, том 11, с.31
2. 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.
3. Лацис А.О. Вычислительная система МВС-Экспресс http://www.kiam.ru/MVS/research/mvs_express.html