КОМПЬЮТЕРНЫЕ НАУКИ

УДК 519.684

© *С. П. Копысов, А. К. Новиков, Л. Е. Тонков, Д. В. Береснев*

МЕТОДЫ ПРИВЯЗКИ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ И ПОТОКОВ К МНОГОЯДЕРНЫМ УЗЛАМ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ[1]

Рассматриваются способы и варианты привязки параллельных процессов и потоков к ядрам, сокетам вычислительных узлов. Приводятся результаты выполнения тестовых примеров на MPI и MPI/OpenMP. Обсуждаются возможности достижения желаемого размещения параллельного приложения на процессорах и ядрах.

Ключевые слова: параллельные вычисления, привязка процессов, многоядерные процессоры, промежуточное программное обеспечение.

Введение

Сегодня только небольшая часть разработанного программного обеспечения, реализованного в модели программирования MPI, может эффективно выполняться на современных многоядерных вычислительных узлах. Применение многоядерных процессоров позволяет получать более совершенные распределения процессов для задействования ядер в целях минимизации конфликтов в кэш-памяти и оперативной памяти, процессоре и т. д. Для этого пользователь должен иметь возможность определения плана размещения и выполнения задания на вычислительных узлах, процессорах и ядрах [1].

Привязка позволяет сохранить местоположение данных и не допускает миграцию процесса далеко от кэша, который содержит его данные. Для предотвращения перемещения процесса или потока будем связывать процесс с логическими процессорами. В том случае, когда привязка не используется, операционная система может переместить процесс или поток на другой процессор (в силу различных причин). Процесс привязки позволяет получать более устойчивые показатели производительности, потому что перемещение процесса требует аннулирования кэша и может ухудшить производительность приложений.

В современных вычислительных системах, как правило, применяются многоядерные (до 12 ядер), многосокетовые (до 8 сокетов) вычислительные узлы, использующие стандартные комплектующие для построения высокопроизводительных систем. Данные системы характеризуются многоуровневой организацией коммуникаций и многоуровневым доступом к оперативной памяти. С увеличением количества процессорных ядер в значительной степени возрастает и сложность платформы. При этом некоторые ресурсы дублируются, а некоторые используются совместно. В NUMA-архитектуре разделение данных между узлами может быть весьма затратным по причине удаленного доступа к памяти. Это также следует принимать во внимание при миграции потоков, особенно между разными узлами, так как в случае кэш-промахов потребуется доступ к памяти другой группы ядер.

Сложность состоит в распределении потоков и процессов между ядрами таким образом, чтобы оптимально задействовать возможности системной топологии и архитектуры. В большинстве случаев планировщики в операционных системах (ОС) общего назначения назначают потоки ядрам с наименьшей загрузкой, чтобы обеспечить сбалансированную нагрузку, либо ядрам с уже подготовленными данными в кэше (cache-warm), используя преимущества привязки к кэшу (cache affinity).

[1]Работа выполнена при финансовой поддержке РЦП УрО РАН и РФФИ (грант 09–01–00061).

Как правило, на многосокетовой/многоядерной архитектуре MPI-приложение запускается таким образом, что несколько параллельных процессов выполняются на одном вычислительном узле и связываются с множеством процессов на других вычислительных узлах по коммуникационной сети. Поэтому алгоритмы коммуникаций (особенно коллективных) должны быть построены и реализованы так, чтобы эффективно использовать такую иерархию топологии коммуникаций. Различные реализации MPI содержат свои механизмы оптимизации выполнения межузловых и внутриузловых коммуникаций (алгоритмы, функции, параметры запуска приложения). И здесь к факторам, имеющим существенное значение, можно отнести расположение процессов/ядер на вычислительных узлах.

processor	0	1	2	3	4	5	6	7
model name	Intel(R) Xeon(R) CPU X5365 @ 3.00GHz							
cpu MHz	3000.111							
cache size	4096 KB							
physical id	0	1	0	1	0	1	0	1
core id	0	0	2	2	1	1	3	3
cpu cores	4							

Рис. 1. Вычислительный узел МВС-100К

processor	0	1	2	3	4	5	6	7
model name	Intel(R) Xeon(R) CPU E5430 @ 2.66GHz							
cpu MHz	2660.073							
cache size	6144 KB							
physical id	0	0	1	1	0	0	1	1
core id	0	2	0	2	1	3	1	3
cpu cores	4							

Рис. 2. Вычислительный узел кластера X4

Нумерация ядер изменяется для разных систем и ее нельзя наследовать для достижения оптимальных результатов на всех системах. Например, для двух систем на которых проводилось тестирование МВС-100К МСЦ РАН (каждый ВУ — два четырехядерных процессора Xeon 5365, 3GHz) и кластер X4 ИПМ УрО РАН (каждый ВУ — два четырехядерных процессора Xeon 5420, 2.66GHz), данные приведены на рис. 1 и рис. 2.

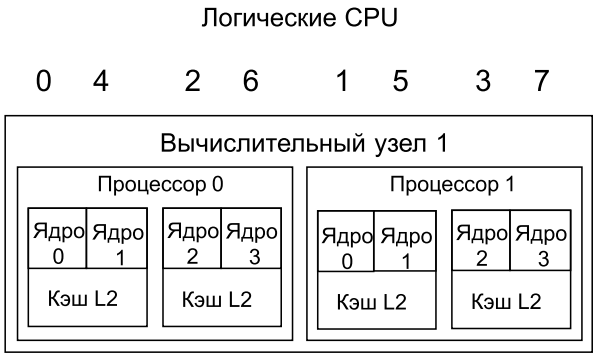

Рис. 3. Соответствие логических CPU ядрам вычислительных узлов МВС-100К

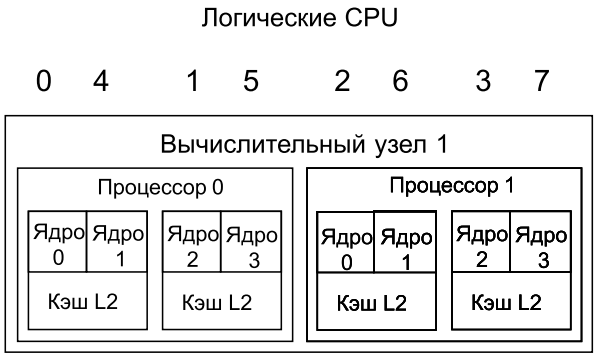

Рис. 4. Соответствие логических CPU ядрам вычислительных узлов кластера X4

Соответствие логических CPU ядрам вычислительных узлов для них показано на рис. 3, рис. 4. Для получения желаемого размещения на процессорах и ядрах пользователи должны знать отображение ядер на используемой вычислительной системе.

Отметим, что при выполнении привязки процессов к ресурсам вычислительной системы следует учитывать комбинацию аппаратного обеспечения, системного и промежуточного программного обеспечения.

§ 1. Способы привязки процессов для Unix-систем

Промежуточное обеспечение MPI не поддерживает полного связывания с CPU, поэтому необходимо использовать дополнительные средства, позволяющие управлять привязкой процессов при выполнении `mpirun`. Рассмотрим существующие способы привязки, отличающиеся

вариантами задания и взаимодействием с промежуточным программным обеспечением (различные MPI-реализации, планировщики).

1.1. Высокоуровневые системные вызовы привязки

Для вычислительных узлов (ВУ) с многоядерными процессорами операционная система определяет нумерацию ядер (см. `cpuinfo` в табл. 1).

Для привязки процессов в Linux пользователь может привязать процесс к отдельному CPU, используя вызов функций `sched_setaffinity()`, `sched_getaffinity()` с соответствующим параметром `affinity_mask`. Эти функции могут отличаться типом параметров в зависимости от производителей Linux и версий библиотеки Glibc. Реализованы функции в библиотеке PLPA (The Portable Linux Processor Affinity) [2]. Привязка процесса к определенному CPU (`cpu_affinity`) исключает ситуацию мигрирования между процессорами, а также позволяет изменить для него алгоритм работы планировщика задач и увеличить приоритет.

Отметим также, что для привязки процессов в других операционных системах также имеются системные вызовы: в Windows — `SetThreadAffinityMask()` ; в Solaris — `pbind()`.

1.2. Использование поддержки компилятора

Наряду с возможностью привязки на уровне операционной системы в некоторых компиляторах существуют средства для управления привязками. Например, для компиляторов Intel C/C++ для этого используется переменная окружения `KMP_AFFINITY`. Данная переменная по существу разрешает три реализации: привязки потоков к отдельным ядрам или перемещения их среди имеющихся ядер процессора; определения маски привязки потоков по умолчанию; расположения последовательности потоков на соседних ядрах (для оптимизации эффектов кэша памяти).

Другая возможность контроля привязки потоков к процессорам на уровне компилятора реализована PathScale Compiler Suite [3].

1.3. Промежуточное программное обеспечение SLURM

В настоящее время практически все менеджеры ресурсов не позволяют адекватно размещать процессы и выполнять стратегии планирования на многоядерных вычислительных узлах. Не так давно появилась система управления ресурсами SLURM (Simple Linux Utility for Resource Management) [4], включающая компоненты: анализ состояния кластера, система планирования, модули управления и планирования с поддержкой многоядерности.

Более подробно остановимся на функциональности SLURM, которая включает поддержку многоядерности и привязки: 1. Размещение заданий по вычислительным узлам, сокетам, ядрам и потокам: явное указание масок с `--cpu_bind` и `--mem_bind`; автоматическое получение привязки простыми директивами. 2. Выполнение контроля над заданиями располагаемыми на сокетах, ядрах и потоках.

Размещение заданий осуществляется командой выделения ресурсов `salloc` или командой запуска `srun` с указанием вычислительных узлов и привязок.

Для обычных пользователей SLURM предоставляет привязку заданий к логическим процессорам (`sockets`, `cores` на сокетах и `threads` на ядрах) через множество логических процессоров, использующих высокоуровневые флаги, а для подготовленных пользователей — через низкоуровневые флаги.

Пользователь устанавливает окружение, включая все необходимые CPU и автоматическое генерирование маски. В этом случае используются высокоуровневые флаги, поддерживающие привязку процессов к ядрам. Пользователям разрешается подключать эту функциональность SLURM двумя способами: использованием высокоуровневых флагов, действующих через абстрактный слой, прозрачный для пользователя, и создающих отображение логических процессоров на физические ядра; указанием маски на прямую через флаг `--cpu_bind`.

В SLURM реализован вызов функции `task`, оперирующей логическими процессорами. Параметром `--ntasks-per-{node,socket,core}=ntasks` пользователь может задать максимальное число заданий `ntask`, приходящихся на каждый вычислительный узел, сокет, ядро. Для многопоточных приложений применяется флаг `--cpus-per-task=ncpus`, выделяющий `ncpus` ядер на процесс.

1.4. Промежуточное программное обеспечение MPICH2 со SLURM

На кластере X4 ИПМ УрО РАН задания запускаются в окружении SLURM с MPICH2. Запуск приложения осуществляется входящей в SLURM командой `srun`:

```
srun -n64 -w имена узлов --cpu_bind=map_cpu:0,1,4,5,2,3,6,7 bt.B.64
```

Привязка задается значением параметра `cpu_bind`. Кроме `map_cpu` и `mask_cpu` с явным указанием номеров логических CPU(ядер) `cpu_bind` принимает значения: `sockets`, `cores`, `threads`, `rank` (см. табл. 1).

1.5. Промежуточное программное обеспечение MVAPICH

В MVAPICH поддержка привязки обеспечивалась через PLPA. Привязка может быть определена установкой переменных окружения `MV2_CPU_MAPPING`. Кроме того, SLURM поддерживает привязку в MVAPICH, но на МВС-100К библиотека не установлена.

Приложение запускается скриптом `mpirun`:

```
mpirun -np 64 ... VIADEV_USE_AFFINITY=1 VIADEV_CPU_MAPPING=parameters bt.B.64
```

Параметр `VIADEV_USE_AFFINITY=1` инициирует привязку MPI-процессов. Собственно распределение задается параметром `VIADEV_CPU_MAPPING`. Номера логических CPU(ядер) указываются явно, например `VIADEV_CPU_MAPPING=0,2,4,6,1,3,5,7` (см. табл. 1).

Для проверки правильности задания привязки в обоих случаях (MPICH2 и MVAPICH) можно вызывать функцию `sched_getaffinity` в каждом MPI-процессе.

1.6. Промежуточное программное обеспечение Open MPI

Отметим, что Open MPI является одной из реализаций MPI, которая содержит собственный механизм привязки процессов. В этом случае приложение запускается с собственными параметрами привязки Open MPI:

```
mpirun -np 64 ... параметр привязки ./xhpl
```

где `параметр привязки`: `bynode` — привязка к вычислительным узлам по кольцевой схеме; `byslot` — привязка к процессорам по кольцевой схеме; `loadbalance` — равномерное распределение процессов; `slot-list 0,4,1,5,2,6,3,7` — явное соответствие процессов логическим CPU. Маска привязки может быть указана в отдельном файле и задействована с опцией `rankfile` *ranks*, здесь *ranks* — имя файла. Файл содержит последовательность строк вида:

```
rank номер процесса=имя узла slot=номер слота:номер ядра
```

При запуске Open MPI заданий в SLURM ресурсы для заданий выделяются командой `salloc` (SLURM), а запуск осуществляется `mpirun` (Open MPI) с обычным набором аргументов. Список вычислительных узлов и процессов, которые будут выполняться на каждом узле, передается из SLURM.

1.7. Привязка в гибридной модели MPI/OpenMP

Привязка позволяет определить, какое ядро будет выполнять потоки. Установка маски привязки применима как в случае чистого MPI-приложения, так и для гибридного MPI/OpenMP, но особенно значимо влияние привязки в гибридной модели параллельного программирования.

Для MPI-приложения устанавливаются маски привязки, гарантирующие, что процессы ограничены отдельными ядрами. При использовании гибридной MPI/OpenMP-модели OpenMP-потоки создаются как часть MPI-процесса. Если привязка потоков не установлена явно, они все наследуют привязку MPI-процесса. Это подразумевает, что по умолчанию все OpenMP-потоки будут выполняться на том же ядре, что и MPI-процесс. В этом случае становится важным установить привязку явно при использовании гибридной модели программирования.

В случае применения привязок `map_cpu`, `rank`, `cores`, `threads` в программах, основанных на гибридной модели MPI/OpenMP, используются не все доступные ядра ВУ. MPI-процессы получают привязку к ядрам, и все порождаемые ими OpenMP-потоки наследуют эту привязку. Например, в случае программы, запущенной с двумя MPI-процессами на двух ядрах и порождением в каждом процессе еще четырёх OpenMP-потоков, загружены будут только два ядра — по четыре потока на каждом ядре.

При запуске гибридного MPI/OpenMP-приложения число запускаемых OpenMP-потоков задается переменной окружения `OMP_NUM_THREADS`. Для осуществления миграции OpenMP-потоков на свободные ядра вычислительного узла использовался флаг `--cpu_bind=sockets` или сочетание флагов `--cpus-per-task` и `--cpu_bind`. Во втором случае значение `ncpus` полагалось равным числу OpenMP-потоков, флаг `--cpu_bind` принимал значения `cores` или `threads`.

§ 2. Сравнение различных вариантов привязки на тестовых задачах

Рассмотрим варианты задания привязок MPI-процессов к ядрам с помощью описанных выше средств. Тестирование проводилось на МВС-100К МСЦ РАН (каждый ВУ — два четырехядерных процессора Xeon 5365, 3GHz) с MVAPICH-1.0.1 и кластере X4 ИПМ УрО РАН (каждый ВУ — два четырехядерных процессора Xeon 5420, 2.66GHz) с промежуточным программным обеспечением Open MPI 1.3.1, MPICH2-1.0.8, SLURM-1.3.10.

2.1. Тесты NPB-MPI

На примере выполнения существующих не оптимизированных под многоядерную архитектуру тестовых задач NAS Parallel Benchmarks [5] рассмотрим результаты привязки MPI-приложений. Тесты NAS Parallel Benchmarks состоят из ряда задач: базовых приложений и псевдоприложений, эмулирующих вычисления на реальных задачах (в частности, в области вычислительной гидродинамики). Рассматривались задачи класса «A» и «B».

Приведем характеристики тестов по типу коммуникаций и объему пересылаемых данных (класс «A» для 16 MPI-процессов): EP — объем коммуникаций незначителен (коллективные обмены); BT — преобладают коммуникации точка-точка (объем данных 14.1 Гб); CG — коммуникации точка-точка (2.24 Гб); LU — основные коммуникации точка-точка (5.24 Гб); SP — преобладают коммуникации точка-точка (23.7 Гб); FT — только коллективные коммуникации (3.73 Гб); IS — преимущественно коллективные коммуникации (384 Мб).

Все эти приложения требуют оптимизации отображения при выполнении на многоядерном кластере, например: для тестов с преобладанием интенсивных вычислений, выполняемых на многоядерной системе, все ядра процессоров должны быть полностью загружены; для приложений с высокой коммуникационной нагрузкой необходима локализация коммуникаций в пределах ВУ (возможно, с учетом иерархии обменов) и использование многопоточности, что в целом обеспечит прирост производительности.

Рассмотрим использование привязки на примере запуска 2x8 (здесь и далее первая цифра определяет число ВУ, вторая — число процессов на ВУ) MPI-процессов (рис. 5, 6) тестов на кластере X4 (ИПМ УрО РАН) и вычислительной системе МВС-100К (МСЦ РАН). Для тестирования различных вариантов привязки была использована версия NPB 3.3 с MPI-реализацией тестовых задач класса «A» [5].

Таблица 1. Привязки MPI-процессов к логическим CPU для кластера X4 и МВС-100К

Вариант привязки (`bind`) (кластер X4/МВС-100К)	Значение `cpu_bind`/`VIADEV_CPU_MAPPING`	Распределение MPI-процессов по сокетам (0/1) ВУ
Файл `cpuinfo`	0,1,4,5,2,3,6,7 / 0,2,4,6,1,3,5,7	
def	без `cpu_bind`/`VIADEV_CPU_AFFINITY=0`	
c	`map_cpu:`0,2,1,3,4,6,5,7 / 0,1,2,3,4,5,6,7	0,2,4,6 / 1,3,5,7
b	`map_cpu:`0,1,4,5,2,3,6,7 / 0,2,4,6,1,3,5,7	0–3 / 4–7
b2	`map_cpu:`0,1,2,3,4,5,6,7 / 0,2,1,3,4,6,5,7	0,1,4,5 / 2,3,6,7
b4	`map_cpu:`7,0,6,1,3,4,2,5 / 7,0,5,2,3,4,1,6	1,3,5,7 / 0,2,4,6
b5	`map_cpu:`0,5,2,7,1,4,3,6 / 0,6,1,7,2,4,3,5	0,1,4,5 / 2,3,6,7
b6	`map_cpu:`0,2,4,6,1,3,5,7 / 0,1,4,5,2,3,6,7	0,2,4,6 / 1,3,5,7
b7	`map_cpu:`0,4,2,6,1,5,3,7 / 0,4,1,5,2,6,3,7	0,1,4,5 / 2,3,6,7
s /–	sockets/–	0,2,4,6 / 1,5,3,7 или 1,5,3,7 / 0,2,4,6
cor /–	cores/–	0,2,4,6 / 1,5,3,7
r /–	rank/–	0,1,4,5 / 2,3,6,7
t /–	threads/–	0,2,4,6 / 1,5,3,7

Результаты представлены в виде отношения производительности данного теста с привязкой R_{bind} (где `bind` принимает значение привязок из табл. 1) к производительности, измеренной с привязкой по умолчанию на данной вычислительной системе (R_{def}).

Необходимо отметить, что выбор той или иной привязки для конкретного теста может приводить как к повышению производительности, так и ее уменьшению. Для отдельного теста влияние привязки на разном коммуникационном обеспечении (аппаратное и программное) существенно отличается. Выделим следующие результаты: для сети Gigabit максимальное влияние привязки получено на тестах IS, CG, а на InfiniBand — EP, CG. С увеличением числа ВУ (рис. 7, 8) влияние привязки имеет примерно такой же вид.

На тесте IS максимальная производительность получена с привязкой `bind=b4`, при которой чётные и нечётные MPI-процессы располагались на разных сокетах.

Относительно теста CG можно отметить, что лучшие результаты получены при распределении MPI-процессов по привязке `bind=b` (см. табл. 1) в соответствии с физическими CPU (см. `cpuinfo`). Прирост производительности в тесте EP связан с его вычислительными, а не коммуникационными особенностями. Для остальных тестов требуется проведение более глубокого анализа тонкой структуры программы для объяснения влияния привязки.

2.2. Тесты NPB-MZ

В данном разделе рассматривается возможность повышения производительности гибридных MPI/OpenMP-приложений выполнения за счет привязки процессов. Для исследования была взята версия NPB-Multi-Zone 3.3, реализующая гибридную модель программирования MPI/OpenMP [6]. В Multi-Zone версии используется гибридный параллелизм: на грубом уровне распараллеливания — MPI и для распараллеливания циклов — OpenMP. Пакет содержит три приложения: BT-MZ, LU-MZ, SP-MZ. Для каждого теста определены число зон и их размеры. Задачи класса «B» содержат 64 зоны разного размера. Проведенные на кластере X4 тесты показали, что без привязки MPI/OpenMP-версия приложения SP-MZ (в котором размеры зон совпадают) существенно менее производительна, чем MPI-версия (см. рис. 9). Использование флага `--cpus-per-task` со значением, равным числу OpenMP-потоков, в сочетании с привязкой `cores` или `threads` позволило уменьшить разность в производительности до 10% .

В MPI-версии BT-MZ имеется существенная разбалансировка процессоров (размеры зон отличаются), поэтому использование гибридной MPI/OpenMP-модели может повлиять на рас-

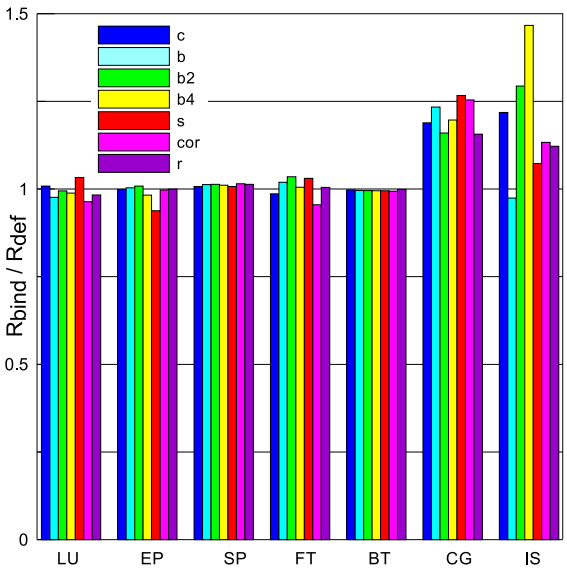

Рис. 5. Сравнение вариантов привязки для тестов NPB-MPI на сети Gigabit (2x8)

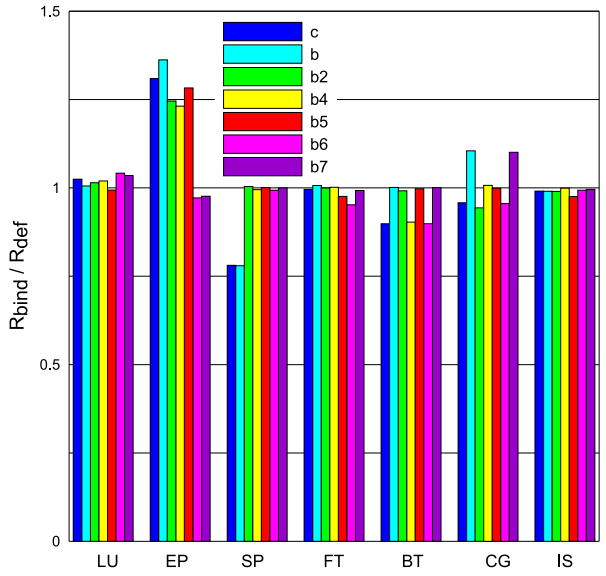

Рис. 6. Сравнение вариантов привязки для тестов NPB-MPI на сети InfiniBand (2x8)

пределение нагрузки. На тесте BT-MZ наиболее производительными были именно гибридные приложения. На рис. 10 представлены результаты использования различных вариантов привязок для данного теста (задача класса «B»). Приращение производительности $R_{bind} - R_{def}$ (по сравнению с вариантом без привязки R_{def}) отмечено изменением цвета столбца диаграммы.

В случае разных вычислительных систем наиболее эффективным для BT-MZ оказался вариант запуска (8x2x4) на каждом из восьми ВУ по два MPI-процесса с порождением в них четырёх OpenMP-потоков. Следует отметить, что на кластере X4 это был вариант привязки `--cpu_bind=sockets`, а на МВС-100К — `VIADEV_CPU_AFFINITY=0`.

Таким образом, наиболее эффективными для гибридного приложения оказались привязки к сокетам и ВУ, которые разрешали миграцию потоков-потомков на другие ядра в пределах ВУ. Более того, производительность (R, Моп./сек.) гибридного приложения на сети Gigabit Ethernet отличалась менее, чем на 25% по сравнению с полученной на сети InfiniBand.

Средствами Linux и PLPA было подтверждено, что при других вариантах привязки OpenMP-потоки не использовали ядра, к которым не были привязаны MPI-процессы, и, как следствие, получена существенно меньшая (в 2-3 раза) производительность гибридных приложений.

Таблица 2. SLURM + MPICH2-Nemesis (N=32381)

Привязка	CPU	Время, сек.	Ускорение, %
Без привязки	$4 \times 2(\times 2)$	204.30	
-B 1:1	$4 \times 2(\times 2)$	204.98	-0.33
cpu_bind=map_cpu:0,1,2,3	$4 \times 2(\times 2)$	198.37	2.9
cpu_bind=cores	$4 \times 2(\times 2)$	205.14	-0.41
cpu_bind=rank	$4 \times 2(\times 2)$	198.77	2.7
cpu_bind=sockets	$4 \times 2(\times 2)$	203.39	0.45

2.3. Тест LINPACK

Linpack является классическим примером теста-ядра (причем, поскольку к решению тех или иных СЛАУ сводятся очень многие реальные расчетные задачи измеренные им характеристики являются в высокой степени достоверными). Использовалась параллельная реализация

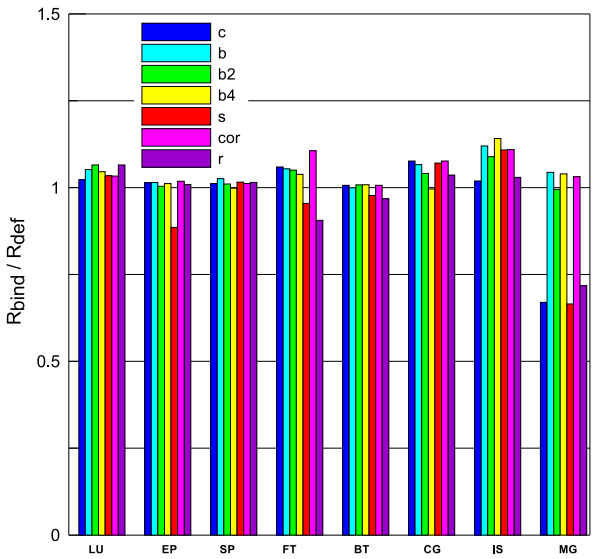

Рис. 7. Сравнение вариантов привязки для тестов NPB-MPI на сети Gigabit (8x8)

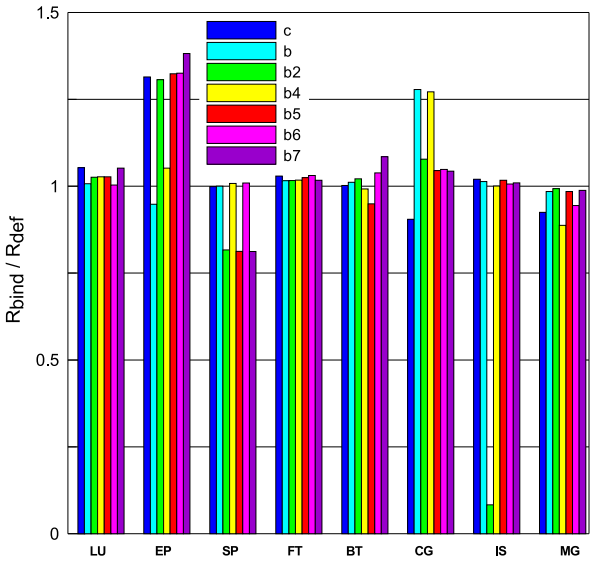

Рис. 8. Сравнение вариантов привязки для тестов NPB-MPI на сети InfiniBand (8x8)

теста LINPACK — HPL 2.0. Параметры теста были фиксированы (размер задачи $N = 32381$), а менялись только варианты привязки. Сравнение полученных результатов (таблица 2) с результатами (таблица 3), приведенными в работе [7] показало, что привязка дала меньшее ускорение (2.9% вместо 8.55%). Это связано с отличием не только в программном (SLURM + HP-MPI), но и в аппаратном обеспечении (в работе [7] применялись процессоры AMD Opteron).

Таблица 3. SLURM+HP-MPI (N=32381)

Привязка	CPU	Время, сек.	Ускорение, %
Без привязки	$4 \times 2(\times 2)$	467.16	
taskset 0xf	$4 \times 2(\times 2)$	481.83	-3.04
taskset 0x1; 0x2; 0x4; 0x8	$4 \times 2(\times 2)$	430.44	8.53
-B 1:1	$4 \times 2(\times 2)$	430.36	8.55
cpu_bind=map_cpu:0,1,2,3	$4 \times 2(\times 2)$	430.36	8.55

Для более точной оценки влияния привязки были проведены исследования на восьми ВУ и размере задачи $N = 81000$. Следует отметить, что при использовании собственных привязок Open MPI (из файлов *ranks* и *ranks*1 ,таблица 4) удалось ускорить вычисления на 19%.

§ 3. Заключение

В ходе исследования были рассмотрены несколько методов привязки MPI-процессов на различных уровнях (системное, промежуточное ПО). Следует отметить, что ПО SLURM обеспечивает наибольшую гибкость при задании привязок процессов/потоков к вычислительным узлам, процессорам, ядрам, что является критическим для гибридной модели.

Установлено, что некоторые привязки процессов обеспечивают существенное ускорение: 19% (тест HPLinpack) и 47% (тест IS в NPB-MPI), без модификации исходного кода MPI-приложения. Отметим, что для гибридных MPI/OpenMP-приложений ускорение за счет использования привязок достигается на уровне 17% для теста BT-MZ и 35% — SP-MZ.

Исследования показали, что для эффективного применения привязки существующего параллельного приложения к ядрам необходимо учитывать его карту связей (обменов), их интенсивность и используемое коммуникационное обеспечение (аппаратное и программное).

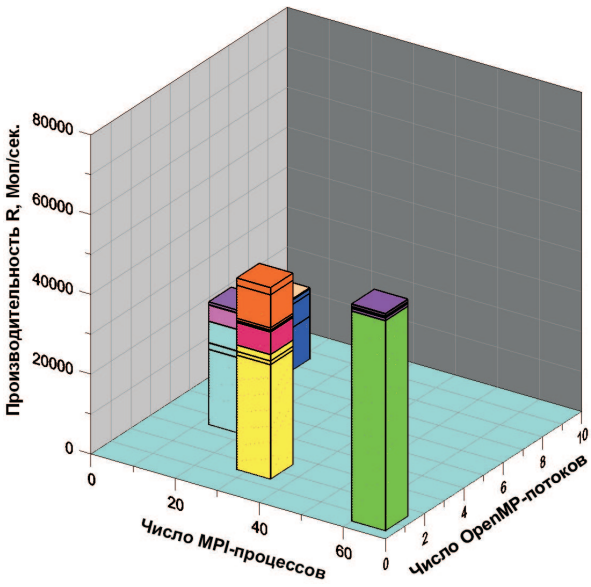

Рис. 9. Влияние привязки для теста SP-MZ (8xXxZ, Gigabit)

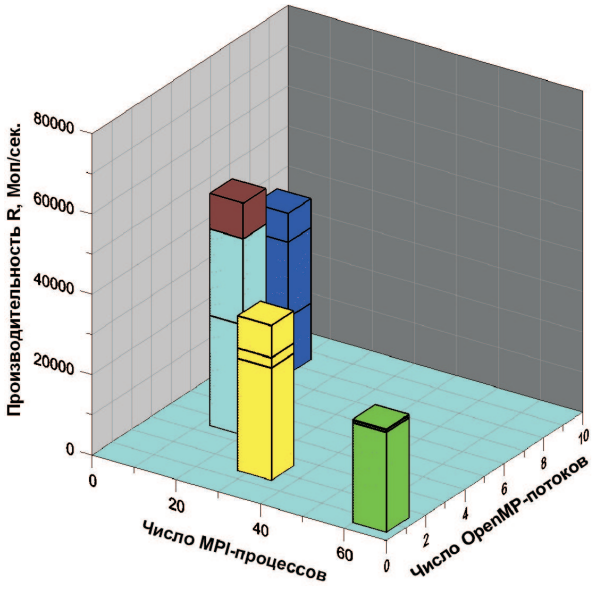

Рис. 10. Влияние привязки для теста BT-MZ (8xXxZ, Gigabit)

Таблица 4. N=81000

Привязка	CPU	Время, сек.	Ускорение, %
SLURM + MPICH2-Nemesis			
Без привязки	8 × 8	814.39	
cpu_bind=rank	8 × 8	815.26	-0.1
cpu_bind=sockets	8 × 8	815.89	-0.18
OpenMPI			
Без привязки	8 × 8	907.10	
–rankfile *ranks*[2]	8 × 8	777.43	16.67
–rankfile *ranks*1[3]	8 × 8	762.04	19.03

[2]Значения slot в файле *ranks* 0:0,0:2,1:0,1:2,0:1,0:3,1:1,1:3

[3]Значения slot в файле *ranks*1 0:0,0:1,0:2,0:3,1:0,1:1,1:2,1:3

С другой стороны, зная возможности привязки к ядрам ВУ, необходимо на этапе разработки параллельного приложения использовать отображение процессов и потоков на многоядерные процессоры, учитывая привязку к ядрам, сокетам и ВУ. Этот подход может применяться и к модернизации существующих MPI-программ (выделение коммуникационных процессов, переход к гибридной модели, использование динамического порождения процессов).

Кроме того, переход к многоядерным процессорам требует разработки нового системного, промежуточного и прикладного ПО, способного в полной мере задействовать все исполнительные механизмы многоядерных процессоров. Очевидно, что использование многоядерных процессоров оправдано лишь в хорошо распараллеленном приложении при соответствующей поддержке системного окружения.

СПИСОК ЛИТЕРАТУРЫ

1. Береснев Д. В., Копысов С. П., Новиков А. К., Тонков Л. Е. Привязка MPI-процессов к многоядерным узлам вычислительных систем // Параллельные вычислительные технологии (ПаВТ 2009): Труды Междунар. конф. Челябинск: ЮУрГУ, 2009. — С. 393–398.
2. Portable Linux Processor Affinity (PLPA)
`http://www.open-mpi.org/projects/plpa`
3. PathScale Compiler Suite
`http://www.pathscale.com`
4. Yoo A., Jette M. A., Grondona M. SLURM: Simple Linux Utility for Resource Management // Lecture Notes in Computer Science. — 2003. — Vol. 2862. P. 44–60.
5. Bailey D. The NAS Parallel Benchmarks // International Journal of Supercomputer Applications. — 1991. — Vol. 5, № 3. — P. 66–73.
6. Van der Wijngaart R. F. NAS Parallel Benchmarks. Multi-Zone Versions // NAS Technical Report NAS-03-010. NASA, 2003.
7. Balle S. M., Palermo D. J. Enhancing an Open Source Resource Manager with Multi-Core Multi-threaded Support // Lecture Notes in Computer Science. — 2007. — Vol. 4942. P. 37–50.

Поступила в редакцию 12.10.09

S. P. Kopysov, A. K. Novikov, L. E. Tonkov, D. B. Beresnev
Methods of a binding of parallel processes and threads for multicore computers

In operation ways and variants of a binding of parallel processes and threads to cores, sockets of computing systems are considered. Results of performance of test examples on MPI and MPI/OpenMP are resulted. Possibilities of reaching of desirable allocation of the parallel application on processors and kernels are discussed.

Keywords: parallel calculations, binding of processes, multicores processors, the middleware software

Mathematical Subject Classifications: 68T19, 68N30

Копысов Сергей Петрович, д. ф.-м. н., ведущий научный сотрудник, Учреждение РАН Институт прикладной механики УрО РАН, 426067, Россия, г. Ижевск, ул. Т. Барамзиной, 34,
E-mail: s.kopysov@gmail.com

Новиков Александр Константинович, к. ф.-м. н., старший научный сотрудник, Учреждение РАН Институт прикладной механики УрО РАН, 426067, Россия, г. Ижевск, ул. Т. Барамзиной, 34,
E-mail: an@udman.ru

Тонков Леонид Евгеньевич, к. ф.-м. н., старший научный сотрудник, Учреждение РАН Институт прикладной механики УрО РАН, 426067, Россия, г. Ижевск, ул. Т. Барамзиной, 34,
E-mail: tnk@udman.ru

Береснев Дмитрий Владимирович, ведущий специалист, Учреждение РАН Институт прикладной механики УрО РАН, 426067, Россия, г. Ижевск, ул. Т. Барамзиной, 34,
E-mail: dm@udman.ru