Привязка MPI-процессов к многоядерным узлам вычислительных систем*

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

В работе рассматриваются способы и варианты привязки MPI-процессов к ядрам, сокетам вычислительных узлов. Приводятся результаты выполнения тестовых примеров на MPI и MPI/OpenMP. Обсуждаются возможности достижения желаемого размещения приложения на процессорах и ядрах.

1. Введение

Сегодня только небольшая часть разработанного программного обеспечения может эффективно выполняться на многоядерных процессорах. В настоящее время актуально повышение производительности выполнения существующих MPI-программ. Применение многоядерных процессоров делает возможным получение более совершенных распределений процессов для задействования ядер с целью 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.

При выполнении привязки процессов к ядрам необходимо учитывать некоторую комбинацию аппаратного обеспечения, операционной системы и промежуточного программного обеспечения. Фактически нумерация ядер изменяется для разных систем и ее нельзя наследовать для достижения омтимальных результатов на всех системах. Пользователи должны знать отображение ядер для получения желаемого размещения на процессорах и ядрах.

2. Способы задания привязки

Промежуточное обеспечение MPI не поддерживает полного связывания с CPU, поэтому необходимы системы позволяющие связывать процессы при выполнении `mpirun`. Рассмотрим два способа привязки, отличающиеся вариантами задания и взаимодействием с промежуточным программным обеспечением (различные MPI-реализации, планировщики).

2.1. 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) [1]. 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.

2.2. Промежуточное программное обеспечение SLURM

В настоящее время не все менеджеры ресурсов 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

*Работа поддержана РЦП Уральского отделения РАН

Management) [2, 3], включающая компоненты: анализа 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 ресурсов 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.

В SLURM для обычных пользователей доступна привязка их 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 устнавливает 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` приходящиеся на каждый вычислительный узел, сокет, ядро. Для многопоточных приложений применяется флаг `--cpus-per-task=ncpus`, выделяющий `ncpus` ядер на процесс.

3. Привязка MPI-процессов к ядрам

Рассмотрим варианты задания привязок MPI-процессов к ядрам с помощью рассмотренных выше средств. Тестирование проводилось на МВС-100К МСЦ РАН (каждый ВУ — два четырехядерных процессора Xeon 5365, 3GHz) с MVAPICH-1.0.1 и кластере Х4 ИПМ УрО РАН (каждый ВУ — два четырехядерных процессора Xeon 5420, 2.66GHz) с промежуточным программным обеспечением MPICH2-1.0.8, SLURM-1.3.10.

3.1. Взаимодействие MPICH2 с планировщиком SLURM

На кластере Х4 ИПМ УрО РАН задания запускаются в окружении SLURM с MPICH2. Запуск приложения осуществляется входящей в SLURM командой `srun`:

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

Здесь `machines` — файл с именами вычислительных узлов. 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).

3.2. Взаимодействие MVAPICH с библиотекой PLPA

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).

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

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

4. Привязка к ядрам в гибридной модели MPI/OpenMP

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 2-х 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=8`. Для осуществления миграции OpenMP-потоков на свободные ядра вычислительного узла использовалась 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`.

5. Тестирование различных вариантов привязки на тестах NPB

На примере выполнения существующих не оптимизированных под многоядерную архитектуру тестовых задач NAS Parallel Benchmarks [4] рассмотрим результаты привязки

MPI-приложений. Тесты NAS Parallel Benchmarks состоят из ряда задач: базовых приложений и псевдо-приложений, 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.

5.1. Тесты NPB-MPI

Рассмотрим использование привязки на примере запуска 2x8 (здесь и далее первая цифра определяет число ВУ, вторая число процессов на ВУ) MPI-процессов (рис. 1, 2) тестов на кластере X4 (ИПМ УрО РАН) и вычислительной системе МВС-100К (МСЦ РАН). 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}).

Для тестирования различных вариантов привязки была использована версия NPB 3.3 с MPI-реализацией тестовых задач класса "A" [4]. Необходимо отметить, что выбор той

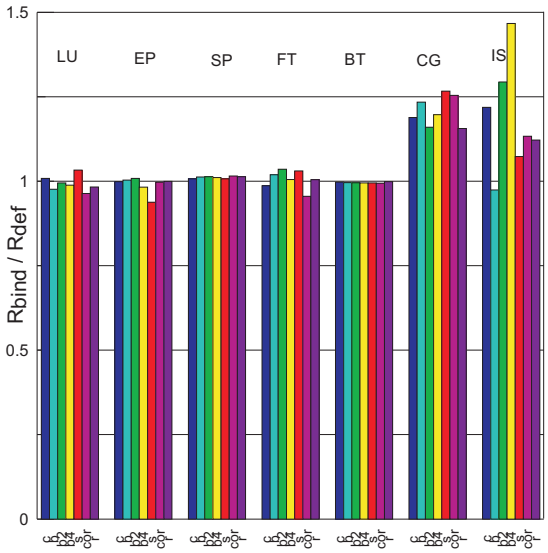

Рис. 1. Сравнение вариантов привязки для тестов NPB-MPI на сети Gigabit (2x8)

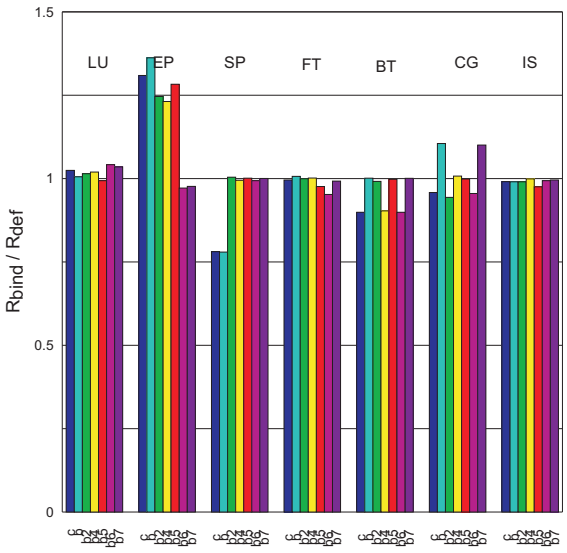

Рис. 2. Сравнение вариантов привязки для тестов NPB-MPI на сети InfiniBand (2x8)

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. 3, 4) влияние

привязки имеет примерно такой же вид. На тесте IS максимальная производительность по-

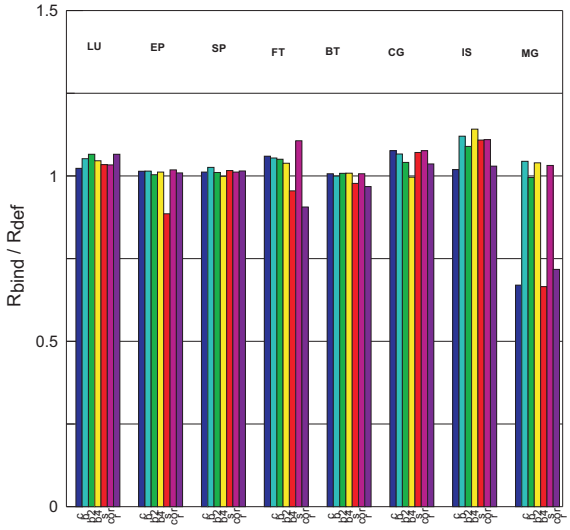

Рис. 3. Сравнение вариантов привязки для тестов NPB-MPI на сети Gigabit (8x8)

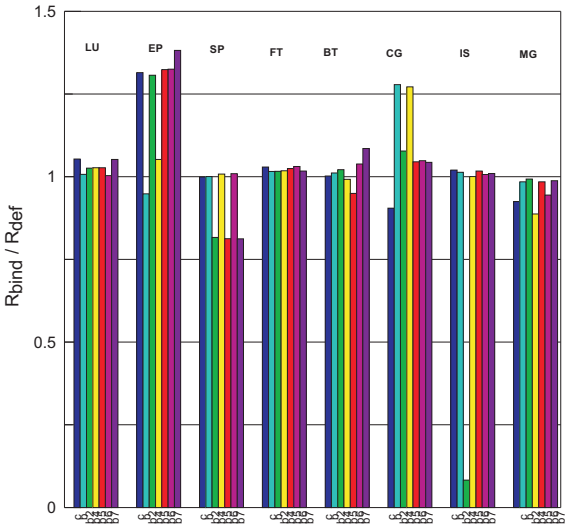

Рис. 4. Сравнение вариантов привязки для тестов NPB-MPI на сети InfiniBand (8x8)

лучена 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, чтобы объяснить влияние привязки.

5.2. 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 [5]. 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 тесты показали, что без привязки MPI/OpenMP-версия приложения SP-MZ (в котором размеры зон совпадают) существенно менее производительна, чем MPI-версия (см. рис. 5). Использование флага `--cpus-per-task` со значением равным числу OpenMP-потоков и привязок `cores`, `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 распределение нагрузки. На тесте BT-MZ наиболее производительными были именно гибридные приложения. На рис. 6 представлены результаты использования различных вариантов привязок для данного теста (задача класса "B") только на кластере X4. 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

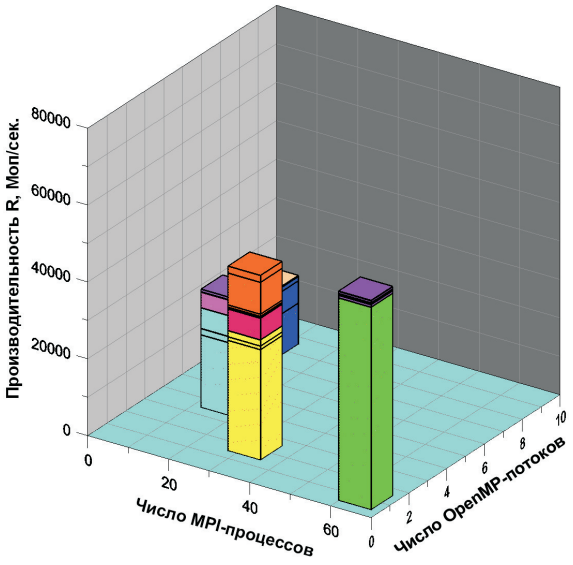

Рис. 5. Влияние привязки для теста SP-MZ (8xXxZ, Gigabit)

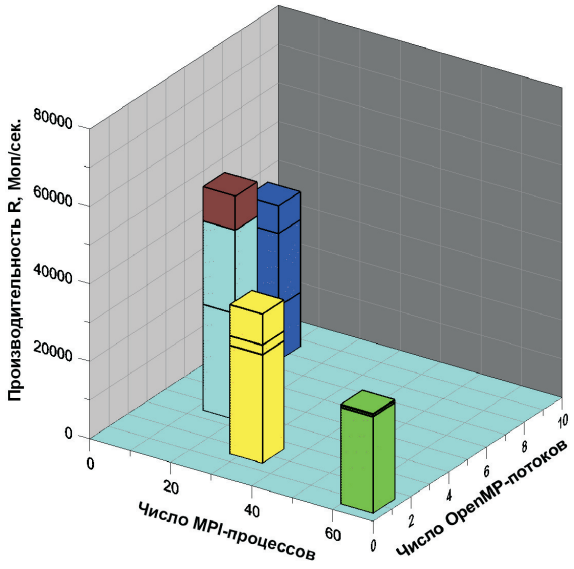

Рис. 6. Влияние привязки для теста BT-MZ (8xXxZ, Gigabit)

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, полученная 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, существенно меньшая (в 2-3 раза) производительность гибридных приложений.

Важно отметить, что ПО SLURM обеспечивает наибольшую гибкость при задании привязок по ВУ, сокетам, ядрам, что оказалось критическим для гибридной модели.

Исследования показали, что для эффективного применение привязки существующего параллельного приложения к ядрам необходимо учитывать его карту связей (обменов), их интенсивность и используемое коммуникационное обеспечение (аппаратное и программное). С другой стороны, зная возможности привязки к ядрам ВУ, необходимо на этапе разработки приложения использовать отображение процессов и потоков на физические процессоры.

Литература

1. Portable Linux Processor Affinity (PLPA): [http://www.open-mpi.org/projects/plpa]

2. Yoo A. SLURM: Simple Linux Utility for Resource Management // Lecture Notes in Computer Science. — 2003. — Vol. 2862. — P. 44–60.

3. Jette M. SLURM Home Page: [http://www.llnl.gov/linux/slurm], 2004.

4. Bailey D. The NAS Parallel Benchmarks // International Journal of Supercomputer Applications. — 1991. — Vol. 5, N 3. — P. 66–73.

5. Van der Wijngaart R. F. NAS Parallel Benchmarks. Multi-Zone Versions // NAS Technical Report NAS-03-010. NASA, 2003.