УДК 519.6

РЕАЛИЗАЦИЯ СРЕДСТВ ПРИВЯЗКИ ПРОЦЕССОВ В БИБЛИОТЕКЕ MPI ДЛЯ ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ ПАРАЛЛЕЛЬНЫХ ПРИЛОЖЕНИЙ

А. В. Байдураев, С. И. Сапронов
(ООО "Центр компетенций и обучения", г. Саров)

Описываются оригинальные программные средства, обеспечивающие закрепление за каждым процессом приложения определенного набора вычислительных ресурсов во время выполнения. Эти программные средства встраиваются в библиотеку MPI и позволяют эффективно влиять на производительность параллельных приложений.

Ключевые слова: MPI, параллельные вычисления, закрепление процессов.

Введение

Внедрение перспективной концепции многоядерной архитектуры процессоров повлекло за собой быстрое развитие средств информационного и управляющего обеспечения этой концепции в реальном секторе применения — высокопроизводительных вычислительных кластерных системах. Все значимые реализации MPI [1] (Intel MPI [2], IBM Platform MPI [3], MPICH [4], Open MPI [5], и др.), предоставляющие программный сервис для кластерных систем, в той или иной степени обеспечивают пользователю средства доступа к инфраструктуре вычислительных систем с целью эффективного использования специфики многоядерных архитектур. Удобство применения этих средств во многом определяет привлекательность конкретной реализации MPI.

Средства привязки процессов в разрабатываемом программном комплексе S-MPI [6] предназначены для обеспечения закрепления за процессами и его потоками определенного набора вычислительных ресурсов во время выполнения приложения.

Функционирование средств привязки

Описываемые программные средства достаточно автономны и могут быть интегрированы в конкретную реализацию MPI в качестве дополнительной библиотеки или встроены непосредственно в код без значительных трудозатрат. Основное требование к исполнительной среде реализации — это предоставление данных о распределении процессов приложения по узлам кластера на стадии запуска.

Средства привязки собирают необходимую информацию о требуемых приложением и имеющихся в наличии ресурсах, составе кластера и топологии каждого вычислительного узла, а также распределении MPI-процессов по узлам. При этом топологическая информация приводится к некоторому унифицированному виду с целью абстрагирования от специфики различных многоядерных архитектур. Во время выполнения приложения собранная информация предоставляется различным компонентам библиотеки MPI, которые могут использовать ее в целях повышения производительности приложения.

Закрепление процессов само по себе может способствовать более эффективному исполнению приложения ввиду отсутствия миграции процесса с ядра на ядро и связанных с этим накладных расходов или, например, возможности *разведения* процессов, конкурирующих за общие ресурсы.

Пользовательский интерфейс средств привязки

Совокупность вычислительных ресурсов, выделяемых отдельному процессу, именуется до-

меном. Домен характеризуется числом счетных элементов (физических или логических ядер) и способом размещения этих элементов внутри узла. Вводится понятие *расстояния* между отдельными счетными элементами, основанное на количестве разделяемых этими элементами ресурсов (сокетов, кэшей, шин передачи данных). Чем больше общих ресурсов, тем расстояние между элементами меньше.

Пример распределения доменов для четырех ядер приведен на рис 1. В данном примере домен содержит два ядра с общим кэшем второго уровня, поэтому имеется возможность запустить в рамках процесса два потока, которые будут наиболее эффективно использовать общие вычислительные ресурсы.

Управление привязкой осуществляется пользователем посредством установки специальных переменных среды или задания опций в командной строке запуска приложения. Отдельной опции соответствует одна переменная среды. Опции командной строки имеют более высокий приоритет по сравнению с переменными среды.

Каждая опция или переменная среды предназначена для выполнения какой-либо функции закрепления: установки режимов работы, определения набора доменов, задания схем отображения множества процессов на множество процессоров.

Отображение множества процессов на домены может состоять из следующих шагов:

1) упорядочение заданного набора доменов;
2) циклический сдвиг на указанный шаг по доменам;
3) дополнительный сдвиг на указанное количество доменов.

В случае, когда какой-либо шаг не указывается, выполняется действие по умолчанию.

Переменные среды или опции определяют:

- разрешающую способность привязки (физическое или логическое ядро);
- характеристики доменов;
- карту привязки процессов к доменам.

Библиотечные средства привязки транслируют требования пользователя во внутрисистемные вызовы, исполняя которые операционная система обеспечивает выполнение каждого процесса на строго определенном подмножестве ресурсов.

Эффект привязки при обменах через разделяемую память

Передача MPI-сообщений через разделяемую память представляет собой копирование блоков данных из памяти одного MPI-процесса в память другого процесса посредством промежуточного буфера (очереди) в разделяемой памяти (рис. 2).

Один из основных методов оптимизации подобных передач состоит в том, чтобы при передачах сообщений учитывать топологию многоядерной архитектуры и, в частности, знать общие ресурсы, которые используют (*разделяют*) MPI-процессы. Например, необходимо знать, как будет осуществляться доступ к промежуточному буферу: будут ли данные буфера располагаться в процессорном кэше или передаваться только по шине данных. Для исключения накладных расходов на получение этой информации перед каждой

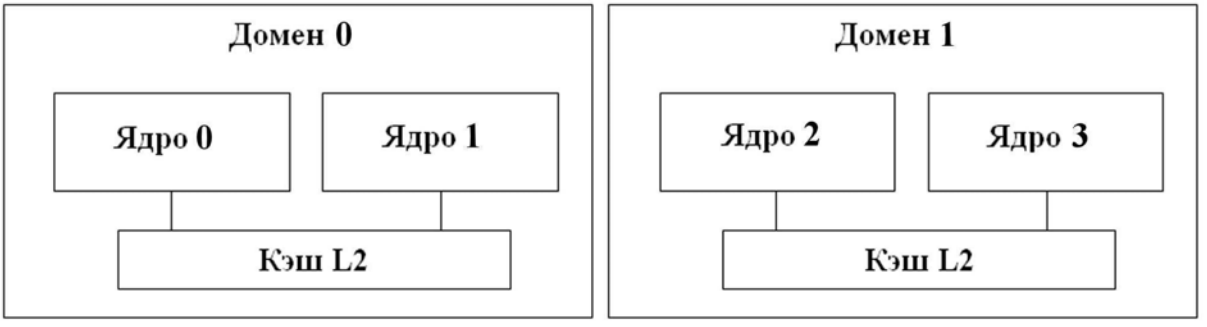

Рис. 1. Пример распределения доменов для четырех ядер

Рис. 2. Схема передачи сообщений через разделяемую память

передачей необходимо, чтобы при инициализации все MPI-процессы были закреплены на определенных процессорных ядрах, а информация о закреплении всех процессов на локальном узле была доступна каждому процессу на этом узле. При передаче данных между процессами, используя информацию о закреплении процессов и доступности общих ресурсов для каждой пары процессов, можно выбрать наиболее оптимальную процедуру копирования.

На рис. 3 приведен пример подобной оптимизации. За каждым MPI-процессом закреплено свое ядро, причем ядра попарно разделяют кэш второго уровня. Поэтому в случае передачи сообщений от процесса 0 к процессу 2 (когда процессы не разделяют общий кэш) выгодно, начиная с определенного порога размеров сообщений, копировать данные без использования кэша. То есть при копировании лучше использовать инструкции процессора, которые загружают данные непосредственно из памяти, минуя кэш (так называемые *non-temporal*-инструкции). В случае передачи сообщений от процесса 0 к процессу 1 (когда оба процесса разделяют кэш) даже большие сообщения размером до нескольких мегабайт выгодно копировать с использованием инструкций процессора, которые загружают сначала данные из кэша и только потом из памяти, если данных нет в кэше.

На рис. 4 приведен график для теста NetPIPE [7], демонстрирующий рост производительности при использовании оптимизации для случая, когда процессы не разделяют кэш. Заметим, что в данном случае оптимизация копирования используется для сообщений размером $l > 16$ кбайт.

Заключение

Исследования повышения производительности параллельных приложений за счет реализации средств привязки процессов в MPI были выполнены авторами на базе открытого кода Open MPI [5] в рамках контракта (№ 07.524.12.4020) с Министерством образования и науки РФ.

Дальнейшее развитие описанных средств предполагается осуществлять в направлении создания гибких унифицированных программных интерфейсов, допускающих несложную интеграцию в различные реализации MPI и осуществляющих информационный и управляющий сервис для всех вычислительных и коммуникационных ресурсов кластера.

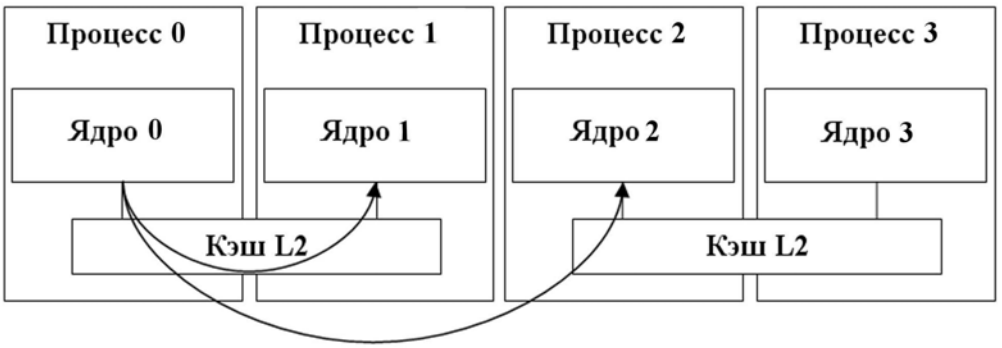

Рис. 3. Схема копирования сообщений через разделяемую память с использованием информации о топологии многоядерной архитектуры

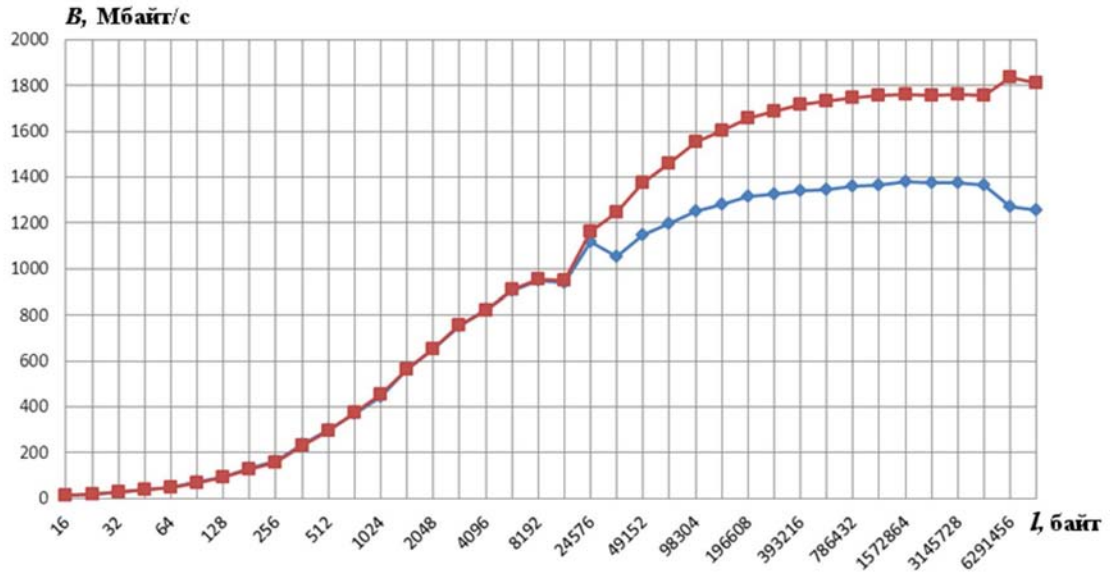

Рис. 4. Пропускная способность общей памяти для процессов, не разделяющих кэш: —◆— — без оптимизации; —■— — с оптимизацией

Список литературы

1. *Message Passing Interface Forum.* MPI: A Message Passing Interface // Proc. of "Supercomputing'93". Los Alamitos: IEEE Computer Society Press, 1993. P. 878—883.
2. Intel MPI. http://software.intel.com/en-us/intel-mpi-library.
3. IBM Platform MPI. http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/mpi/index.html.
4. MPICH. http://www.mpich.org/.
5. *Gabriel E., Fagg G. E., Bosilca G. et al.* Open MPI: goals, concept, and design of a next generation MPI implementation // Proc. 11th European PVM/MPI Users' Group Meeting. Budapest, Hungary. September 2004. P. 97—104.
6. *Воронов Г. И., Трущин В. Д., Шумилин В. В., Ежов Д. В.* Программный комплекс S-MPI для обеспечения разработки, оптимизации и выполнения высокопараллельных приложений на суперкомпьютерных кластерных системах // См. настоящий выпуск. С. 55—60.
7. NetPIPE. http://bitspjoule.org/netpipe.

Статья поступила в редакцию 07.12.12.