ПРОГРАММИРОВАНИЕ, 2019, № 3, с. 43–56

ЯЗЫКИ, КОМПИЛЯТОРЫ И СИСТЕМЫ ПРОГРАММИРОВАНИЯ

УДК 004.432

DVM-ПОДХОД К АВТОМАТИЗАЦИИ РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ ДЛЯ КЛАСТЕРОВ

© 2019 г. В. А. Бахтин[a,b,*], В. А. Крюков[a,b,]**

[a] *Институт прикладной математики им. М.В. Келдыша Российской академии наук*
125047 Москва, Миусская пл., 4, Россия

[b] *Московский государственный университет имени М.В. Ломоносова*
119991 Москва, Ленинские горы, д. 1, Россия

** E-mail: bakhtin@keldysh.ru*

*** E-mail: krukov@keldysh.ru*

Поступила в редакцию 11.01.2019 г.
После доработки 11.01.2019 г.
Принята к публикации 15.01.2019 г.

В статье описан DVM-подход для разработки параллельных программ для гетерогенных вычислительных кластеров с ускорителями. Рассматриваются основные возможности систем DVM и САПФОР, автоматизирующих процесс распараллеливания прикладных программ.

DOI: 10.1134/S0132347419030038

1. ВВЕДЕНИЕ

Используемые в настоящее время суперкомпьютерные системы можно разделить на 4 типа [1]:

– Суперкомпьютеры общего назначения. Предназначены для решения научно-технических задач, имеющих хорошую или среднюю пространственно-временную локализацию обращений к памяти. Где пространственная локализация предполагает, что данные, которые будут использованы приложением вскоре, расположены в памяти близко (по адресам) по отношению к адресам уже используемых данных. Временная локализация предполагает, что данные, которые используются сейчас, вскоре будут использованы вновь. Для актуальных задач, решаемых при помощи суперкомпьютерных систем, локализация ухудшается. Например, вместо работы с плотно заполненными матрицами сейчас характерна работа с разреженными матрицами.

– Суперкомпьютеры с огромной памятью, используемой в режиме интенсивных обращений к ней с плохой пространственно-временной локализацией. Типичные задачи: Big Data, моделирование функционирования сложных изделий и систем, искусственный интеллект.

– Суперкомпьютеры со встроенной основной памятью малого объема, обладающей малыми задержками и повышенной пропускной способностью. Такие системы обладают повышенной производительностью в сравнении с суперкомпьютерами общего назначения и предназначены для решения таких задач, как обработка сигналов и изображений, информационная безопасность, глубинное обучение.

– Суперкомпьютеры высшей производительности. Ориентированы на вычисления с хорошей пространственно-временной локализацией обращений к памяти. Для них важно наличие большой кэш-памяти, допускается низкий баланс пропускной способности памяти и производительности; характерны задачи работы с плотно заполненными матрицами. Возможности таких суперкомпьютеров адекватно оцениваются тестом Linpack (рейтинг Top500 [2]).

Каждый тип суперкомпьютеров обладает специфической элементной базой. Суперкомпьютеры общего назначения, как правило, представляют собой кластерные системы, в узлах которых используются многоядерные процессоры. Для систем 2-го типа характерно применение массово-мультитредовых и векторных микропроцессоров от Cray, NEC, NUDT. Для систем 3-го типа наиболее важны большие программируемые интегральные схемы от Xilinx и Altera, а также проблемно-ориентированные СБИС. В суперкомпьютерах высшей производительности, как правило, используются графические процессоры компаний NVIDIA и AMD, массово-многоядерные процессоры (сопроцессоры) Intel Xeon Phi.

Основные проблемы, которые возникают при использовании столь отличающихся по своим возможностям суперкомпьютерных систем: сложность разработки прикладного программного обеспече-

ния (далее ПО); переносимость разработанного прикладного ПО на различные типы суперкомпьютеров; надежность функционирования разработанного прикладного ПО при использовании тысяч узлов, десятков тысяч ядер и ускорителей; эффективность полученного параллельного ПО.

В настоящее время при разработке программ для высокопроизводительных вычислений на современных кластерах широко используются следующие модели программирования – MPI (для отображения программы по узлам кластера), POSIX Threads (для отображения программы по ядрам процессора), CUDA и OpenCL (для отображения программы по ядрам ускорителя). Все эти модели программирования являются низкоуровневыми. Для отображения программы на все уровни параллелизма программисту приходится использовать комбинацию из перечисленных моделей. Например, MPI+POSIX Threads+CUDA. Технически объединять низкоуровневые модели программирования, реализованные через библиотеки, проще, чем высокоуровневые модели, реализуемые посредством языков и соответствующих компиляторов. Но программировать, отлаживать, сопровождать, переносить на другие ЭВМ такие программы гораздо сложнее. Например, при переходе с графических процессоров фирмы NVIDIA на графические процессоры фирмы AMD придется заменить CUDA на OpenCL. Поэтому важно использовать высокоуровневые модели и языки программирования.

Среди высокоуровневых моделей программирования особое место занимают модели, реализуемые посредством добавления в программы на стандартных последовательных языках спецификаций, управляющих отображением этих программ на параллельные машины. Эти спецификации, оформляемые в виде комментариев в Фортран-программах или директив компилятору (прагм) в программах на языках Си и Си++, не видны для обычных компиляторов, что значительно упрощает внедрение новых моделей параллельного программирования. Примером такой модели для многоядерных процессоров и SMP-систем является модель OpenMP. В июле 2013 года было предложено расширение модели OpenMP [3] для использования ускорителей (OpenMP версии 4.0). Это расширение позволяет описать параллелизм внутри одного узла кластера: задействовать ядра центрального процессора и ядра ускорителей. Аналогичный подход был реализован совместными усилиями компаний Cray, CAPS, NVIDIA и PGI. Разработан стандарт OpenACC [4], который описывает набор директив компилятора, предназначенных для упрощения создания гетерогенных параллельных программ, задействующих как центральный, так и графический процессор. Использование высокоуровневых спецификаций позволяет программисту абстрагироваться от особенностей графического процессора, вопросов передачи данных и т.п. С появлением новых высокоуровневых моделей (OpenMP версии 4.0 и OpenACC) процесс разработки параллельных программ намного упростился. Теперь вместо 3-х моделей можно использовать только две. Например, MPI+OpenMP или MPI+OpenACC.

Но следует отметить, что существует множество альтернатив MPI. Например, модель SHMEM (Symmetric Hierarchical MEMory), в которой реализована поддержка односторонних коммуникаций. В этой модели чтобы данные были переданы из процесса А в процесс Б, не требуется согласованная активность обоих процессов, как в MPI, а лишь “желание” одного из участников. Например, процесс А может послать данные процессу Б без какой-либо ответной активности с его стороны. Процесс Б, в свою очередь, может прочитать данные из процесса А. Такие односторонние коммуникации поддерживаются во многих аппаратных решениях компаний Mellanox, Quadrics, QLogic и др. Для эффективного использования новых возможностей аппаратуры программисту снова придется переписывать свою программу (например, с MPI+OpenMP на SHMEM+OpenMP), что потребует от него значительных усилий, т.к. обе модели – и MPI, и SHMEM являются низкоуровневыми.

В 2011 году в ИПМ им. М.В. Келдыша РАН была разработана высокоуровневая модель программирования DVMH [5] (Distributed Virtual Memory for Heterogeneous Systems), которая позволяет создавать программы для гетерогенных вычислительных кластеров с различными ускорителями. Использование данной модели позволяет отказаться от использования MPI и SHMEM. Прикладные программы, реализованные в данной модели (далее DVMH-программы), не требуют изменений при их переносе с одной вычислительной системы на другую. Задача эффективного отображения программы на все вычислительные устройства суперкомпьютера решается DVMH-компиляторами и системой поддержки выполнения DVMH-программ.

Использование моделей типа OpenMP, OpenACC, DVMH серьезно упрощает процесс разработки параллельных программ. Компиляторы с языков OpenMP, OpenACC и DVMH выполняют сложную работу по отображению программ на параллельную ЭВМ. Но для того чтобы воспользоваться данной помощью, программисту требуется выполнить не менее сложную работу по определению и описанию свойств программы, которые важны для такого отображения: найти циклы, витки которых можно выполнять в любом порядке; определить циклы, между витками которых существует зависимость, но данная зависимость может быть устранена в результате преоб-

разования текста программы; для отображения на кластер, важно найти массивы, которые могут быть распределены по узлам кластера, задать способ распределения данных, определить необходимые обмены данными между узлами кластера и многое другое. Даже для программистов, использующих высокоуровневые модели программирования, требуются инструментальные средства, которые выполняют анализ и определяют свойства программ, необходимые для их отображения на параллельные ЭВМ.

В последние годы разработано множество систем, автоматизирующих процесс разработки параллельных программ для мультипроцессора. Примерами таких систем могут быть промышленные компиляторы компаний Intel, PGI; системы PLUTO, Polaris, SUIF, VAST/Parallel, OSCAR и другие. Среди отечественных разработок можно отметить компилятор для платформы Эльбрус. В данных инструментах поддержана возможность автоматического распараллеливания программ по ядрам процессора.

Существуют системы, которые автоматизируют отображение программ на графические ускорители. Примерами таких систем могут быть: Par4all, KernelGen, OMP2HMPP, Cetus.

Если распараллеливание программ для мультипроцессоров и графических ускорителей можно выполнять инкрементально (постепенно, цикл за циклом), то для эффективного распараллеливания программ на кластер требуется находить глобальные решения по распределению данных и вычислений, что существенно усложняет процесс разработки таких систем. Систем, автоматизирующих процесс разработки параллельных программ для кластеров, заметно меньше: CAPTools/Parawise, FORGE Magic/DM, BERT77. Среди отечественных разработок можно отметить систему ДВОР (Диалоговый высокоуровневый оптимизирующий распараллеливатель программ), в которой реализован экспериментальный режим распараллеливания Си-программ с использованием MPI.

В настоящее время существует всего лишь одна система автоматизированного распараллеливания, в которой качестве целевой архитектуры рассматривается гетерогенный кластер с ускорителями – система САПФОР [6], которая разрабатывется в ИПМ им. М.В. Келдыша РАН при активном участии студентов и аспирантов МГУ им. М.В. Ломоносова.

В данной статье рассматривается DVM-подход для разработки прикладного программного обеспечения суперкомпьютерных систем с ускорителями, который основан на использовании возможностей системы САПФОР и системы DVM [7].

2. DVM-ПОДХОД ДЛЯ РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

Для разработки параллельных программ для гибридных суперкомпьютерных систем с ускорителями предлагается использовать следующий подход.

На первом шаге пользователь подает свою программу на языке Fortran или Си на вход системе автоматизации разработки параллельных программ (далее системе САПФОР). Блок анализа системы переводит программу во внутреннее представление. Программа анализируется с помощью различных алгоритмов статического анализа, в результате которого выявляются особенности программы, влияющие на ее распараллеливание. Для всех циклов программы, в том числе неявных, собирается информация следующего вида: зависимости по данным (FLOW, ANTI, OUTPUT), регулярные зависимости по данным (для массивов), редукционные переменные, приватные переменные (скаляры и массивы), индукционные переменные, прогнозируется время последовательного выполнения циклов, определяются самые значимые циклы. Пользователь может в диалоговом режиме просматривать результаты анализа и получать разъяснения по обнаруженным особенностям.

Затем пользователь переходит ко второму шагу – нахождению подходящих схем распараллеливания программы (распределение данных и вычислений, организация доступа к данным, расположенным на других процессорах и.т.п.). Эти действия осуществляются для выбранной пользователем параллельной ЭВМ. Выбор/задание параллельной ЭВМ осуществляется через диалоговую оболочку. Пользователь может определить тип вычислительной системы, для которой разрабатывается параллельная программа: кластер, многоядерный кластер, кластер с ускорителями; указать количество узлов кластера; число и производительность ядер центрального процессора; количество и тип установленных ускорителей; задать коммуникационные характеристики сети: топологию, латентность, пропускную способность каналов и др. Система САПФОР должна построить наилучший вариант параллельной программы для заданной ЭВМ. Отбор подходящих схем распараллеливания осуществляется на основе прогнозируемых характеристик эффективности выполнения параллельной программы, получаемой для конкретной схемы распараллеливания на указанной ЭВМ и при заданных пользователем параметрах решаемой задачи (значениях переменных, от которых зависят размеры массивов, количество витков циклов и.т.п.). Прогнозирование эффективности заключается в моделировании выполнения программы для этой ЭВМ по внутреннему представлению программы. Система может предлагать пользователю на выбор разные схемы рас-

параллеливания, сопровождая их прогнозируемыми характеристиками эффективности и причинами отказа от распараллеливания фрагментов программы. Пользователь определяет значимые фрагменты программы и исследует причины отказа от их распараллеливания. Он может уточнить результаты автоматического анализа (например, используя дополнительные спецификации) и повторить второй шаг, и/или преобразовать текст последовательной программы и начать с первого шага.

Для уточнения результатов анализа может также использоваться динамический анализ (например, для автоматического определения значений параметров задачи, времени выполнения витков цикла, наличия/отсутствия зависимостей). Некоторые преобразования исходной программы могут быть также выполнены системой автоматически (например, инлайн подстановка процедур, разбиение/слияние циклов, приведение циклов к тесно-вложенному виду и.т.д.).

Если второй шаг для данной ЭВМ и данной задачи выполнен успешно (нашлись приемлемые по эффективности схемы распараллеливания), то пользователь может повторить его для другой ЭВМ, для других параметров задачи, и т.д., пока не исчерпается список интересующих его ЭВМ и задач.

По отобранным пользователем схемам распараллеливания и внутреннему представлению программы, система САПФОР генерирует текст параллельной программы в модели DVMH на языке Fortran-DVMH [8] или C-DVMH [9].

3. DVMH-МОДЕЛЬ ПРОГРАММИРОВАНИЯ И ПАРАЛЛЕЛИЗМА. ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Модель программирования DVMH позволяет разрабатывать параллельные программы для кластеров, в узлах которых помимо универсальных многоядерных процессоров установлены ускорители компании NVIDIA и сопроцессоры Intel Xeon Phi. Модель поддерживает использование всех перечисленных архитектур, как по отдельности, так и одновременно в рамках одной программы.

Модель параллелизма базируется на специальной форме параллелизма по данным: одна программа-множество потоков данных (ОПМД). В этой модели одна и та же программа выполняется на всех процессорах, но каждый процессор выполняет свое подмножество операторов в соответствии с распределением данных.

Определяются массивы и витки циклов, которые должны быть распределены между процессорами. Распределенные массивы специфицируются директивами отображения данных, а циклы – директивами распределения вычислений. Остальные переменные отображаются по одному экземпляру на каждый процессор (размноженные данные).

Распределение данных определяет множество локальных или собственных переменных для каждого процессора. Множество собственных переменных определяет правило собственных вычислений: процессор присваивает значения только собственным переменным.

При вычислении значения собственной переменной процессору могут потребоваться как значения собственных переменных, так и значения несобственных (удаленных) переменных. Для организации доступа к удаленным данным служат специальные директивы (SHADOW, REMOTE_ACCESS, ACROSS и др.).

Определяются фрагменты кода, которые могут быть выполнены на ускорителях. Такие фрагменты называются вычислительными регионами или просто регионами. Регион может выполняться на одном или нескольких ускорителях одного или разных типов и/или на центральном процессоре.

Фрагменты программы вне регионов всегда выполняются на центральном процессоре. Для каждого региона указываются данные, необходимые для его выполнения (входные, выходные, локальные). Перемещения данных между вычислительными регионами осуществляются в основном автоматически в соответствии с имеющейся в описании регионов информацией об используемых ими данных. Для управления перемещениями данных между регионами и фрагментами программы вне регионов предусмотрены специальные директивы.

Параллелизм в DVMH-программах проявляется на нескольких уровнях:

1) Распределение данных и вычислений по процессорам. Этот уровень задается директивами распределения и перераспределения данных и циклов.

2) Распределение данных и вычислений по вычислительным устройствам при входе в вычислительный регион.

3) Параллельная обработка в рамках конкретного вычислительного устройства. Этот уровень появляется при входе в параллельный цикл, находящийся внутри вычислительного региона.

Наличие этих уровней дает возможность эффективно отобразить DVMH-программу на кластер с многоядерными процессорами и ускорителями в узлах.

Языки Fortran-DVMH и C-DVMH представляют собой расширение стандартных языков Fortran 95 и Си 99 в соответствии с моделью DVMH. При использовании данных языков программист имеет один вариант программы и для последовательного, и для параллельного выполнения. Программа, помимо описания алгоритма обычными

средствами языка, содержит правила параллельного выполнения этого алгоритма. Правила оформляются синтаксически таким образом, что они являются "невидимыми" для стандартных компиляторов с языка Fortran и Си для последовательных ЭВМ и не препятствуют выполнению и отладке DVMH-программы на рабочих станциях как обычной последовательной программы.

4. DVMH-КОМПИЛЯТОРЫ И СИСТЕМА ПОДДЕРЖКИ ВЫПОЛНЕНИЯ DVMH-ПРОГРАММ

Для отображения DVMH-программы на параллельную ЭВМ используются специальные компиляторы, которые входят в состав DVM-системы.

DVMH-компилятор преобразует исходную программу в параллельную программу на языке Fortran или Си с вызовами функций системы поддержки параллельного выполнения DVMH-программ (библиотеки Lib-DVMH). Для каждого параллельного цикла компилятор генерирует процедуру-обработчик для вычислений на ускорителе (например, для графического процессора с использованием технологии CUDA) и процедуру-обработчик для выполнения этого параллельного цикла на центральном процессоре (например, для многоядерного процессора с использованием технологии OpenMP). Обработчик – это подпрограмма, которая осуществляет обработку части параллельного цикла на конкретном вычислительном устройстве. Аргументами обработчика являются описатель устройства и часть параллельного цикла. Обработчик запрашивает порцию для исполнения (границы цикла и шаг), конфигурацию параллельной обработки (количество нитей), инициализацию редукционных переменных, а после выполнения порции передает результат частичной редукции в систему поддержки.

Система поддержки выполнения DVMH-программ обеспечивает эффективное использование всех вычислительных устройств кластера. Основные функции системы поддержки:

1) Создание и уничтожение распределенного массива.

2) Отображение распределенного массива.

3) Отображение параллельного цикла.

4) Выполнение редукционных операций.

5) Обновление теневых граней распределенных массивов.

6) Создание и загрузка буферов для доступа к удаленным данным.

7) Выполнение операций ввода-вывода.

8) Определение состава ускорителей в момент запуска программы.

9) Регистрация фрагментов программы для выполнения на ускорителях (вычислительных регионов).

10) Сохранение информации об используемых в вычислительном регионе данных.

11) Выделение памяти на ускорителе под данные, используемые в регионе.

12) Копирование входных данных региона в память ускорителя.

13) Определение оптимальной конфигурации блоков нитей для циклов, выполняемых на ускорителе.

14) Копирование выходных данных региона из памяти ускорителя.

15) Регистрация данных, которые могут быть удалены из памяти ускорителя.

16) Освобождение памяти на ускорителе в случае ее нехватки для размещения требуемых данных.

Система поддержки реализована на основе стандартных технологий параллельного программирования MPI (для взаимодействия между узлами кластера), OpenMP (для распределения вычислений между ядрами узла), CUDA (для распределения вычислений и отображения данных на графические ускорители).

Основная задача компиляторов с языков Fortran-DVMH и C-DVMH – эффективное отображение DVMH-программы на кластеры с многоядерными процессорами и ускорителями в узлах. Это достигается как за счет различных оптимизаций, которые выполняются статически при компиляции программы, так и за счет динамических оптимизаций, которые выполняются во время работы программы. Рассмотрим некоторые из реализованных оптимизаций.

Для отображения данных в DVMH-модели реализованы различные виды распределений: блочное распределение (равными, неравными, взвешенными блоками), поэлементное распределение (косвенное и производное), а также распределение через выравнивание. Одна из проблем, которая может возникнуть при распределении данных – неоднородность вычислительных узлов кластера. Различные узлы кластера могут иметь разную производительность и архитектуру. Примером такого "неоднородного" кластера может стать кластер с сопроцессорами Intel Xeon Phi. Одним из возможных вариантов использования такого кластера является т.н. "симметричный режим". В этом режиме центральный процессор и сопроцессор можно рассматривать как отдельные вычислительные узлы. Одна и та же программа компилируется отдельно для центрального процессора и отдельно для сопроцессора. Скомпилированные программы одновременно запускаются на сопроцессоре и центральном процессоре и могут обмениваться данными с использованием механизма

передачи сообщений. При использовании данного режима необходимо сбалансировать нагрузку процессов, которые выполняются на различных вычислительных устройствах (процессоре и сопроцессоре).

Для сбалансированного распределения данных и вычислений в DVM-системе введена возможность измерения времен выполнения некоторого тестового программного фрагмента и получения по результатам измерений весов производительности узлов кластера. Измерение весов производительности осуществляется в самом начале работы DVMH-программы при запуске системы поддержки. Распределение данных осуществляется с учетом найденных весов (узел кластера, вес которого больше, получит большую порцию данных). Такая возможность позволяет DVMH-программам эффективно выполняться на неоднородных кластерах.

Для отображения массивов и циклов на вычислительные устройства узла кластера с учетом их производительности в системе поддержки выполнения DVMH-программ реализованы три режима распределения данных и вычислений в точках входа в вычислительные регионы:

1) Простой статический режим.

2) Динамический режим с подбором схемы распределения.

3) Статический режим с использованием подобранной схемы распределения.

Рассмотрим подробнее эти режимы распределения. В простом статическом режиме в каждом регионе распределение производится одинаково. При помощи переменных окружения пользователь задает вектор весов вычислительных устройств, имеющихся в каждом узле кластера (или они могут быть грубо определены автоматически), затем эти веса накладываются на параметры внешнего распределения данных. В таком режиме сводятся к минимуму перемещения данных в связи с их перераспределением, но не учитывается различное соотношение производительности вычислительных устройств на разных участках кода.

В основе режимов подбора лежит предположение об итерационности программы, т.е. повторении одних и тех же вычислений достаточно большое количество раз. В динамическом режиме с подбором схемы распределения в каждом регионе распределение выбирается на основе постоянно пополняющейся истории запусков данного региона и его соседей.

В процессе работы DVMH-программы ведут учет времени обработки всех параллельных циклов на используемых устройствах, поддерживая в актуальном состоянии табличную функцию зависимости производительности (в витках в секунду) от объема вычислений (в витках). После каждого выполнения цикла эта зависимость уточняется – новое измерение заменяет старые, находящиеся в заданной окрестности. Наличие и накопление этой информации позволяет оптимизировать веса распределения.

Построенная схема распределения может быть сохранена в файл и использована при последующих запусках программы. В качестве начального приближения может быть использован вектор весов вычислительных устройств, как и для простого статического режима.

Из данного режима возможен переход в третий режим в любой точке выполнения программы – статический режим с использованием подобранной схемы распределения. В этом режиме в каждом регионе распределение выбирается на основе предоставленной схемы распределения, построенной при работе программы во втором режиме, причем есть возможность как перейти в этот режим непосредственно из второго, так и использовать схему распределения из файла, полученного в результате работы программы во втором режиме. При использовании схемы распределения из файла не гарантируется ее корректное применение в случае, если параметры программы были изменены, в особенности это касается тех параметров, которые влияют на путь выполнения программы (выбор другого метода расчета, отключение или включение этапов расчета).

Одним из преимуществ DVMH-программ является то, что обращения к массивам в параллельных циклах, исполняемых на ускорителях, программируются в обобщенном виде, позволяющем во время выполнения иметь иной порядок измерений в массиве, нежели было задано в исходной программе. Данная особенность дает еще одну степень свободы для оптимизации во время выполнения – выбор представления массива при выполнении конкретного цикла. Например, в памяти вычислительного устройства массив расположен “по столбцам”, а обращения к элементам массива в цикле происходят “по строкам”, что приводит к существенному замедлению выполнения программы. Изменив расположение массива в памяти, можно существенно повысить эффективность выполнения программы. Такой механизм динамической реорганизации данных был реализован в системе поддержки выполнения DVMH-программ.

Для реализации этих возможностей система поддержки ведет учет текущего представления массива на всех вычислительных устройствах. Перед выполнением каждого цикла анализируется его правило отображения и сопоставляется с правилами выравнивания использующихся в нем массивов. Эта информация позволяет связать измерения цикла с измерениями массива и вывести оптимальный для данного цикла вид представления массива в памяти вычислительного устройства.

Для повышения эффективности использования графических ускорителей компании NVIDIA реализован механизм динамической компиляции, который позволяет компилировать CUDA-обработчики, сгенерированные DVMH-компиляторами во время выполнения программы.

В момент выполнения программы становятся известны значения многих скалярных переменных. Например, вместо передачи значения скалярной переменной в CUDA-ядро можно выполнить преобразование кода следующим образом: вместо передаваемого параметра, например, int param_1, объявить в теле CUDA-ядра переменную const int param_1 = dyn_value, где dyn_value – известное в момент компиляции значение данной переменной. В результате происходит сокращение числа регистров, требуемых для работы CUDA-ядра, что приводит к увеличению производительности данного ядра.

Описанные выше оптимизации повышают эффективность отображения DVMH-программ на кластеры с ускорителями. Получаемые программы могут динамически настраиваться при запуске на выделенные для их выполнения ресурсы (количество узлов кластера, ядер, ускорителей и их производительность) [10].

5. ОТЛАДКА DVMH-ПРОГРАММ

Как уже отмечалось ранее, система САПФОР активно использует диалог с программистом для уточнения свойств распараллеливаемой программы, для выполнения различных преобразований и.т.п. В процессе такого диалога программист может указать неверную информацию о программе, например, указать, что в цикле нет зависимости, хотя такая зависимость есть, выполнить некорректное преобразование программы. Опираясь на неверные указания программиста, система САПФОР может сгенерировать неэффективную и/или некорректную параллельную программу. В DVM-системе реализованы инструменты, которые автоматизируют процесс отладки параллельных программ [11–13].

5.1. Функциональная отладка программ

Ручные методы отладки, такие как пошаговое исследование процесса выполнения алгоритма в определенных точках останова или отладочная печать, часто не позволяют адекватно работать с реальными научными и инженерными программными комплексами, спроектированными на непрерывную работу в течение часов или даже дней, используя сотни параллельных потоков.

Для обнаружения ошибок в DVMH-программах используются автоматизированные методы отладки: динамический контроль и сравнительная отладка. Данные методы позволяют найти большинство ошибок в автоматическом режиме с минимальным участием программиста.

Для применения автоматизированных методов отладки компиляторы с языков C-DVMH и Fortran-DVMH выполняют инструментацию генерируемых программ. В программы добавляются специальные обращения (вызовы), которые передают отладчику информацию обо всех спецификациях параллелизма; позволяют контролировать все чтения/записи переменных, начало/конец циклов и других параллельных конструкций, операции синхронизации, редукционные операции и т.п.

Для отладки корректности параллельных программ в DVM-системе используется следующий подход. Сначала программа отлаживается с использованием привычных (штатных) средств отладки. Поскольку DVMH-директивы являются спецкомментариями языка Fortran или прагмами языка Си, то это позволяет отлаживать DVMH-программу как обычную последовательную программу (в режиме игнорирования DVMH-директив).

Затем программа запускается в режиме динамического контроля, который позволяет проверить корректность спецификаций параллелизма. Для этого моделируется параллельное выполнение DVMH-программы. Все ошибки, которые проявились в результате такого моделирования, выдаются пользователю:

– Необъявленная зависимость по данным в параллельном цикле.

– Использование в параллельном цикле или после выхода из него приватных переменных без их предварительной инициализации.

– Запись в переменные, доступные только на чтение.

– Использование редукционных переменных после запуска асинхронной редукции, но до ее завершения.

– Необъявленный доступ к нелокальным элементам распределенного массива.

– Запись в теневые грани массива.

– Чтение теневых элементов массива до завершения операции их обновления.

– Модификация нелокального элемента распределенного массива в последовательной части программы.

– Выход за пределы распределенного массива.

– Запись в буфер удаленного доступа.

Не все ошибки могут быть определены в результате динамического контроля. Например, с помощью динамического контроля не могут быть проанализированы процедуры и функции, для которых отсутствует исходный код. Для поиска ошибок в таких программах используется метод накопления и сравнения трасс, который позволяет определить место в программе и момент, когда

появляются расхождения в результатах вычислений.

Общая схема сравнительной отладки выглядит следующим образом:

1) Получение эталонной трассы. При трассировке выполняется сбор информации обо всех чтениях и модификациях переменных, о начале выполнения каждого витка цикла, о начале и конце выполнения параллельного цикла. В качестве эталонной трассы может выступать трасса последовательного выполнения программы (т.к. DVMH-программа одновременно может быть и последовательной, и параллельной); трасса параллельного выполнения программы, в т.ч. трасса, полученная на другом вычислительном кластере.

2) Автоматическое сравнение результатов выполнения программы с накопленной ранее эталонной трассой. Все целочисленные данные сравниваются на совпадение, а вещественные числа сравниваются с заданной точностью по абсолютной и относительной погрешности. В случае нахождения расхождений выдается информация о найденных различиях.

Сравнительная отладка показала свою эффективность на простых модельных задачах, но наткнулась на два препятствия: ресурсы и точность. Рассмотрим подробней суть этих проблем.

После полной инструментации каждый оператор окружается несколькими вызовами подпрограмм трассировщика, которые должны сформировать записи с читаемыми и записываемыми значениями переменных. Неудивительно, что при этом программа значительно замедляется. Например, в результате экспериментов с программами из пакета NAS NPB были получены следующие результаты. Замедление только от инструментации составляет 50–100 раз. Объемы трассы – несколько десятков байтов на каждый выполненный оператор присваивания – тоже оказались совершенно неприемлемыми для реальных программ (средний размер трасс >6.57e+12 байт, т.е. терабайты, и при среднем времени выполнения исходных программ ~15 сек – среднее время сбора трасы ~28 000 сек, т.е. увеличивается в 2000 раз!). Поэтому полная сравнительная отладка оказалась применимой только на “модельных” данных, которые не всегда доступны.

Другая проблема, с которой столкнулась система отладки, – “допустимое” несовпадение значений переменных. Такими переменными являются, например, редукционные переменные. При вычислении суммы элементов распределенного массива меняется порядок операций: сначала вычисляются локальные суммы, а потом они суммируются в непредсказуемом порядке. Результаты при этом получаются разными (типично расхождение в 1–2 младших разрядах, хотя в принципе могут различаться сколь угодно сильно), но, с точки зрения программиста, эти результаты могут быть одинаково допустимыми. Редукционные операции создают четыре проблемы для сравнительной отладки (“ложные тревоги”):

1) значения редукционной переменной на промежуточных итерациях не совпадают, потому что вычисляются только частичные суммы или максимум ищется в другом диапазоне и т.п.;

2) окончательное значение суммы, как сказано выше, может отличаться (недетерминизм в слабом смысле);

3) отличие в одной переменной может сразу распространиться на многие другие (например, если посчитана норма вектора и затем вектор нормируется);

4) а если это значение используется в критерии окончания итераций или при выборе ветви вычислений, то дальше могут быть выбраны разные ветви и трассы вообще могут оказаться несопоставимыми (недетерминизм в сильном смысле).

Для сокращения времени работы и объема трассы в DVM-системе были предложены и реализованы средства управления трассировкой:

– выборочная трассировка, например, только запись, только распределенные массивы и т.п.;

– локализация инструментации, т.е. выделение неинструментируемых фрагментов программы;

– предварительная оценка объема трассы, получение так называемого “файла циклов” и затем его ручная коррекция для отключения трассировки на определенных итерациях и т.д.;

– автоматический выбор итераций для трассировки: “границы” и “уголки”;

– генерация двух тел цикла: одно без вызовов трассировщика, другое инструментировано; при этом на выбранных для трассировки итерациях работает инструментированное тело цикла, на остальных – исходная программа;

– условный вызов подпрограмм трассировщика (т.е. перед вызовом подпрограммы трассировщика проверяется необходимость ее вызова);

– трассировка “контрольных сумм” массивов по завершении цикла вместо трассировки элементов массивов при выполнении тела цикла.

Проблема редукционных переменных была решена следующим образом:

– редукционные переменные распознаются (т.к. они описаны в DVMH-директивах), и либо обращения к ним в теле цикла не трассируются, либо уже в трассе соответствующие записи не сравниваются;

– трассировка и/или сравнение значений выполняется с некоторой заданной точностью. Ошибка фиксируется, только когда различие становится достаточно велико. Точность сравнения может задаваться программистом;

– в режиме сравнения реального выполнения одного варианта программы с трассой выполнения другого варианта реально вычисленное значение редукционной переменной заменяется ее значением из “эталонной” трассы.

Эти приемы позволили подавить “ложные тревоги”, связанные с редукционными переменными.

Следует отметить, что в настоящее время ведется разработка новой версии системы сравнительной отладки, которая будет основана на следующих принципах:

1) одновременное выполнение эталонной и отлаживаемой программы;

2) обмен информацией между эталонной и отлаживаемой программой в процессе выполнения;

3) коррекция отличающихся значений вещественных переменных в отлаживаемой программе на эталонные значения.

Реализация нового подхода (сравнение “на лету”, без использования трасс) сделает процесс отладки более гибким и гораздо менее требовательным к памяти вычислительного комплекса.

5.2. Анализ эффективности DVMH-программ

Для отладки эффективности параллельных программ в DVM-системе используется анализатор производительности, который позволяет получить информацию об основных характеристиках эффективности выполнения программы (или ее частей) на параллельной системе.

Эффективность выполнения параллельных программ на многопроцессорных ЭВМ с распределенной памятью определяется следующими основными факторами:

– эффективностью вычислений на каждом процессоре;

– степенью распараллеливания программы – долей параллельных вычислений в общем объеме вычислений;

– равномерностью загрузки процессоров во время выполнения параллельных вычислений;

– временем, необходимым для выполнения межпроцессорных обменов;

– степенью совмещения межпроцессорных обменов с вычислениями.

Существенным достоинством DVM-подхода по сравнению с подходами, базирующимися на явном использовании коммуникационных библиотек (MPI, SHMEM), является то, что в любой момент выполнения программы на любом процессоре всегда известно, какой участок программы выполняется – последовательный или параллельный. Кроме того, известны все точки программы, в которых выполняются операции, требующие синхронизации процессоров. Поэтому имеется возможность количественно оценить влияние на эффективность выполнения программы каждого из пяти перечисленных выше факторов. При явном использовании коммуникационных библиотек, когда параллельная программа представлена в виде системы взаимодействующих процессов, невозможно отличить последовательные вычисления от параллельных и определить степень распараллеливания программы.

Все функции системы поддержки выполнения DVMH-программ можно разделить на следующие непересекающиеся группы:

– функции передачи сообщений;

– функции запуска редукционных операций;

– функции ожидания завершения редукции;

– функции запуска обмена границами распределенных массивов;

– функции ожидания завершения обмена границами;

– функции распределения данных по процессорам;

– функции перераспределения данных между процессорами;

– функции организации выполнения параллельных циклов;

– функция опроса очередной порции витков параллельного цикла для ее выполнения;

– функции ввода/вывода;

– функции обеспечения доступа к удаленным данным;

– функции копирования данных на ускорители;

– функции запуска вычислительных регионов на ускорителях;

– прочие функции системы поддержки (не вошедшие ни в одну из перечисленных выше групп).

Во время работы DVMH-программы система поддержки накапливает характеристики выполнения функций каждой группы, а по окончании выполнения DVMH-программы записывает их в двоичном виде в файл, который затем может быть обработан и визуализирован анализатором производительности.

Пользователю выдаются следующие характеристики эффективности:

1. Коэффициент эффективности, который равен отношению полезного времени к общему времени использования процессоров.

2. Время выполнения – максимальное значение среди времен выполнения программы на всех используемых ею процессорах.

3. Число используемых процессоров.

4. Общее время использования процессоров – произведение времени выполнения на число используемых процессоров.

5. Полезное время – сумма трех составляющих: полезного процессорного времени, времени ввода-вывода и полезного системного времени.

6. Потерянное время – разница между общим временем использования процессоров и полезным временем. Недостаточный параллелизм, коммуникации и простои – составляющие потерянного времени.

7. Недостаточный параллелизм (например, дублирование вычислений) и его компоненты.

8. Коммуникации.

9. Простои процессора из-за его недостаточной загрузки.

10. Потенциальные потери из-за разбалансировки.

11. Потенциальные потери из-за синхронизации при выполнении коллективных операций.

12. Потенциальные потери из-за разброса времен выполнения коллективных операций и все их компоненты.

13. Время перекрытия. Эта характеристика отражает потенциальное сокращение коммуникационных расходов за счет совмещения межпроцессорных обменов с вычислениями.

14. И другие.

Данные характеристики эффективности вычисляются как для всей программы целиком, так и для заданных программистом фрагментов (интервалов). В компиляторах с языков C-DVMH и Fortran-DVMH реализованы специальные опции, которые позволяют оформить каждый последовательный, параллельный цикл в качестве интервала. Такой подход – представление программы в виде дерева интервалов – существенно упрощает процесс анализа эффективности DVMH-программ. Методика отладки производительности DVMH-программ описана в [8, 9].

6. СИСТЕМА САПФОР

Входными языками системы САПФОР являются Fortran 95 и Си 99, а результат распараллеливания представляет собой программы на языках Fortran-DVMH или C-DVMH. Основными компонентами САПФОР являются:

– анализаторы последовательных программ, которые определяют свойства программ, важные при их распараллеливании, выполняют статический и динамический анализ зависимостей по данным между витками циклов, определяют регулярные зависимости по данным, приватные и редукционные переменные, параметры задачи и.т.п.;

– блоки преобразования последовательных программ в параллельные программы (эксперты);

– диалоговая оболочка для взаимодействия с пользователем;

– генератор кода, создающий на основе принятых экспертом решений параллельную версию DVMH-программы.

Высокий уровень выходного языка (Fortran-DVMH или C-DVMH) позволяет продемонстрировать пользователю результат распараллеливания в понятных для него терминах. Как уже отмечалось, для данных языков существуют развитые средства отладки функциональности и эффективности, что позволяет ускорить развитие самой системы САПФОР, т.к. многие ошибки, которые возникают в процессе разработки, могут быть обнаружены во много раз быстрее.

Отличительной особенностью САПФОР является использование автоматически распараллеливающего компилятора, преобразующего потенциально параллельную программу в ее параллельную версию для заданной ЭВМ. При этом предварительный анализ программы и приведение программы к потенциально параллельному виду может выполняться в полуавтоматическом режиме. Для уточнения свойств последовательной программы, выявленных анализатором, используется либо диалоговая оболочка, либо специальные аннотации в тексте программы. В САПФОР наиболее сложный этап распараллеливания (получение параллельной версии программы) выполняется полностью автоматически.

Другой отличительной особенностью САПФОР является использование прогнозирования параллельного выполнения DVMH-программ для выбора наилучшего варианта распараллеливания. В составе DVM-системы реализован специальный инструмент – предсказатель производительности (предиктор), который предназначен для анализа и отладки производительности DVMH-программ без использования реальной параллельной машины. С помощью предиктора пользователь имеет возможность получить оценки временных характеристик выполнения его программы на кластере с различной степенью подробности. Предиктор представляет собой систему обработки информации в трассе, собранной системой поддержки выполнения DVMH-программ во время прогона программы на инструментальной ЭВМ и состоит из двух крупных компонент: интерпретатора трассы и оценщика времени. По трассе и заданным пользователем параметрам целевой ЭВМ интерпретатор трассы вычисляет временные характеристики выполнения данной программы на целевой ЭВМ. При этом он вызывает функции оценщика времени, который, в свою очередь, моделирует параллельное выполнение DVMH-программ и является, фактически, моделью библиотеки системы поддержки DVMH-программ, используемой ею библиотеки передачи сообщений (MPI) и аппаратуры.

На основе предиктора в системе САПФОР была реализована библиотека, которая используется при оценке различных схем распараллеливания и выборе оптимальной конфигурации системы. Использование данной библиотеки позволяет “увидеть” программисту все недостатки программы, полученной в результате автоматизированного распараллеливания, не прибегая к ее реальному выполнению. Данный прогноз может быть также востребован при выборе оптимизирующих преобразований, которые необходимо выполнить для повышения эффективности распараллеливаемой программы.

Использование сложных структур данных и указателей, наличие процедур или функций, исходный код которых недоступен или не может быть проанализирован (например, написан на другом языке программирования) серьезно усложняет процесс распараллеливания программ с помощью САПФОР. Очень часто процесс распараллеливания таких программ заканчивается неудачей.

Для решения данной проблемы может использоваться метод инкрементального распараллеливания. Идея этого метода заключается в том, что распараллеливанию подвергается не вся программа целиком, а ее части – в них заводятся дополнительные экземпляры требуемых данных, производится распределение этих данных и соответствующих вычислений.

Для взаимодействия с теми частями программы, которые не подвергались распараллеливанию, используются операции копирования исходных (нераспределенных) данных в дополнительные (распределенные) данные и обратно.

Для реализации данного подхода в системе САПФОР введены области распараллеливания. Областью распараллеливания называется последовательность исполняемых операторов в рамках одной области видимости (функции или процедуры).

По умолчанию вся программа рассматривается как одна область распараллеливания. Если САПФОР не справляется с распараллеливанием всей программы, то пользователь может определить свои области распараллеливания, используя специальные указания (директивы). Также возможно автоматическое определение областей, например, на основе профилирования программы в область распараллеливания могут попасть самые времяемкие фрагменты программы (например, процедуры, функции и циклы, которые занимают 90% времени выполнения программы).

Каждая область имеет свой уникальный идентификатор, называемый именем области. Совокупность разных областей (внутри функции, а также в разных файлах) с одинаковым именем будут рассматриваться системой САПФОР как одна область распараллеливания. Для каждой области распараллеливания с разными именами будут строиться разные и независимые решения по распределению данных и вычислений. Объявление всех необходимых дополнительных экземпляров данных, выполнение требуемых операций копирования данных САПФОР берет на себя.

Использование областей распараллеливания имеет следующие преимущества:

1) Возможность распараллелить не всю программу, а ее времяемкие фрагменты, упрощает работу системы и/или программиста и позволяет потратить больше времени ЭВМ (и программиста), чтобы найти лучшие схемы распараллеливания времяемких фрагментов;

2) Отказ от распараллеливания сложных фрагментов позволяет с большей вероятностью найти хорошие решения, поскольку эти сложные фрагменты могут приводить к неверной работе алгоритмов анализа, распределения данных и вычислений, или к тому, что не найдется схем распараллеливания, обеспечивающих приемлемое ускорение;

3) Возможна реализация ручного распараллеливания некоторых фрагментов программы и учета принятых программистом решений при распараллеливании других фрагментов.

Реализация режима инкрементального распараллеливания существенно расширила область применимости системы САПФОР [14].

7. АПРОБАЦИЯ ПОДХОДА

С использованием DVM-подхода было разработано множество параллельных программ, среди которых:

1) Elasticity3D (численное моделирование 3D сейсмических полей в упругих средах для областей со сложной геометрией свободной поверхности) [15];

2) MHPDV (моделирование сферического взрыва во внешнем магнитном поле с помощью решения уравнений идеальной магнитной гидродинамики) [16];

3) “Композит” (моделирование многокомпонентной многофазной изотермической фильтрации при разработке месторождений нефти и газа) [17];

4) HyperbolicSolver2D (решение по явной схеме систем гиперболических уравнений в двумерных областях сложной формы с использованием неструктурированных сеток) [17];

5) “Теплопроводность” (решение краевой задачи для двумерного квазилинейного параболического уравнения, записанного в дивергентной форме в различной постановке на неструктурированных треугольных сетках) [17];

6) “Каверна” (моделирование циркуляционного течения в плоской квадратной каверне с движущейся верхней крышкой) [18];

7) “Контейнер” (моделирование течения вязкой тяжелой жидкости под действием силы тяжести в прямоугольном контейнере с открытой верхней стенкой и отверстием в одной из боковых стенок) [18];

8) GIMM_APP_Powder_3D/GIMM_APP_Powder_3D (параллельные программы двухмерного/трехмерного моделирования процессов плавления многокомпонентных порошков при селективном лазерном спекании на основе многокомпонентной и многофазной гидродинамической модели) [19];

9) QuantumBitStates (расчет состояний кубитов квантового компьютера на основе решения нестационарного уравнения Шредингера для двух частиц с учетом спина) [20];

10) GIMM_APP_Crystal_2D/GIMM_APP_Crystal_3D (параллельные программы двухмерного/трехмерного моделирования процессов объемной кристаллизации при воздействии на образец лазерного или электронного пучка на основе многокомпонентной и многофазной гидродинамической модели) [20];

11) пакет прикладных программ “РЕАКТОР” (нейтронно-физический расчет ядерных реакторов и гибридных ядерных установок) [21];

12) программы из пакета NAS Parallel Benchmarks [22, 23];

и многие другие.

Разработанные DVMH-программы без каких-либо изменений могут эффективно выполняться на кластерах различной архитектуры, использующих многоядерные универсальные процессоры, графические ускорители и сопроцессоры Intel Xeon Phi. На суперкомпьютерных системах “K-100” (ИПМ им. М.В. Келдыша РАН), “Ломоносов” (НИВЦ МГУ), “МВС-10П” (МСЦ РАН) были получены результаты расчетов данных программ при использовании до 30000 процессорных ядер и до 1280 графических ускорителей, что говорит о применимости данного подхода для разработки параллельных программ.

Эффективность разработанных в высокоуровневой модели DVMH программ близка к эффективности программ, написанных с использованием низкоуровневых моделей MPI, CUDA или OpenCL и превосходит эффективность программ в модели OpenACC [15, 18, 22–25].

8. ЗАКЛЮЧЕНИЕ

Множество параллельных программ создаются с использованием того огромного программного задела, который был получен на последовательных ЭВМ. Необходимость разработки методики распараллеливания существующих последовательных программ ощущается очень остро. В ИПМ им. М.В. Келдыша РАН ведутся работы по обобщению опыта распараллеливания программ и созданию соответствующей методики. Такая методика поддержана специальными инструментами, автоматизирующими анализ последовательных программ, извлечение их свойств, существенных для распараллеливания этих программ, и выполняющими их эффективное отображение на гетерогенные вычислительные системы.

Описанный в статье DVM-подход для разработки прикладного программного обеспечения для суперкомпьютерных систем с ускорителями основан на использовании возможностей систем САПФОР и DVM.

По сравнению с существующими системами автоматизации распараллеливания система САПФОР имеет следующие преимущества:

1) Наличие автоматически распараллеливающего компилятора для кластера с ускорителями;

2) Поддержка режима инкрементального распараллеливания программ для кластера;

3) Использование прогнозирования параллельного выполнения программы для выбора наилучшего варианта распараллеливания;

4) Использование языка высокого уровня в качестве выходного языка системы.

Использование DVM-системы в процессе разработки параллельной программы дает следующие преимущества:

1) Все больше моделей параллельного программирования реализуются посредством добавления в программы на стандартных последовательных языках спецификаций, управляющих отображением этих программ на параллельные машины. Примерами таких моделей являются: OpenMP, OpenACC, HMPP, APM PGI и др. Но данные модели позволяют описать параллелизм внутри одного узла кластера. Для использования нескольких узлов кластера требуется использовать комбинацию из нескольких моделей, например, MPI+OpenMP или SHMEM+OpenACC, что существенно усложняет процесс разработки параллельной программы, т.к. модели MPI, SHMEM являются низкоуровневыми. Модель DVMH лишена данного недостатка. Высокоуровневые DVMH-спецификации позволяют описать все уровни параллелизма, существующие в аппаратуре на сегодняшний день;

2) Для анализа эффективности параллельного выполнения программ разработано множество различных инструментов, например, NVIDIAVisual Profiler, Intel Vtune Amplifier, Intel Trace Analyzer and Collector, MPI Performance Snapshot и др. Основная проблема заключается в том, что на сегодняшний день нет единого удобного инструмента для анализа эффективности параллельного

выполнения программ для кластеров с ускорителями. Такой анализ может потребовать нескольких запусков программы с использованием различных инструментов. Например, MPI Performance Snapshot для анализа потерь, связанных с межпроцессорными обменами, и NVIDIA Visual Profiler для обнаружения узких мест при выполнении программы на графическом ускорителе. Анализатор производительности, реализованный в DVM-системе, лишен данного недостатка. За один запуск анализатор собирает информацию об эффективности выполнения программы на узлах кластера, на ускорителях, а также на ядрах центрального процессора;

3) Для отладки параллельного выполнения программ разработано множество различных инструментов, например, NVIDIA Nsight Graphics, Intel Inspector, Intel Message Checker, Valgrind и др. Для отладки программы, использующей несколько моделей параллельного программирования, требуется использовать различные инструменты. Например, MPI+OpenMP программа может быть сначала отлажена, как MPI-программа, затем как OpenMP, но при этом нет никаких гарантий, что при работе программы в режиме MPI+OpenMP не проявятся новые ошибки. В DVM-системе реализованы единые средства для автоматизированной отладки DVMH-программ, которые выдают ошибки, проявившиеся при параллельном выполнении программы, в терминах программы пользователя.

Таким образом, использование DVM-подхода существенно упрощает процесс разработки параллельных программ для гибридных вычислительных кластеров. Получаемые DVMH-программы без каких-либо изменений могут эффективно выполняться на кластерах различной архитектуры, использующих многоядерные универсальные процессоры, графические ускорители и сопроцессоры Intel Xeon Phi. Это достигается как за счет различных оптимизаций, которые выполняются статически при компиляции DVMH-программ, так и за счет динамических оптимизаций. Параллельные программы могут настраиваться при запуске на выделенные для их выполнения ресурсы (количество узлов кластера, ядер, ускорителей и их производительность). Такое свойство программ позволяет запускать их на произвольной конфигурации (число процессоров, нитей, ускорителей), компоновать сложные программы из имеющихся простых программ, а также повысить эффективность использования параллельных систем коллективного пользования за счет более гибкого распределения ресурсов между отдельными программами.

СПИСОК ЛИТЕРАТУРЫ

1. *Эйсымонт Л.К.* Гибридная стратегия развития элементной базы // Открытые системы. СУБД. 2017. № 2. С. 8–11. http://www.osp.ru/os/2007/09/4569294
2. Top500 List – November 2018. http://top500.org/list/2018/11/
3. OpenMP 4.0 Specifications. http://openmp.org/wp/openmp-specifications/
4. OpenACC. Specification. https://www.openacc.org/specification
5. *Бахтин В.А.* Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами / В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов // Вестник Южно-Уральского государственного университета, серия: “Математическое моделирование и программирование”. 2012. № 18 (277). С. 82–92.
6. *Бахтин В.А.* Автоматическое отображение программ на языке Фортран на кластеры с графическими процессорами / В.А. Бахтин, М.С. Клинов, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2014. Т. 3. № 3. С. 86–96.
7. DVM-система. Система разработки параллельных программ. http://dvm-system.org/
8. Язык Fortran-DVMH, Fortran-DVMH компилятор, компиляция, выполнение и отладка DVMH-программ. http://dvm-system.org/static_data/tutorial/FDVMH-tutorial.pdf
9. Язык C-DVMH, C-DVMH компилятор, компиляция, выполнение и отладка DVMH-программ. http://dvm-system.org/static_data/docs/CDVMH-reference-ru.pdf
10. *Бахтин В.А.* Методы динамической настройки DVMH-программ на кластеры с ускорителями / В.А. Бахтин, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула // Суперкомпьютерные дни в России: Труды международной конференции (28–29 сентября 2015 г., г. Москва), М.: Изд-во МГУ. 2015. С. 257–268.
11. *Крюков В.А.* Отладка DVM-программ / В.А. Крюков, Р.В. Удовиченко // Программирование. 2001. № 3. С. 19–29.
12. *Крюков В.А.* Автоматизация отладки параллельных программ / В.А. Крюков, М.В. Кудрявцев // Вычислительные методы и программирование. 2006. Т. 7. № 4. С. 102–110.
13. *Ермичев А.А.* Развитие метода сравнительной отладки DVMH-программ / А.А. Ермичев, В.А. Крюков // Научный сервис в сети Интернет: труды XIX Всероссийской научной конференции (18–23 сентября 2017 г., г. Новороссийск), М.: ИПМ им. М.В. Келдыша. 2017. С. 150–156.
14. *Бахтин В.А.* Распараллеливание программных комплексов. Проблемы и перспективы / В.А. Бахтин, О.Ф. Жукова, Н.А. Катаев, А.С. Колганов, В.А. Крюков, М.Ю. Кузнецов, Н.В. Поддерюгина, М.Н. Притула, О.А. Савицкая, А.А. Смирнов //

Научный сервис в сети Интернет: труды XX Всероссийской научной конференции (17–22 сентября 2018 г., г. Новороссийск)/ М.: ИПМ им. М.В. Келдыша. 2018. С. 63–72.

15. *Катаев Н.А.* Автоматизированное распараллеливание задачи моделирования распространения упругих волн в средах со сложной 3D геометрией поверхности на кластеры разной архитектуры / Н.А. Катаев, А.С. Колганов, П.А. Титов // Параллельные вычислительные технологии – XI международная конференция, ПаВТ'2017, г. Казань, 3–7 апреля 2017 г. Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ. 2017. С. 341–355.
16. *Бахтин В.А.* Автоматическое распараллеливание последовательных программ для многоядерных кластеров / В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина // В сборнике: Научный сервис в сети Интернет: суперкомпьютерные центры и задачи Труды международной суперкомпьютерной конференции. Российская академия наук Суперкомпьютерный консорциум университетов России. 2010. С. 12–15.
17. *Алексахин В.Ф.* Опыт решения прикладных задач с использованием DVM-системы / В.Ф. Алексахин, В.А. Бахтин, Д.А. Захаров, А.С. Колганов, А.В. Королев, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула // Труды международной конференции Суперкомпьютерные дни в России (25–26 сентября 2017 г., г. Москва). М.: Изд-во МГУ. 2017. С. 650–661.
18. *Бахтин В.А.* Расширение DVM-модели паралельного программирования для кластеров с гетерогенными узлами / В.А. Бахтин, В.А Крюков, Б.Н. Четверушкин, Е.В. Шильников // Доклады Академии наук. 2011. Т. 441. № 6. С. 734–736.
19. *Бахтин В.А.* Автоматическое отображение Фортран-программ на кластеры с ускорителями / В.А. Бахтин, М.С. Клинов, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула // В сборнике: Научный сервис в сети Интернет: многообразие суперкомпьютерных миров Труды международной суперкомпьютерной конференции. Российская академия наук Суперкомпьютерный консорциум университетов России. 2014. С. 17–22.
20. *Притула М.Н.* Отображение DVMH-программ на кластеры с графическими процессорами [Текст]: дис. канд. физ.-мат. наук: 05.13.11: защищена 17.12.13 / Автор Михаил Николаевич Притула. М., 2013. 105 с. http://keldysh.ru/council/1/pritula-diss.pdf
21. *Воронков А.В.* Параллельная версия пакета РЕАКТОР-ГП / А.В. Воронков, А.С. Голубев, В.А. Крюков, Е.П. Сычугова // Вопросы атомной науки и техники. Серия: Обеспечение безопасности АЭС, М.: Изд. ОАО ОКБ “ГИДРОПРЕСС”. 2009. № 24. С. 64–74.
22. *Алексахин В.Ф.* Распараллеливание на графические процессоры тестов NAS NPB3.3.1 на языке Fortran DVMH / В.Ф. Алексахин, В.А. Бахтин, О.Ф. Жукова, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, О.А. Савицкая, А.В. Шуберт // Вестник Уфимского государственного авиационного технического университета. 2015. Т. 19. № 1 (67). С. 240–250.
23. *Алексахин В.Ф.* Распараллеливание на языке Fortran-DVMH для сопроцессора Intel Xeon Phi тестов NAS NPB3.3.1 / В.Ф. Алексахин, В.А. Бахтин, О.Ф. Жукова, А.С. Колганов, В.А. Крюков, И.П. Островская, Н.В. Поддерюгина, М.Н. Притула, О.А. Савицкая // Сборник трудов международной научной конференции “Параллельные вычислительные технологии 2015”, Челябинск: Издательский центр ЮУрГУ. 2015. С. 19–30.
24. *Крюков В.А.* Разработка параллельных программ для вычислительных кластеров и сетей / В.А. Крюков // Информационные технологии и вычислительные системы. 2003. № 1–2. С. 42–61. ftp://ftp.keldysh.ru/dvm-distr/journ1-2_page42_61.pdf
25. Эффективность выполнения тестов NAS NPB. http://dvmsystem.org/ru/category/performance/