АВТОМАТИЗАЦИЯ ВЫЯВЛЕНИЯ ПРИЧИН ПОТЕРИ ПРОИЗВОДИТЕЛЬНОСТИ MPI ПРОГРАММ НА ЭКЗАФЛОПСНЫХ И ДРУГИХ БОЛЬШИХ СУПЕРКОМПЬЮТЕРАХ

А.В. Дергунов

По прогнозам мировых экспертов, в ближайшем будущем производительность суперкомпьютерных систем достигнет и превзойдет экзафлопсный уровень. Одной из проблем при создании подобных систем является анализ производительности программ, которые выполняются на таких суперкомпьютерах. Наиболее часто для анализа производительности программ используют средства, которые осуществляют сбор трассы и ее визуализацию. Но анализ такой трассы вручную на предмет выявления причин недостаточной производительности затруднен, так как необходимо анализировать большой объем данных, характеризующийся сложными зависимостями. Эта проблема особенно остро стоит при увеличении количества узлов при анализе трасс с больших суперкомпьютеров. Поэтому возникает потребность в средствах, которые бы автоматизировали анализ трассы и подсказали пользователю, как повысить производительность его программы. В данной работе описывается программная система, выполняющая эту задачу.

Введение

Для анализа MPI программ наиболее часто используют программные системы, которые осуществляют сбор и визуализацию трассы выполнения программы. Примером такого средства является Jumpshot [1]. На рис. 1 показано окно временной диаграммы системы Jumpshot, показывающее трассу работы MPI программы, реализующей сеточные вычисления при работе на 32 процессорах в течение 9 секунд (см. описание этой программы далее в разделе «Эксперимент по повышению производительности MPI программы»).

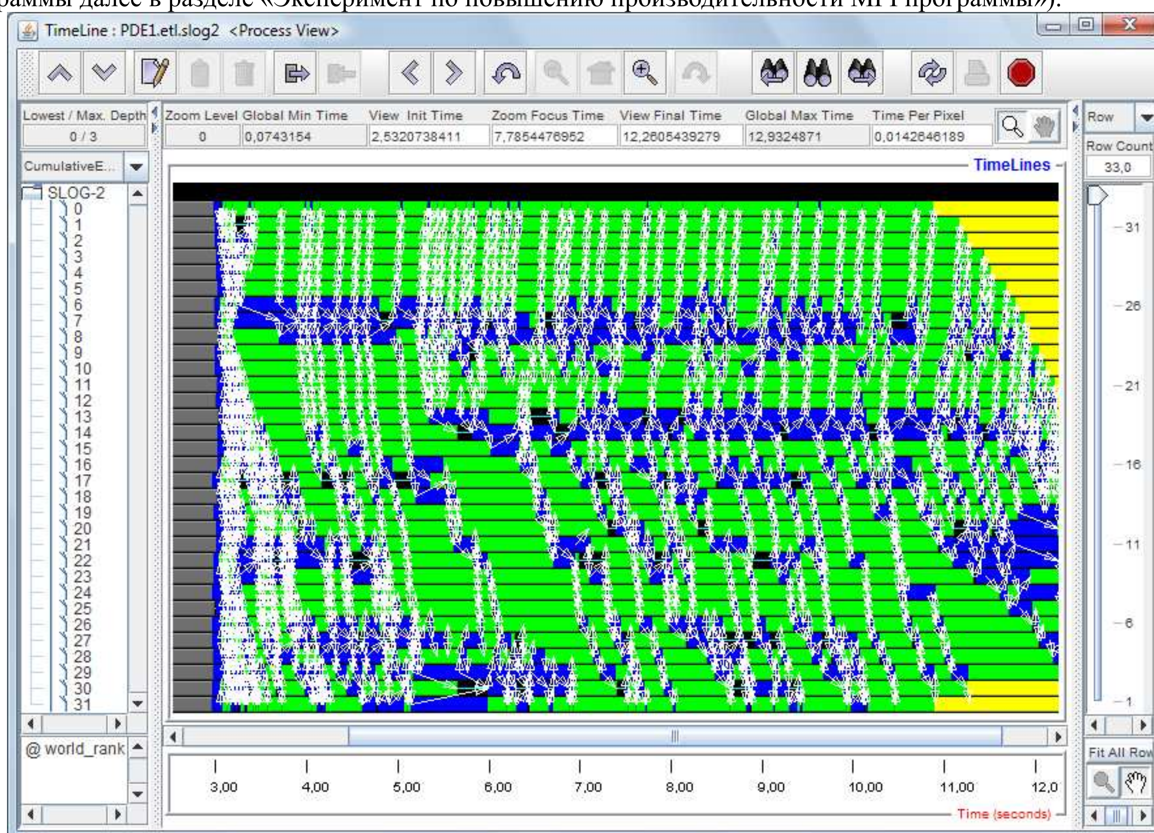

Рис. 1. Окно временной диаграммы системы Jumpshot для трассы MPI программы, реализующей сеточные вычисления при работе на 32 процессорах в течение 9 секунд.

Но при использовании таких инструментов пользователь сталкивается с проблемой анализа больших объемов информации. Пользователь должен использовать инструменты зуммирования и фильтрации, чтобы исследовать события трассы. Эта проблема особенно актуальная при анализе трассы с большим количеством процессов, собранных на больших суперкомпьютерах.

Другой проблемой при использовании средств визуализации трассы является то, что часто встречающиеся ситуации, приводящие к потерям производительности MPI программ, явно не визуализируются.

Таким образом, пользователь должен опираться на свой опыт работы с MPI и детально исследовать трассу программы для выявления причин недостаточной производительности.

Одной из причин недостаточной производительности является плохая синхронизация приемов и передач данных в MPI программе. В результате процедура приема данных может простаивать, дожидаясь посылки данных (см. рис. 2). Для решения этой проблемы нужно обеспечить, чтобы сообщения посылались как можно раньше, и т.о. повысить вероятность того, что сообщение прибудет до момента, когда оно потребуется другому процессу.

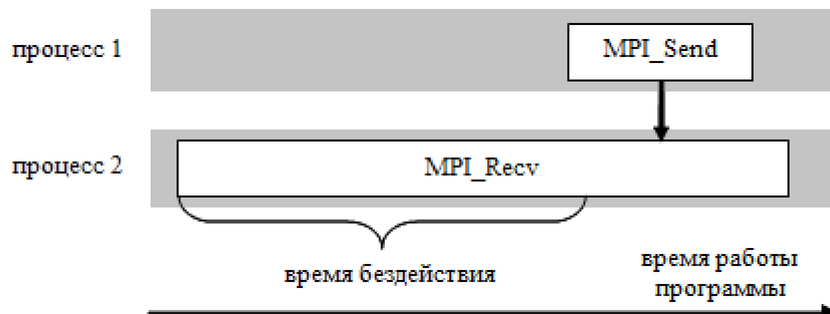

Рис. 2. Поздняя посылка данных.

Но даже если пользователь обнаружил причину падения производительности, то оказывается невозможным оценить, насколько она влияет на общее время работы программы. Например, чтобы оценить, в какой степени поздняя посылка данных замедлила работу программу, не достаточно просто обратиться к суммарному времени, затраченному на вызов процедур MPI_Recv, т. к. простой при ожидании данных составляет лишь долю времени работы этой процедуры. Поэтому нет возможности оценить, какой выигрыш производительности получит пользователь, если соответствующим образом изменит свою программу.

В этой работе рассматривается созданная автором система, которая позволяет автоматизировать анализ трасс MPI программ. В рамках системы разработан декларативный язык анализа событий трассы. С помощью этого языка описаны правила выявления причин недостаточной производительности MPI программ.

Общая схема работы системы

Схема работы системы представлена на рис. 3. Исходными данными является трасса выполнения MPI приложения, полученная с помощью трассировщика. Она анализируется модулем автоматического анализа трассы. Для этого используется описание правил анализа трассы на специальном языке. Результат анализа – список выявленных причин недостаточной производительности пользовательского приложения с указанием степени их влияния на общее время работы приложения.

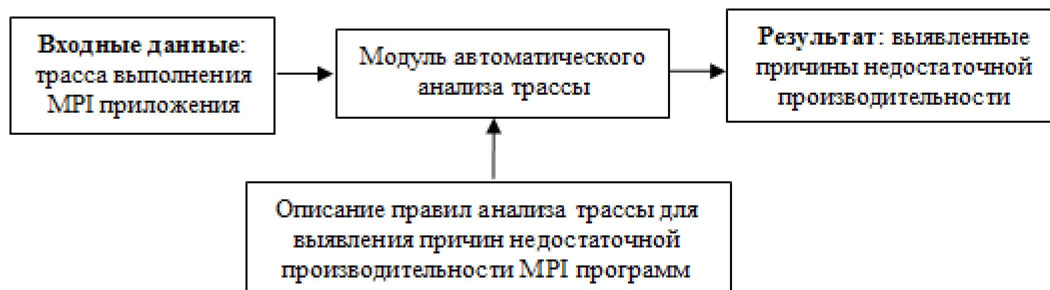

Рис. 3. Схема работы системы.

Декларативный язык анализа событий трассы

Для автоматизации анализа трассы был разработан специальный декларативный язык, который позволяет описывать события трассы, а также правила для их анализа.

Система использует как простые события (например, вызов определенной функции), записанные в трассу во время работы программы, так и составные события, которые представляют собой высокоуровневые операции (например, операция двухточечного обмена сообщениями) и формируются во время анализа трассы на основе данных простых событий. Событие характеризуется типом и набором параметров. В таблице 1 в качестве примера приведены параметры простого события function_call_non_blocking_receive_operation (которое соответствует вызову функции неблокирующей операции приема MPI — MPI_Irecv или MPI_Recv_init):

Таблица 1. Параметры события function_call_non_blocking_receive_operation

Название параметра	Тип значений	Описание

@start_time	целочисленный	время начала выполнения функции
@finish_time	целочисленный	время окончания выполнения функции
@function_name	стоковый	название функции
@call_stack	стоковый	стек вызовов
dest	целочисленный	ранг принимающего процесса
source	целочисленный	ранг передающего процесса
tag	целочисленный	идентификатор (тэг) сообщения
comm	целочисленный	коммуникатор
request	целочисленный	идентификатор запроса на передачу сообщения

Для анализа трассы используются правила, синтаксис описания которых следующий:

```
rule
    event <имя 1-й переменной> type <тип 1-го события>
    ...
    event <имя N-й переменной> type <тип N-го события>
    where
        <условное выражение>
    <действия>;
```

Для каждого правила перечисляется, какие события и каких типов надо найти в трассе. Каждому событию присваивается имя переменной. Для срабатывания правила необходимо выполнение условного выражения. Для правила также указываются действия при его выполнении. Примерами действий являются: операции с событиями (создание нового события, удаление или изменение существующего события), а также вывод причины недостаточной производительности программы. Правила используются в системе для следующих целей:

- описание составных событий, которые формируются на основе простых событий, собранных во время работы программы, или других составных событий и представляют собой высокоуровневые операции программы;
- описание причин недостаточной производительности MPI процессов.

Ниже приведен пример правила, описывающего событие операции приема данных, как состоящее из простых событий, соответствующих вызовам функций MPI_Recv_init и MPI_Start.

```
rule
    event e1 type function_call_non_blocking_receive_operation
    event e2 type function_call_request_handling
    where
        e1.@function_name eq "MPI_Recv_init" and
        e2.@function_name eq "MPI_Start" and
        e1.request = e2.request
    compose event of type point_to_point_operation
        type = RECEIVE_OPERATION
        function_name = e1.funcion_name
        start_time = e2.start_time
        finish_time = e2.finish_time
        source = e1.source
        dest = e1.dest
        tag = e1.tag
        comm = e1.comm
    delete e1
    delete e2;
```

Следующее правило описывает ситуацию поздней посылки данных в MPI программе:

```
rule
  event e type point_to_point_operation_group
  where
```

```
        e.receive_function_name eq "MPI_Recv" and
        e.send_start_time > e.receive_start_time
      create performance observation
        name = "Поздняя посылка данных"
          description = "Операция посылки данных вызывается позднее операции
приема.
          Из-за этого блокирующая операция приема вынуждена простаивать."
          advice = "Улучшить синхронизацию приемов и передач данных, оправляя
данные
              по возможности раньше, или использовать неблокирующие операции
приема."
        impact = op1.send_start_time - op1.receive_start_time;
```

С помощью приведенного правила осуществляется обнаружение операций двухточечного обмена данными, операция приема данных в которых блокирующего типа (MPI_Recv) и время начала посылки данных позднее времени начала приема данных. При выполнении перечисленных условий система делает вывод о неэффективном выполнении программы, связанным с поздней посылкой данных. В правиле указывается подробное описание этой причины недостаточной производительности и совет по ее устранению. Также вычисляется степень влияния на общее время работы программы, как разница между временем начала посылки данных и временем его приема.

Автоматизация повышения производительности MPI программ

С помощью описанного языка в системе реализовано выявление следующих причин недостаточной производительности MPI программ:

- поздняя посылка данных;
- поздний прием данных;
- прием сообщений в неверном порядке;
- ранний прием данных при операции «от многих к одному»;
- поздняя посылка данных при операции «от одного ко многим»;
- задержка перед операцией «от многих ко многим»;
- задержка перед барьерной синхронизацией;
- зависимость по данным при двухточечном обмене;
- задержка при создании или освобождении «окна» для удаленного доступа к памяти;
- ожидание при исключающей синхронизации удаленного доступа к памяти;
- позднее начало эпохи предоставления доступа к «окну»;
- раннее завершение эпохи предоставления доступа к «окну».

Описание некоторых из перечисленных причин недостаточной производительности приведено в [2]. Этот список был составлен на основе анализа литературы по оптимизации MPI программ ([3; 4; 5]).

В рамках этой работы был также реализован трассировщик для сбора трассы работы программы. События для трассировки задаются с помощью конфигурационного файла. В системе используется конфигурационный файл, который указывает трассировщику записывать в трассу события при вызове пользовательской программой MPI функций. Трассировщик реализован с помощью динамического инструментирования программы и, таким образом, может использоваться совместно с любой MPI библиотекой, и не требует перекомпиляции программы. Он реализован с помощью системы PIN [6].

Эксперимент по повышению производительности MPI программы

Для проверки разработанной системы было осуществлено повышение производительности MPI программы, реализующей сеточные вычисления, а именно метод итераций Якоби для численного решения задачи Дирихле для уравнения Лапласа [7]. Уравнение Лапласа представлено ниже:

$$\frac{\partial^2 \Phi}{\partial x^2}+\frac{\partial^2 \Phi}{\partial y^2}=0$$

В задаче заданы значения на граничных точках двумерной сетки. Необходимо вычислить значения во внутренних точках. Метод итераций Якоби [7] предусматривает несколько итераций, в которых новые значения в точках вычисляются как среднее из значений четырех ее соседних точек, вычисленных на предыдущей итерации:

```
for (int iter = 0; iter < iterations; iter++)
{
  for (int i = 0; i < height; i++)
```

```
    for (int j = 0; j < width; j++)
      new[i][j] = (grid[i-1][j] + grid[i][j-1] +
                   grid[i+1][j] + grid[i][j+1]) / 4;
  copy(grid, new);
}
```

Для реализации метода итераций Якоби на компьютере с распределенной памятью удобно избавиться от операции копирования матрицы new в grid, продублировав предыдущий цикл и поменяв в нем new и grid местами, а затем каждому процессу необходимо назначить свою прямоугольную полосу. Для вычисления значений на краях нужно использовать данные, размещенные на других процессорах. Поэтому необходимо использовать соответствующие операции обмена MPI. Таким образом, каждый процесс выполняет следующую последовательность действий:

1. вычислить значения во внутренних точках своей полосы;
2. отправить соседям вычисленные значения на краях своей полосы;
3. получить от соседей значения на краях их полос.

В рамках эксперимента MPI программа, реализующая данный алгоритм, была запущена на кластере с 32 процессами и собрана трасса ее выполнения. На рис. 1 показано окно временной диаграммы, визуализирующее эту трассу. В результате анализа трассы в автоматическом режиме разработанная система выявила позднюю посылку данных.

Учитывая проведенный анализ трассы, разработанная система в качестве одного из возможных способов повышения производительности предложила улучшить синхронизацию приемов и передач данных, отправляя данные по возможности раньше. В этом случае между отправкой и получением сообщений будут выполняться локальные вычисления. Принимая во внимание предложенный совет, исходная программа была изменена так, чтобы каждый процесс выполнял следующую последовательность действий:

1. отправить соседям значения на краях своей полосы;
2. вычислить значения во внутренних точках своей полосы;
3. получить от соседей значения на краях их полос;
4. вычислить значения на краях своей полосы.

Таблица 2 содержит сравнение времени работы двух версий программы. Обе программы были запущены на кластере с 32 процессами с одними и теми же параметрами (одинаковым количеством итераций цикла и размером матрицы). В результате оптимизации программы время работы приложения сократилось в 1,78 раза.

Таблица 2. Результат повышения производительности MPI программы

Версия программы	Время работы
До оптимизации	9,12 сек.
После оптимизации	5,11 сек.

Заключение

В данной статье была рассмотрена программная система, которая значительно облегчает задачу анализа производительности MPI приложений. Предложен декларативный язык анализа событий трассы, использование которого позволяет автоматизировать анализ трассы MPI программы и выявлять часто встречающиеся причины недостаточной производительности. Продемонстрировано использование разработанной системы для повышения производительности MPI программы, в результате которого время ее работы сократилось в 1,78 раза. Система также может использоваться для выполнения других задач анализа трассы (например, выявление ошибок в MPI программах, таких как взаимная блокировка или несогласованные прием-передача сообщений), а также для анализа любых других трасс (например, для приложений, использующих другие библиотеки). Система особенно актуальна для больших суперкомпьютеров, использующих большое количество процессоров, т. к. в этом случае задача анализа трассы особенно трудна.

ЛИТЕРАТУРА:

1. Omer Zaki, Ewing Lusk, William Gropp, Deborah Swider. Toward Scalable Performance Visualization with Jumpshot // High-Performance Computing Applications, volume 13, number 2, pages 277-288, 1999.
2. Карпенко С.Н., Дергунов А.В. Модели и программные средства повышения производительности MPI-приложений // Технологии Microsoft в теории и практике программирования: Материалы конференции. Н.Новгород: Изд. Нижегородского госуниверситета, 2009. С. 188-192.
3. William G., Ewing L. Tuning MPI programs for peak performance [Электронный ресурс] // Argonne National Laboratory : сайт. – URL: http://www.mcs.anl.gov/research/projects/mpi/tutorials/perf/ (дата

обращения 26.10.2010)
4. Barney B.. MPI Performance Topics [Электронный ресурс] // High Performance Computing. Lawrence Livermore National Laboratory : сайт. – URL: https://computing.llnl.gov/tutorials/mpi_performance/ (дата обращения 26.10.2010)
5. Rabenseifner R.. Optimization of MPI Applications [Электронный ресурс] // High Performance Computing Center Stuttgart : сайт. – URL: https://fs.hlrs.de/projects/par/par_prog_ws/engl/mpi_optimize_3/index.html (дата обращения 26.10.2010)
6. Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, Kim Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation // Programming Language Design and Implementation (PLDI), Chicago, IL, June 2005, pp. 190-200.
7. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования. М.: Издательский дом «Вильямс», 2003. 512 с.