ОБЗОР И СРАВНЕНИЕ ПОДХОДОВ К АВТОМАТИЗАЦИИ АНАЛИЗА ПРОИЗВОДИТЕЛЬНОСТИ MPI-ПРИЛОЖЕНИЙ

А.В. Дергунов

Нижегородский госуниверситет им. Н.И. Лобачевского

В настоящее время разработано несколько инструментальных средств для анализа производительности MPI-приложений. В статье рассмотрены подходы этих систем для выявления причин недостаточной производительности MPI-программ. Рассматривается повышение производительности одной MPI-программы, реализующей сеточные вычисления, с использованием этих систем.

Эксперимент по анализу производительности MPI-приложения

Для сравнения рассматриваемых систем был проведен анализ производительности MPI программы, реализующей сеточные вычисления, а именно метод итераций Якоби для численного решения задачи Дирихле для уравнения Лапласа:

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

В задаче даны значения на граничных точках двумерной сетки. Необходимо вычислить значения во внутренних точках. Для реализации метода итераций Якоби на компьютере с распределенной памятью каждому процессу назначается своя прямоугольная полоса. Каждый процесс выполняет следующую последовательность действий:

- вычислить значения во внутренних точках своей полосы;
- отправить соседям вычисленные значения на краях своей полосы (с помощью MPI_Send);
- получить от соседей значения на краях их полос (с помощью MPI_Recv).

В рамках эксперимента MPI программа, реализующая данный алгоритм, была запущена на кластере и собрана трасса ее выполнения. Рассмотрим подходы, предлагаемые существующими системами для анализа этой трассы.

Сбор и визуализация трассы выполнения приложения

Пожалуй, наиболее известными и часто используемыми являются системы для сбора и визуализации трассы выполнения MPI-приложения. Примером такой системы является Intel Trace Collector and Analyzer [1]. Для анализа производительности доступен достаточно широкий набор визуальных средств, в частности пространственно-временная диаграмма и статистика по функциям. На рис. 1 показаны эти средства для трассы рассматриваемой MPI программы. При использовании перечисленных графических инструментов возникают следующие трудности.

- Анализ большого объема данных. Даже при достаточно коротком времени работы программы и небольшом числе процессов на временной диаграмме зачастую отображается очень большое количество информации (на рис. 1 показана только малая часть трассы). Статистика по функциям представляет данные в более компактной форме, но их не всегда достаточно для анализа.

- Часто встречающиеся проблемы производительности (например, «поздняя посылка данных» [2]) явно не обозначаются. Таким образом, для выяснения причин недостаточной производительности требуется анализ данных вручную с помощью инструментов зуммирования, фильтрации и т.д.
- Отсутствуют средства для выдачи рекомендаций пользователю по улучшению производительности. Таким образом, оптимизация производительности зачастую требует от пользователя глубоких знаний о MPI (возможностях по оптимизации, предусмотренных в этой библиотеке, принципах работы реализаций библиотеки MPI), особенностях сети передачи данных и других знаний.

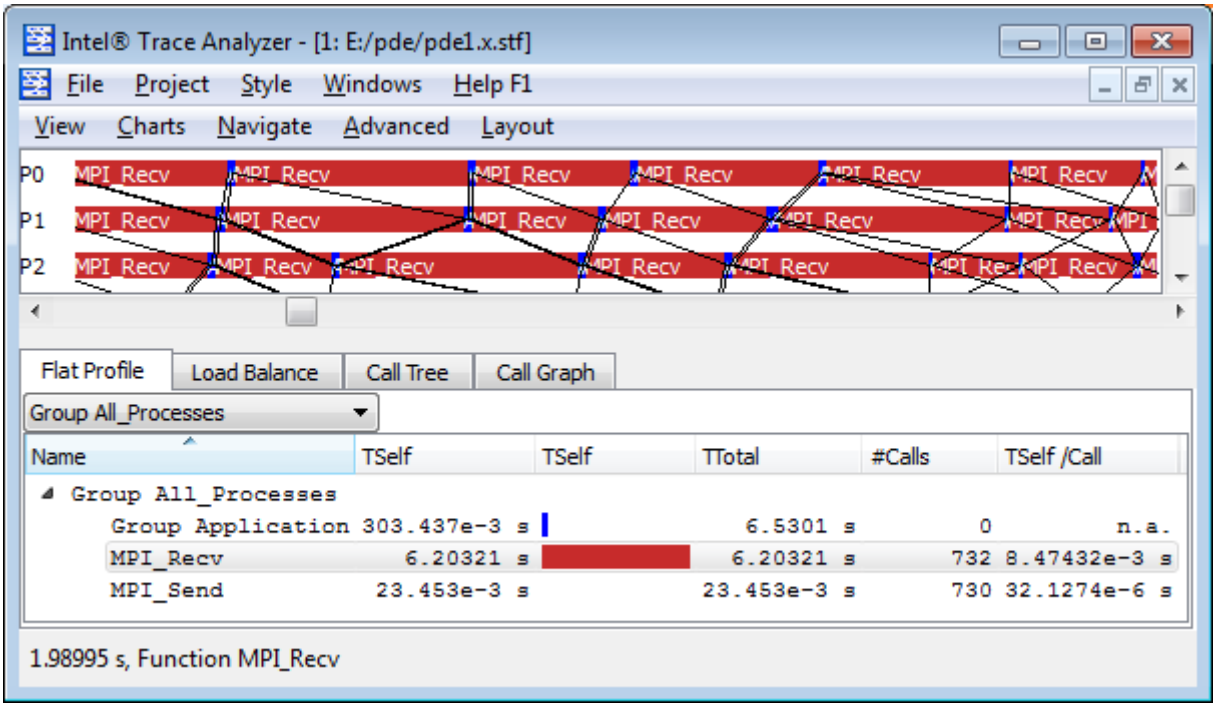

Рис. 1. Визуализация пространственно-временной диаграммы и статистики по функциям в системе Intel Trace Analyzer

Анализ производительности с использованием статистических данных

В рамках системы Aksum [3] рассматривается описание характеристик производительности (performance property), которые рассчитываются на основе статистических данных (например, общее время, затраченное приложением при вызове функций барьерной синхронизации). В результате эта система не способна диагностировать целый класс проблем производительности, возникающих в результате взаимодействия процессов MPI.

В работе [4] предлагается рассматривать проблемы производительности на уровне моделей взаимодействия процессов. Рассмотрены, в частности, такие модели, как «управляющий-рабочие», «конвейер», «разделяй и властвуй». Используя информацию о модели, реализуемой в анализируемой программе, система рассчитывает соответствующие ей метрики. Например, в модели «управляющий-рабочие» система рассчитывает степень загруженности управляющего процесса, высокое значение которой может свидетельствовать о том, что рабочим процессам назначается слишком маленький объем задач.

Эти системы рассчитаны на использование граничных значений. В статье [5] отмечается, что сравнение с граничным значением не всегда применимо при анализе производительности, т.к. сложно заранее определить граничные значения, которые можно

использовать для всех возможных приложений. На данный момент эти системы не доступны для широкого использования, что не позволяет в полной мере судить об их возможностях.

Анализ производительности с помощью шаблонов неэффективного взаимодействия процессов

Как отмечается в [6], для понимания причин возникновения проблем производительности недостаточно рассматривать только статистические данные (например, данные о суммарном времени выполнения каждой из процедур MPI). Для приложений, использующих библиотеки передачи данных, главную сложность в интерпретации данных производительности составляет временная и пространственная отдаленность между причиной и симптомом проблемы производительности. Для диагностики производительности MPI-приложений необходимо анализировать взаимодействие процессов, и такая возможность реализована в системе KOJAK.

Система KOJAK (Kit for Objective Judgement and Knowledge-based Detection of Performance Bottlenecks [7]) состоит из нескольких компонентов, из которых в рамках данного обзора наибольший интерес представляют 2 средства:

- Библиотека EARL (Event Analysis and Recognition Language), которая предоставляет интерфейсы для C++ и Python для обработки трассы выполнения программ.
- Система EXPERT (EXtensible PERformance Tool) для автоматизации диагностики проблем производительности. Эта система использует библиотеку EARL и состоит из набора подпрограмм, каждая из которых предназначена для обнаружения проблемы производительности определенного типа.

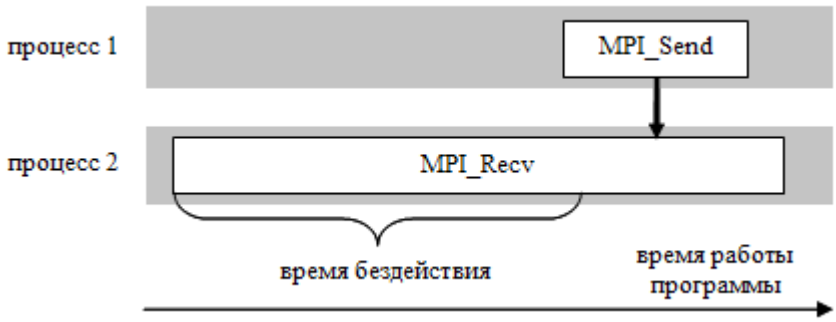

Рис. 2. Поздняя посылка данных (Late Sender)

Примером проблемы производительности, которая диагностируется системой KOJAK, является позняя посылка данных при операции двухточечного обмена сообщениями (Late Sender). Эта проблема возникает, когда блокирующая функция приема данных вызывается позднее функции посылки и в результате первая функция должна простаивать, ожидая данные (см. рис. 2).

Для визуализации выявленных проблем производительности применяется система CUBE. На рис. 3 показано окно CUBE для трассы рассматриваемого MPI-приложения, на котором явно обозначена проблема поздней посылки данных. В рамках эксперимента исходная программа была изменена для совмещения обменов данными и вычислений, в результате чего общее время ее работы уменьшилось с 42.11 до 18.18 секунд (параметры программы: матрица 6400x6400, 1000 итераций, 64 процесса; параметры кластера: узлы с двумя двухъядерными процессорами Intel Xeon 2.66 GHz, 4 Gb оперативной памяти, сеть Gigabit Ethernet).

Итак, система KOJAK позволяет автоматизировать выявление проблем производительности, которые возникают при взаимодействии MPI процессов. К недостаткам этой системы можно отнести то, что типы распознаваемых проблем производительности за-

ранее определены и для их расширения необходимо изменять исходный код системы EXPERT. Попытка решить эту проблему была сделана в системе KappaPI 2 [8]. Но в настоящее время эта система пока еще недоступна для использования, что не позволяет в полной мере судить о ее возможностях.

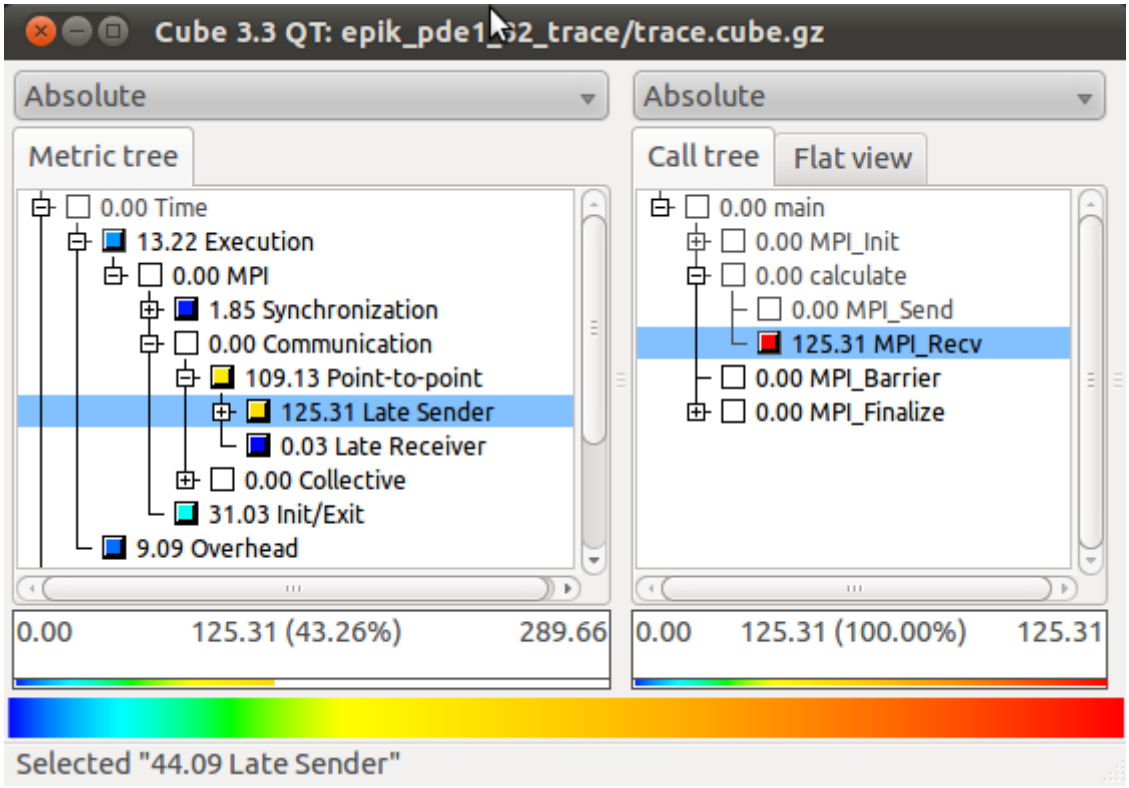

Рис. 3. Визуализация неэффективного взаимодействия процессов в CUBE

Подход системы KOJAK может быть также применен для анализа других трасс. Например, в работе [9] рассматривается применение этого подхода для анализа производительности UPC-приложений. В рамках этой работы разработан трассировщик UPC-программ и описан набор шаблонов неэффективного поведения, позволяющий автоматически выявлять типовые проблемы производительности UPC-приложений. Остальные компоненты для реализации этой системы взяты из системы KOJAK (система CUBE и библиотека EARL).

Сравнительный анализ рассмотренных подходов

В этом обзоре рассмотрены три подхода к автоматизации анализа производительности параллельных приложений. При использовании первого подхода задача выявления причин недостаточной производительности выполняется пользователем. Второй подход - это попытка решить задачу автоматизации анализа производительности, но, как было уже замечено и отмечается в [6], для понимания причин возникновения проблем производительности недостаточно рассматривать только статистические данные. Необходимо анализировать взаимодействие процессов. Эта задача решается третьим подходом, который реализован в системе KOJAK.

Недостатоком системы KOJAK является то, что в этой системе не используется декларативный подход для описания шаблонов. Это затрудняет задачу их описания и поддержки для экспертов по библиотеке MPI, в том числе задачу адаптации для новых стандартов MPI, а также затрудняет ее использование для других задач анализа (например, поиск ошибок в MPI программе) или для других библиотек параллельного программирования.

Таким образом, актуальной является задача разработки декларативного языка для описания шаблонов неэффективного взаимодействия процессов. В работе [10] описана система, которая решает эту задачу.

Литература

1. Intel Trace Analyzer. Reference Guide. URL:http://software.intel.com/en-us/articles/intel-trace-analyzer-and-collector-documentation (дата обращения: 01.09.2011).
2. Fahringer T., Gerndt M., Mohr B. et al. Knowledge Specification for Automatic Performance Analysis: Tech. rep.: ESPRIT Working Group on Automatic Performance Analysis: Resources and Tools, 2001.
3. Fahringer T., Seragiotto C. Aksum: a performance analysis tool for parallel and distributed applications // Performance Analysis and Grid Computing. 2004. No. 2. P. 189-208.
4. Li L., Malony A.D. Knowledge engineering for automatic parallel performance diagnosis // Concurrency and Computation: Practice and Experience. 2007. Vol. 19, No. 11. P. 1497-1515.
5. Hellerstein J. L. How Expert is Your Expert System for Performance Measurement // Proceedings of the Computer Management Group's International Conference. 1994. URL: http://www.cmg.org/proceedings/1994/94INT015.pdf.
6. Hermanns M. Verifying Causal Connections between Distant Performance Phenomena in Large-Scale Message-Passing Applications // Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing. 2009. P. 78-84.
7. Wolf F., Mohr B. Automatic performance analysis of hybrid MPI/OpenMP applications // Journal of Systems Architecture: the EUROMICRO Journal. 2003. Vol. 49, No. 10-11. P. 421-439.
8. Jorba J., Margalef T., Luque E. Search of Performance Inefficiencies in Message Passing Applications with KappaPI 2 Tool // Proceedings of the 8th international conference on Applied parallel computing: state of the art in scientific computing. 2006. P. 409-419.
9. Андреев Н.Е. Исследование и реализация эффективных методов анализа производительности параллельных программ: Кандидатская диссертация. 2011.
10. Карпенко С.Н., Дергунов А.В. Программные средства повышения производительности MPI-приложений // Супервычисления и математическое моделирование. Труды XII международного семинара. Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2011. С. 195-202.