2. Востокин С.В. Templet: язык разметки для параллельного программирования. СГАУ им. академика С.П. Королёва, Самара, 2014.

3. Таненбаум Э. Распределенные системы. Принципы и парадигмы. «Питер», Санкт-Петербург, 2003

Ю.В. Орлов

ОСНОВНЫЕ ПРОБЛЕМЫ РЕАЛИЗАЦИИ СРЕДЫ КОМПЛЕКСНОГО АНАЛИЗА ПРОИЗВОДИТЕЛЬНОСТИ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ ОПТИМИЗАЦИИ[4]

(Вычислительный центр им. А.А. Дородницына
Федерального исследовательского центра «Информатика и управление» Российской академии наук)

Многие задачи глобальной оптимизации относятся к классу NP и их решение требует значительных вычислительных ресурсов. Поэтому представляется целесообразным применение методов параллельных [1,2] и распределенных вычислений [3]. Метод ветвей и границ (МВГ) является одним из наиболее распространенных алгоритмов решения задач дискретной оптимизации. В его основе лежит идея декомпозиции, которая делает естественным применение параллельных вычислений. Обзор различных подходов можно найти в работах [4-6]. Основной проблемой при параллельной реализации методов типа ветвей и границ является адекватная балансировка вычислительной нагрузки между параллельными процессорами. Так как информационный граф алгоритма [7], в данном случае представляющий собой дерево, заранее не известен, особую важность получают методы динамической балансировки, перераспределяющие вычисления в процессе работы в зависимости от загрузки процессоров.

В ближайшее время ожидается, что суперкомпьютеры обретут производительность порядка экзафлопс (около 10^{18} операций с плавающей точкой в секунду). Основным способом увеличения производительности является наращивание количества ядер, число которых в некоторых современных системах уже превосходит 10^{16}. В такой ситуации балансировка нагрузки становится весьма нетривиальной задачей. Следовательно, необходим развитый инструментарий для анализа производительности алгоритмов балансировки. Часто без визуализации, используя одну лишь трассу, выявить причины потери производительности практически невозможно, так как трассы выполнения параллельных программ обычно очень велики по объему и слабо поддаются визуальному анализу.

[4] Работа выполнена при поддержке РФФИ (проекты № 16-07-00873 А и № 16-07-00458 А).

Основные принципы работы среды

Для того, чтобы проводить исследование алгоритмов балансировки без проведения ресурсоемких вычислений, был разработан симулятор многопроцессорной системы BNB-Simulator[8] на базе компонентов библиотеки BNB-Solver[9,10]. Данная библиотека предлагает набор модулей для разработки параллельных приложений. Симулятор представляет собой приложение, управляемое через файл настроек settings.json. Он имитирует выполнение реального параллельного приложения. Результатом работы симулятора является трасса, которая имеет тот же формат, что и трасса параллельного приложения, разработанного на основе BNB-Solver.

Среда комплексного анализа позволяет анализировать трассу, загруженную из файла с расширением «.trc». Таким образом, визуальная среда дает возможность работать с трассой установленного формата, независимо от источника ее получения – реального приложения или симулятора (Рис. 1).

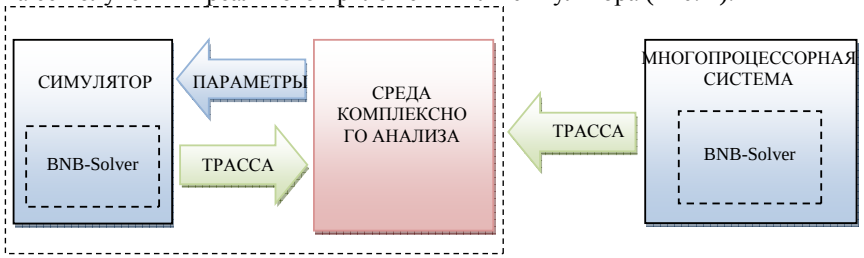

Рис. 1. – Общая схема работы визуальной среды

Схема отображения трассы показана на рис. 2.

Изначально предполагалось, что среда комплексного анализа в процессе своей работы будет хранить данные трассы внутри двумерного массива процессоров размерности n*m, где n – количество процессоров, а m – количество проработанного времени, измеряемого специальными метками. Однако такой подход оказался неприемлемым при обработке трасс большого размера, так как при превышении лимита оперативной памяти, программа начинает задействовать пространство жесткого диска машины, что в свою очередь значительно замедляло ее работу, а в ряде случаев приводило к аварийному завершению программы.

Таким образом, понадобился более гибкий способ обработки трассы. Было принято решение делить в процессе обработки трассу на множество файлов, состоящих из результирующей и содержательной частей. Содержательная часть описывает работу подмножества процессоров на некотором промежутке времени. Результирующая часть содержит итоговые статистические данные о результате работы подмножества процессоров на установленном промежутке времени. Для обеспечения быстрого доступа к таким данным в процессе счета программы результирующая часть помещается в начале файла.

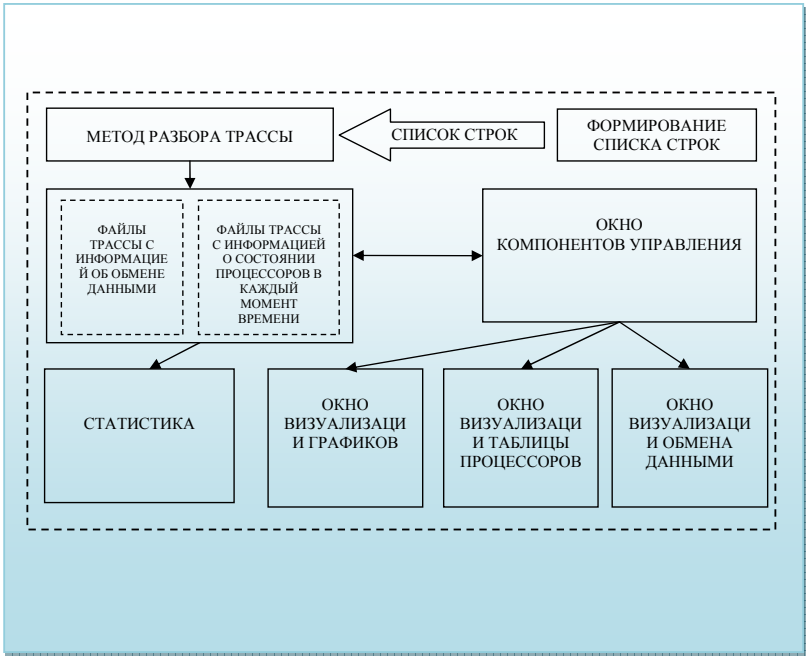

Рис.2. Схема отображения трассы

Для отражения низкоуровневых событий (обмен данными) внутри многопроцессорной системы из трассы извлекается информация о таких событиях и сохраняется в отдельные файлы.

Основные компоненты визуальной среды

Окно компонентов управления представляет собой интерфейс, напоминающий мультимедийный проигрыватель. Данное окно содержит полосу прокрутки, максимальное значение которой равно максимальному значению логического времени работы системы, кнопки для управления прокруткой («Rewind», «Play», «Pause», «Stop», «Forward») и слайдер для регулирования скорости воспроизведения действий процессоров.

Для более детального исследования поведения процессов на определенном промежутке времени удобно использовать окно визуализации графиков производительности. В данном окне поведение каждого процесса представлено в виде набора графиков на координатной плоскости, отображающими состояние процесса в конкретный момент времени. Красным цветом отображается состояние ожидания, синим – состояние счета, зеленым – состояние отправки данных другому процессу.

Такой подход удобен при небольшом (не более 10) числе процессоров. Если запуск параллельного приложения осуществляется на довольно большом количестве узлов, удобнее использовать окно визуализации таблицы процессоров. Каждый процесс в данном окне изображен в виде квадрата. В зависимости от своего состояния квадрат меняет цвет. Такой способ визуализации не позволяет нам исследовать поведение процесса на отрезке времени, зато дает возможность видеть состояние всей системы в каждый момент времени ее работы. Для полноценного анализа параллельного алгоритма оптимизации одной лишь информации о состоянии процессов недостаточно. Часто при отладке алгоритма по трассе нужно обладать информацией о коммуникации процессов. С этой целью в среду комплексного анализа было добавлено окно визуализации обмена данными. визуализация дает возможность выявлять логические ошибки в алгоритме балансировки нагрузки. Так, например, можно выявлять тупиковые ситуации по трассе, используя простейший алгоритм: если все процессы находятся в ожидании на последней метке времени, и при этом не зафиксировано ни одной передачи между ними, можно сделать вывод, что один или несколько процессов захватили ресурсы и по какой-то причине не передали их другим процессам, ожидающим данные ресурсы.

Заключение

В данной работе рассмотрены программная реализация и основные функции среды комплексного анализа производительности алгоритмов балансировки в параллельном методе ветвей и границ.

Среда комплексного анализа производительности параллельных алгоритмов оптимизации является удобным и многофункциональным расширяемым средством для исследования алгоритмов балансировки нагрузки процессоров при решении задач глобальной оптимизации методом ветвей и границ. Данное программное средство позволяет осуществлять как визуальный, так и автоматизированный анализ производительности алгоритмов.

Литература

1. М. А. Посыпкин, И. Х. Сигал, Исследование алгоритмов параллельных вычислений в задачах дискретной оптимизации ранцевого типа // Ж. вычисл.матем. и матем. физ., 2005, том 45, номер 10, С. 1801–1809.

2. И. Х. Сигал, Я. Л. Бабинская, М. А. Посыпкин Параллельная реализация метода ветвей и границ в задаче коммивояжера на базе библиотеки BNB-Solver комплексах // Труды ИСА РАН 2006. Т. 25, С.26-36.

3. Distributed Computing and Its Applications. // Felicity Press, Bristol,USA, 2005.ISBN: 0-931265-10-2, 298p. Монография (соавторы: S.V. Emelyanov, A.P. Afanasiev, Y.R. Grinberg, V.E. Krivtsov,B.V. Peltsverger, O.V. Sukhoroslov, R.G. Taylor, V.V. Voloshinov).

4. Лупин С. А., Посыпкин М. А. Технологии параллельного программирования: Учеб. пос //Сер. Высш. образ-ние. М.: Форум Инфра-М. – 2008. – Т. 208. – С. 2000.

5. Gendron B., Crainic T. G. Parallel branch-and-branch algorithms: Survey and synthesis //Operations research. – 1994. – Т. 42. – №. 6. – С. 1042-1066.

6. Стронгин Р. Г., Гергель В. П., Баркалов К. А. Параллельные методы решения задач глобальной оптимизации //Известия высших учебных заведений. Приборостроение. – 2009. – Т. 52. – №. 10. – С. 25-33.

7. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. БХВ-Петербург. 2002. С. 329. 13.

8. Страница проекта BNB-Simulator https://github.com/fominandrey/bnb-simulator.

9. Evtushenko Y., Posypkin M., Sigal I. A framework for parallel large-scale global optimization //Computer Science-Research and Development. 2009. Т. 23. №. 3-4. С. 211-215.

10. Страница проекта BNB-Solver https://github.com/mposypkin/BNB-solver.

Е.В. Пальчевский, А.Р. Халиков

ПАРАЛЛЕЛИЗАЦИЯ НАГРУЗКИ АППАРАТНО-ПРОГРАММНОГО ЯДРА В UNIX-СИСТЕМАХ

(Уфимский государственный авиационный технический университет)

В данной статье рассматривается задача максимального увеличения пропускной способности сетевого стека с взаимодействием аппаратно-программного ядра для обеспечения стабильности работы физического сервера [1]. Анализируется возможность принимать до десяти миллионов входящих сетевых пакетов с помощью программных средств физического сервера, которая позволяет обеспечить стабильную обработку информации ля бесперебойной работы при *DDoS*-атаках.

При *DDoS*-атаках, в программном ядре, происходят многочисленные сбои, ошибки и перегрузки [2]. Это приводит к замедленной работе всей электронно-вычислительной машины и происходит повышение нагрузки на центральные процессоры. Данные по мощности внешних сетевых угроз, за счет которых можно произвести сетевую перегрузку, представлены на рисунке 1.

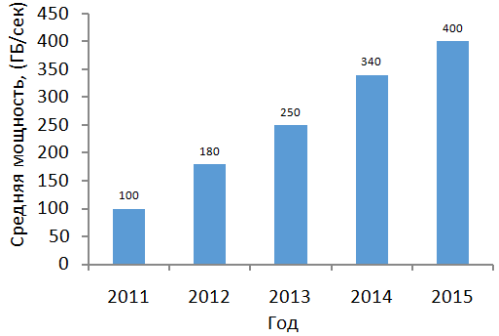

Рисунок 1 – График *DDoS*-атак в годовом эквиваленте за 2011-2015 [2]

Параллелизация программного ядра – это равномерное распределение нагрузочной способности центральных процессоров по физическим ядрам сервера. Подобная операция дает возможность повысить производительность [3].

Появляется возможность повышения устойчивости против разносторонней отправки вредоносного сетевого трафика, направленного с разных ЭВМ [4, 5].

1. Перераспределение нагрузки по физическим ядрам

Для перераспределения нагрузки по ядрам, необходимо включить технологию «*PROMISC*», за счет этого в сетевом стеке срабатывает ускорительный режим для одного ядра [6]. Становится возможным многократное увеличение входящих сетевых пакетов. Запуск производится командой «*ifconfig eth*6 *promisc*». При *DDoS*-атаке, после проделанной процедуры, возникает загруженность первого ядра на максимально возможную. Это позволяет распределять нагрузки *TCP*/*IP*-стека по всем ядрам сервера.

Для того, чтобы организовать просмотр скорости (кбит/с) внешнего сетевого интерфейса нужно применить разработанный скрипт, фрагмент которого написан на языке программирования «*BASH*» [7]:

```
#!/bin/bash
    R1=`cat /sys/class/net/$1/statistics/rx_packets`
    T1=`cat /sys/class/net/$1/statistics/tx_packets`
    sleep $INTERVAL
    R2=`cat /sys/class/net/$1/statistics/rx_packets`
    T2=`cat /sys/class/net/$1/statistics/tx_packets`
    TXPPS=`expr $T2 - $T1`
    RXPPS=`expr $R2 - $R1`
    echo "TX $1: $TXPPS pkts/s RX $1: $RXPPS pkts/s"
done
```

Пример использования вышеприведенного кода, в скриптовом файле: «*bash /root/speed.sh eth0*». Файл «*speed.sh*» создается в системной папке «*root*», с помощью команды «*touch speed.sh*», со вставкой программных строк, которые приведены выше. В ответ, при *DDoS*-атаке, будет выведено сообщение о том, что скорость на внешнем интерфейсе около четырех миллионов пакетов в секунду. Количественная размерность сетевых пакетов зависит от настроек физического сервера.

Таблица – 1 Нагрузка на ядра при применении оптимизации распределенной нагрузки на физический сервер

№ ядра	1	2	3	4	5	6	7	8	9	10	11	12
Нагрузка, %	55,1	52,5	62,5	62,5	57,7	47,7	55,9	61,4	55,1	52,5	62,5	62,5
№ ядра	13	14	15	16	17	18	19	20	21	22	23	24
Нагрузка, %	57,7	47,7	55,9	61,4	55,1	52,5	62,5	62,5	57,7	47,7	55,9	61,4

Для того, чтобы уменьшить нагрузку на первом ядре, был написан специальный скрипт на языке программирования «*BASH*», который позволяет реализовать возможность равномерного распределения нагрузки по всем физическим ядрам: