На правах рукописи

Андреев Никита Евгеньевич

ИССЛЕДОВАНИЕ И РЕАЛИЗАЦИЯ ЭФФЕКТИВНЫХ МЕТОДОВ АНАЛИЗА ПРОИЗВОДИТЕЛЬНОСТИ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ

05.13.11 — математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Автореферат
диссертации на соискание ученой степени
кандидата технических наук

Томск – 2011

Работа выполнена в Кемеровском государственном университете
на кафедре ЮНЕСКО по Новым Информационным Технологиям

Научный руководитель: доктор физико-математических наук,
профессор Афанасьев Константин Евгеньевич

Официальные оппоненты: доктор физико-математических наук,
профессор Старченко Александр Васильевич

кандидат технических наук
Романенко Алексей Анатольевич

Ведущая организация: Институт вычислительных технологий,
г. Новосибирск

Защита состоится 23 июня 2011 г. в 10.30 на заседании диссертационного совета Д 212.267.08 при Томском государственном университете по адресу: 634050, г. Томск, пр. Ленина, 36, корп. 2, ауд. 102.

С диссертацией можно ознакомиться в научной библиотеке ГОУ ВПО «Томский государственный университет» по адресу: 634050, г. Томск, пр. Ленина, 34а.

Автореферат разослан 20 мая 2011 г.

Учёный секретарь
диссертационного совета Д 212.267.08
доктор технических наук, профессор  А. В. Скворцов

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность работы

Создание параллельных приложений – сложный и трудоемкий процесс, поэтому разработчики часто используют инструменты анализа производительности для оптимизации программного кода. Особенно актуальной задача становится в условиях низкой эффективности параллельных приложений на современных высокопроизводительных вычислительных системах, которая составляет по разным оценкам от 3 до 15 % от пиковой производительности.

В условиях постоянного увеличения количества ядер в вычислительных системах старые подходы к анализу производительности становятся непродуктивными. Такие методы, как профилировка и анализ временных шкал, требуют ручного поиска узких мест параллельной программы, что снижает их эффективность и увеличивает трудоемкость поиска с ростом объема поступающих данных. Для оптимизации современных приложений необходимы методы автоматизированного анализа. Среди работ в данном направлении можно выделить B.R. Helm, B.P. Miller, H.L. Truong, J. Vetter, F. Wolf. В России направление анализа производительности развивается в научных коллективах под руководством С.М. Абрамова, А.В. Бухановского, Вл.В. Воеводина, В.П. Гергеля, Б.М. Глинского, В.П. Иванникова, В.А. Крюкова, В.Э. Малышкина, С.А. Немнюгина, Л.Б. Соколинского, А.В. Старченко, Ю.И. Шокина.

Стандартом де-факто для разработки параллельных программ на сегодня является Message Passing Interface (MPI). Несмотря на сравнительно высокую производительность, процесс разработки MPI-программ сложен и подвержен ошибкам, а саму библиотеку иногда называют «ассемблером параллельного программирования». Альтернативой MPI является набирающая популярность модель программирования с разделенным глобальным адресным пространством (Partitioned Global Address Space – PGAS), использующая логически общую память и односторонние коммуникации. Самым «взрослым» представителем модели является язык Unified Parallel C (UPC). Программы, написанные в этой модели, проще для понимания и имеют сравнимую с MPI производительность, а сама модель поддержана со стороны производителей аппаратного обеспечения.

В настоящее время ощущается нехватка инструментальных средств для новой модели, которые бы позволили вывести ее на уровень, сравнимый с MPI по распространенности среди разработчиков. Существующие инструменты, такие как Parallel Performance Wizard (PPW) и upc_trace, предоставляют только статистическую информацию о работе программы. Это требует от программиста последовательного просмотра всех диаграмм и графиков и ручного поиска проблем производительности. Кроме того, для параллельных приложений, генерирующих большой объем трассировочной информации, средства PPW и upc_trace вносят высокий процент накладных расходов на ввод/вывод данных, что снижает точность последующего анализа. К недос-

таткам PPW также можно отнести необоснованно большой размер файлов трасс.

Таким образом, актуальной является задача разработки эффективных методов и подходов к анализу производительности для новой программной модели разделенного глобального адресного пространства, которые бы позволяли выполнять автоматизированный поиск узких мест и снижать влияние на работу параллельных приложений.

Цель работы

Развитие эффективных методов анализа производительности для новой модели параллельного программирования с разделенным глобальным адресным пространством, а также создание программного средства, реализующего данные методы.

Задачи исследования

1. Исследовать методы и алгоритмы анализа производительности параллельных программ, а также модели и методы создания программных систем для высокопроизводительных вычислительных комплексов. Выявить принципы построения инструментальных средств, а также требования, предъявляемые к их реализации.

2. Разработать эффективный подход к анализу производительности, позволяющий идентифицировать узкие места UPC-программ в автоматизированном режиме.

3. Разработать алгоритм трассировки, минимизирующий влияние на выполнение параллельных приложений.

4. Разработать модель программного средства, учитывающую требования к поддержке автоматизированного анализа и минимизации накладных расходов.

5. Реализовать инструментальное средство с учетом разработанных подходов, моделей и алгоритмов. Провести его тестирование на параллельных приложениях.

Методика исследований

Для решения задач, обеспечивающих достижение поставленной цели, использовались методы объектно-ориентированного, системного и структурного программирования, аппарат теории множеств и математической логики, подходы к оптимизации последовательных и параллельных программ, системный анализ, методы объектно-ориентированного проектирования в нотации UML.

Научная новизна

1. Впервые предложен набор шаблонов неэффективного поведения, позволяющий автоматически фиксировать типовые проблемы производительности UPC-программ.

2. Предложен алгоритм трассировки, минимизирующий накладные расходы на анализ и сокращающий объем выходных данных.

3. Предложена модель программного средства, позволяющая создавать на ее основе инструменты для программной модели PGAS, реализующие

подход автоматизированного анализа и оказывающие низкое влияние на работу программ.

4. Реализовано инструментальное средство анализа производительности UPC-программ, отличающееся от других существующих средств поддержкой автоматизированного анализа.

Практическая ценность

1. Разработанные в работе модели, подходы и алгоритмы могут быть использованы при реализации программных средств для других языков программной модели PGAS, таких как: Co-Array Fortran, Titanium, Fortress, Chapel, X10.

2. Разработанное инструментальное средство позволяет выполнять оптимизацию параллельных программ, что сокращает время счета и увеличивает эффективность использования дорогостоящих ресурсов вычислительных установок. Инструмент может использоваться на вычислительных системах как с общей, так и с распределенной памятью.

3. В основе средства лежит метод автоматизированного анализа, снижающий трудоемкость и сокращающий тем самым время, затрачиваемое на полный цикл разработки параллельных приложений и программных комплексов.

Внедрение результатов работы

1. Программное средство было использовано для анализа производительности и последующей оптимизации следующих приложений: реализации итерационного метода Якоби для решения СЛАУ, полученной при дискретизации уравнения Пуассона для трехмерной структурированной сетки, реализации алгоритма блочной сортировки и реализации алгоритма быстрого преобразования Фурье. Оптимизированные версии последних двух алгоритмов вошли в зарубежный пакет NAS Parallel Benchmarks.

2. Реализованный программный комплекс используется в рамках программы «Университетский кластер» представителями научного общества Кемеровского государственного университета, занимающимися численным моделированием.

3. Теоретические и практические результаты работы используются в учебном процессе при преподавании дисциплины «Параллельное программирование» на кафедре ЮНЕСКО по НИТ Кемеровского государственного университета.

Положения, выносимые на защиту

1. Набор шаблонов неэффективного поведения, позволяющий автоматизировать поиск проблем производительности в UPC-программах.

2. Алгоритм трассировки UPC-программ, позволяющий снизить накладные расходы на трассировку и сократить объем выходных данных.

3. Модель программного средства, описывающая компоненты инструмента и связи между ними.

4. Реализация инструментального средства для вычислительных машин с общей и с распределенной памятью.

5. Результаты анализа и оптимизации параллельных приложений при помощи разработанного программного средства.

Апробация работы

Основные результаты диссертации докладывались на следующих научных конференциях и семинарах: VII Всероссийской научно-практической конференции с международным участием «Информационные технологии и математическое моделирование» (Анжеро-Судженск, 2008); Девятой международной конференции-семинаре «Высокопроизводительные Параллельные Вычисления на Кластерных Системах» (Владимир, 2009); Пятой Сибирской конференции по параллельным и высокопроизводительным вычислениям (Томск, 2009); XVII Всероссийской научно-методической конференции «Телематика'2010» (Санкт-Петербург, 2010); IX Всероссийской научно-практической конференции с международным участием «Информационные технологии и математическое моделирование (ИТММ-2010)» (Анжеро-Судженск, 2010); Выставке-конференции «Supercomputing 2010» (New Orleans, 2010); семинаре ИВМиМГ СО РАН под руководством профессора Глинского Б.М. (Новосибирск, 2010); семинаре ИСИ СО РАН «Потоковая обработка данных и программирование» под руководством Непомнящих В.А. и Шилова Н.В. (Новосибирск, 2010); научных семинарах «Информационные технологии и математическое моделирование» кафедры ЮНЕСКО по НИТ КемГУ под руководством профессора Афанасьева К.Е. (Кемерово, 2007–2011).

Личный вклад

Основные научные результаты получены автором самостоятельно. Постановка задачи была выполнена автором совместно с научным руководителем. Разработка набора шаблонов неэффективного поведения, подхода к трассировке, архитектуры и реализация инструментального средства были проведены лично. Внедрение оптимизированных при помощи инструмента алгоритмов в зарубежный пакет NAS Parallel Benchmarks выполнено автором совместно с разработчиками пакета.

Структура и содержание диссертационной работы

Диссертация состоит из введения, четырех глав, заключения и списка цитируемой литературы. Общий объем диссертации составляет 136 страниц машинописного текста. Библиографический список состоит из 111 литературных источников.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ

В первой главе проведено исследование предметной области. Выполнен обзор методов анализа производительности и инструментов для программной модели PGAS. В результате исследования в существующих средствах были выявлены недостатки, обосновывающие актуальность разработки нового программного средства. Были сформулированы требования, предъявляе-

мые к реализации инструментальных средств. Проведен обзор языков программирования класса PGAS. В качестве наиболее «взрослого» представителя модели выделен язык Unified Parallel C (UPC).

Методы анализа производительности можно разделить на несколько категорий. По этапу, на котором выполняется анализ, – на методы предварительного анализа, интерактивные методы и постобработку (post-mortem analysis). По уровню автоматизированности – на ручные и полуавтоматические. Некоторые стратегии анализа и поиска узких мест могут быть в той или иной степени автоматизированы, другие полностью полагаются на пользователя при анализе и интерпретации данных.

Существующие в настоящий момент инструменты для языка UPC, такие как Parallel Performance Wizard (PPW), upc_trace, upc_dump, предоставляют пользователю статистическую информацию о работе программ в виде диаграмм, графиков и таблиц. Это затрудняет анализ, так как требует от пользователя визуального поиска проблем производительности. Ни один из рассмотренных инструментов не имеет средств автоматизированного анализа. Кроме того, представленные программные средства используют алгоритмы трассировки программ, которые приводят к сравнительно высоким накладным расходам и генерируют большие объемы выходных данных, что снижает точность анализа.

Также в процессе обзора были выделены требования к инструментальным средствам анализа производительности параллельных программ, которые состоят в следующем:

- обеспечивать низкий объем накладных расходов по отношению к общему времени выполнения программы;
- иметь средства для ручной корректировки и управления процессами инструментовки и сбора данных;
- иметь сравнительно низкие требования к объему дискового пространства для хранения файлов трасс и результатов анализа;
- скрывать от пользователя сложности параллельной архитектуры программно-аппаратного комплекса при помощи соответствующих скриптов и утилит для запуска и анализа приложений;
- предоставлять в качестве результатов анализа метрики, адекватные для данной программной модели;
- по возможности автоматизировать процесс анализа для снижения нагрузки на пользователя;
- позволять сравнивать данные нескольких экспериментов для оценки результатов оптимизации;
- указывать пользователю на конкретную строку кода, в которой возникла проблема производительности.

Во второй главе автором предложен подход к автоматизированному анализу производительности параллельных программ для языка UPC. Приводится набор шаблонов неэффективного поведения, описывающий типовые

проблемы производительности UPC-программ, а также методика их автоматического обнаружения.

Существующие подходы к автоматизированному анализу производительности используют приемы и методы из области искусственного интеллекта, такие как экспертные системы на основе баз знаний и методы автоматической классификации. Наиболее развитым из таких методов является разработанный F. Wolf и B. Mohr метод поиска шаблонов неэффективного поведения, реализованный для библиотеки MPI и набора директив OpenMP. Метод используется в активно развивающемся наборе инструментов Scalasca. Суть метода заключается в следующем. Различным моделям параллельного программирования присущ ряд типовых проблем производительности или, другими словами, шаблонов неэффективного поведения. Разработчик как эксперт в определенной модели может разработать набор таких шаблонов и в процессе анализа программы выполнять поиск этих шаблонов, последовательно проверяя все записи файла трассы на соответствие каждому из них. По сути, шаблон – это набор найденных в трассировочном файле событий, которые удовлетворяют условиям возникновения некоторой ситуации, которую он описывает. Такой способ представления шаблона позволяет фиксировать сложные ситуации, не охватываемые профилировочными инструментами и визуализаторами трасс.

F. Wolf и B. Mohr предложили набор шаблонов неэффективного поведения для моделей передачи сообщений и общей памяти. Автором был предложен набор из 12 шаблонов неэффективного поведения для модели разделенного глобального адресного пространства. Семь шаблонов из OpenMP и MPI были адаптированы для языка UPC, среди них: конкуренция за блокировку, ожидание в барьере, завершение барьера, поздняя рассылка, ранняя сборка, синхронизация на входе в коллективную операцию многие ко многим, синхронизация на выходе из коллективной операции многие ко многим. Также были разработаны 5 новых шаблонов, характерных только для языка UPC, среди них: синхронизация на входе в коллективную операцию, синхронизация на выходе из коллективной операции, ранняя префиксная редукция, ожидание внутри коллективной операции, ожидание в операции динамического выделения памяти. Приведем для примера один из шаблонов.

Шаблон «Синхронизация на входе в коллективную операцию» справедлив для всех операций релокализации, в которых используется тип синхронизации *UPC _ IN _ ALLSYNC* (рис. 1). Данный тип синхронизации обязывает каждую нить дождаться на входе всех остальных. Когда нити входят в операцию релокализации в разные моменты времени – это вносит нежелательные накладные расходы на синхронизацию. Шаблон не встречается в программной модели передачи сообщений и характерен только для языка UPC. Это объясняется тем, что стандарт MPI не накладывает строгих ограничений на порядок входа нитей в коллективную операцию. Будут ли нити ожидать друг друга, как при использовании барьерной синхронизации, или же будет

использован более оптимальный алгоритм, определяется реализацией, и этого нельзя отследить. В случае с UPC синхронизация указывается программистом явно, что позволяет сформулировать данный шаблон.

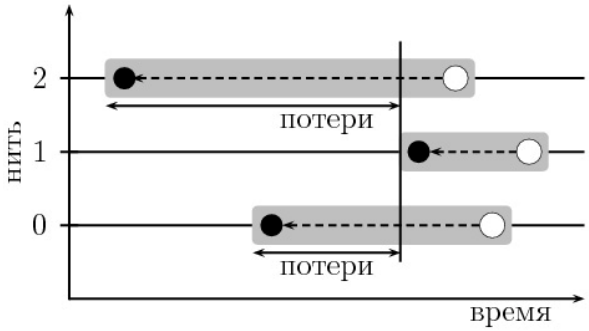

Рисунок 1. Шаблон «Синхронизация на входе
в коллективную операцию»

Шаблон имеет следующий алгоритм поиска. Предположим, что в данном примере используется операция релокализации $upc\_all\_broadcast()$, которая выполняет рассылку значения переменной всем нитям. На первом этапе для всех событий трассы проверяется следующее условие:

$$type(last) = CExit \wedge$$
$$coll(last) \neq \varnothing \wedge$$
$$last.reg = upc\_all\_broadcast.$$

Если встретилось событие выхода из коллективной операции, это событие выхода последней нитью и функцией, из которой был выполнен выход, является $upc\_all\_broadcast()$, то выполняется переход к анализу. Здесь функция $type()$ возвращает тип события, $coll()$ возвращает непустое множество, если все нити завершили выполнение коллективной операции и пустое множество в обратном случае. Атрибут reg для каждого события содержит название функции, с которой связано событие.

Далее проверяется условие срабатывания шаблона:

$$E_1 := coll(last),$$
$$E_2 := \begin{cases} E_1.enterptr, \text{ если } \exists e_i, e_j \in E_1.enterptr : e_i.time \neq e_j.time, \\ \text{неудача, иначе.} \end{cases}$$

Множество E_1 содержит события выхода из функции $upc\_all\_broadcast()$ каждой из нитей. Далее выполняется проверка, если нити вошли в операцию не одновременно, то есть существуют события с разным временем входа в операцию, то E_2 присваиваются все события входа. В данном случае $enterptr$ – это атрибут события выхода из операции, который указывает на соответствующее событие входа.

Если шаблон сработал, то на последнем этапе рассчитывается время, потерянное на синхронизацию:

$$wasted = \sum_{e \in E_2} \max(E_2.time) - e.time.$$

В третьей главе предлагается алгоритм трассировки UPC-программ, который вносит меньший процент накладных расходов и позволяет значительно сократить объем выходных данных по сравнению с существующими аналогами. Приводится модель инструментального средства и описание его реализации.

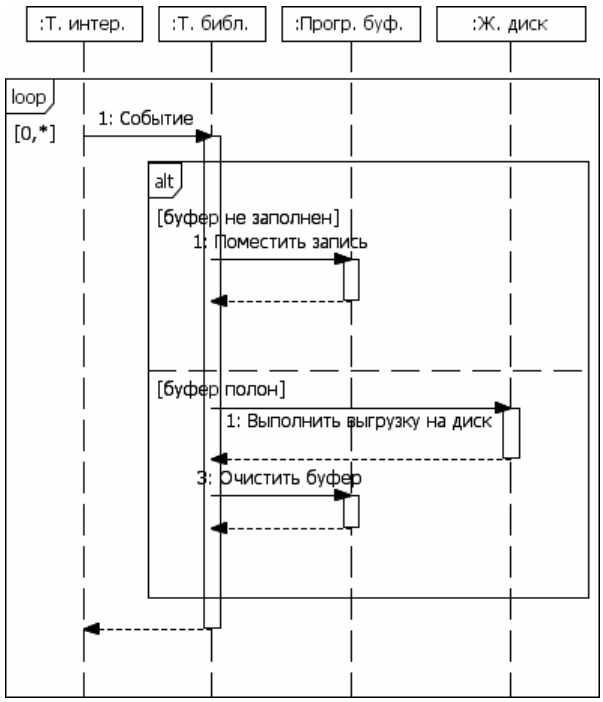

Рисунок 2. Алгоритм трассировки инструмента PPW

Одним из основных показателей качества инструмента анализа производительности является объем вносимых в работу исходной программы накладных расходов во время трассировки. Сильное влияние на приложение в процессе его выполнения может исказить результаты последующего анализа. В PPW, наиболее развитом инструменте для языка UPC – используется следующий алгоритм (рис. 2). Информация о возникающих в программе событиях поступает от трассировочного интерфейса в трассировочную библиотеку. Библиотека содержит промежуточный программный буфер, в котором накапливаются записи. После того как буфер заполняется, данные выгружаются на жесткий диск и буфер очищается. Достоинство такого алгоритма в использовании буферизации, что сокращает количество обращений к медленному внешнему накопителю. Однако запись на диск выполняется синхронно, что требует приостановки параллельного приложения на время записи.

Предлагается расширить базовый алгоритм следующим образом (рис. 3). Первая модификация заключается в использовании библиотеки сжатия в процессе трассировки. Приложения могут генерировать большое количество событий, что приводит к формированию трасс размером в десятки гигабайт. Сжатие «на лету» позволяет, во-первых, сократить размер трасс и, во-вторых, снизить накладные расходы на трассировку за счет сокращения количества записываемых на диск данных. Вторая предлагаемая модификация состоит в идее использования двух «моргающих» буферов и технологии асинхронного ввода/вывода.

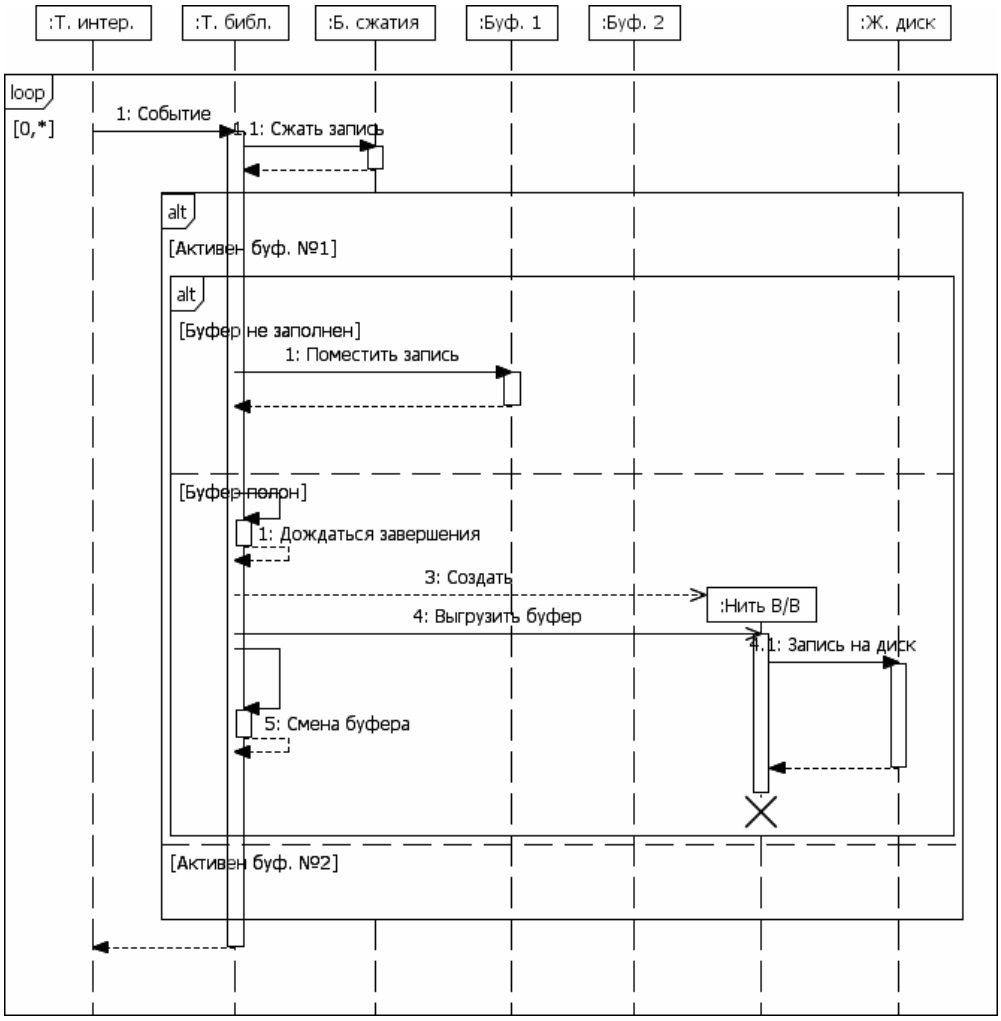

Рисунок 3. Расширенный алгоритм трассировки

После того как запись о событии поступает в трассировочную библиотеку, она сжимается при помощи библиотеки сжатия и помещается в первый буфер. В момент переполнения буфера создается дополнительная нить, которая осуществляет вывод данных на диск параллельно работе основной программы, а запись продолжается во второй буфер. После заполнения второго буфера трассировочная библиотека дожидается завершения предыдущей операции ввода/вывода, если она не успела завершиться к этому моменту, и алгоритм повторяется.

Ввод и вывод данных во всех современных внешних накопителях осуществляются при помощи технологии Direct Memory Access (DMA), разгружающей центральный процессор. Таким образом, предложенный подход снижает накладные расходы, не оказывая в то же время дополнительного влияния на программу пользователя при создании вспомогательной нити.

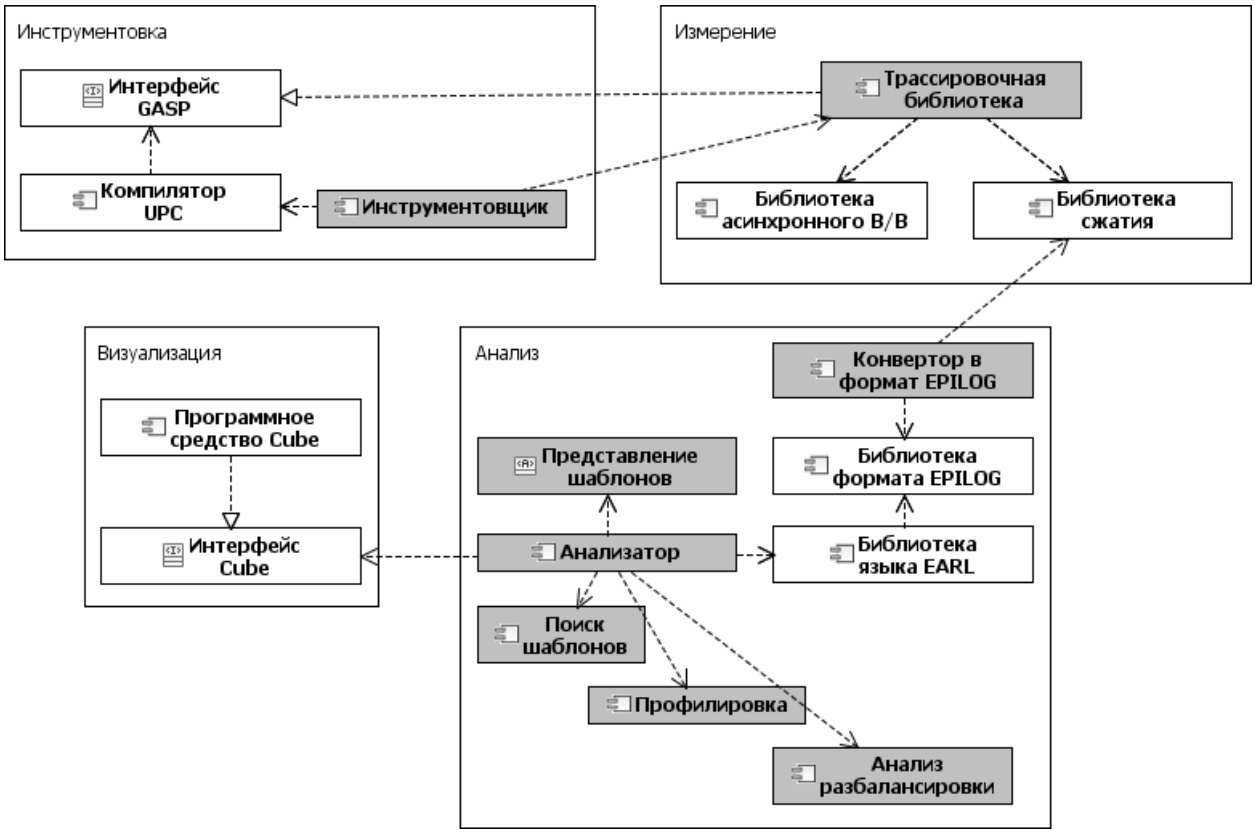

Рисунок 4. Модель архитектуры программного средства

С учетом разработанного подхода к анализу и алгоритма трассировки была разработана модель архитектуры программного средства (рис. 4). Цикл работы с инструментом состоит из четырех этапов – инструментовки, измерения, анализа и визуализации. На первом этапе пользователь компилирует приложение при помощи инструментовщика. Инструментовщик добавляет в исходную программу измерительный код, который будет сообщать о событиях, возникающих в процессе работы программы, а также связывает (link) приложение с трассировочной библиотекой. На втором этапе программа запускается на вычислительной машине. В процессе работы приложения трассировочная библиотека выполняет сжатие информации о событиях и их запись при помощи технологии асинхронного ввода/вывода на диск. На третьем этапе выполняется анализ собранных данных. В качестве интерфейса доступа к файлам трасс предлагается использовать язык распознавания и анализа событий (Event Analysis and Recognition Language – EARL). EARL избавляет от необходимости ручного разбора (парсинга) трасс и представляет дан-

ные о событиях в виде классов C++ с соответствующими методами для доступа к ним. EARL обрабатывает трассы, сформированные в формате EPILOG, поэтому перед непосредственным анализом необходима конвертация. Кроме модуля, выполняющего поиск шаблонов неэффективного поведения, в модель добавлены модули профилировки и анализа разбалансировки нагрузки, которые являются базовыми для большинства инструментов. После завершения анализа формируется XML-файл с результатами для передачи в программное средство Cube. На четвертом этапе результаты визуализируются при помощи Cube.

На рисунке 4 компоненты, разработанные автором, выделены темно-серым цветом. Для поддержки процесса работы с инструментом были разработаны ряд скриптов-оберток для компиляции, запуска, конвертации, анализа и визуализации, а также документация.

В четвертой главе приводится сравнение разработанного инструментального средства с другими инструментами анализа для языка UPC, а также тестирование программного средства на ряде параллельных приложений.

Для сравнения алгоритма трассировки, реализованного в инструменте PPW, и алгоритма, разработанного автором, были выбраны пять тестов из пакета NAS Parallel Benchmarks (NPB): метод сопряженных градиентов (Conjugate Gradient – CG), блочной сортировки (Integer Sort – IS), быстрого преобразования Фурье (FT), решение трехмерного уравнения Пуассона (MG – MultiGrid) и алгоритм генерации пар псевдослучайных чисел согласно гауссовому распределению (EP – Embarrassing Parallel).

Таблица 1

Сравнение накладных расходов на трассировку

Тест	Ориг., с	PPW, с	TM, с	PPW, %	TM, %	Разн.
CG	29,45	69,98	53,19	137,61	80,60	1,71
CG pupc	29,45	31,24	29,79	6,08	1,15	5,27
IS	4,17	72,93	39,54	1649,21	848,41	1,94
IS pupc	4,17	4,31	4,28	3,37	2,65	1,27
FT	51,39	55,31	52,98	7,63	3,08	2,47
MG	11,11	11,75	11,30	5,83	1,72	3,39
EP	16,39	16,85	16,62	2,78	1,41	1,97

В таблице 1 представлено время счета для исходных версий программ, общее время работы инструментованных версий в секундах и количество накладных расходов в процентах от общего времени работы программы. Последний столбец показывает, во сколько раз различаются результаты. Здесь TM – инструмент, разработанный автором.

Для тестов CG и IS проводилось два эксперимента. Первый – с трассировкой всех функций программы и второй – с исключением функций, не представляющих большого интереса с точки зрения анализа, но генерирующих чрезмерный объем трассировочной информации. Результаты первого

эксперимента наглядно показывают преимущество улучшенного алгоритма, однако не позволяют сразу использовать эти данные для анализа.

Накладные расходы для тестов FT, MG, EP и скорректированных версий тестов CG, IS также ниже, чем у PPW. Для теста CG алгоритм асинхронного ввода/вывода показывает результат в 5 раз лучше, чем синхронный алгоритм. Для теста MG накладные расходы ниже в 3,5 раза. В среднем TM вносит в 2–3 раза меньший процент накладных расходов. Это объясняется неэффективностью алгоритма трассировки PPW, который часто блокирует работу программы.

Также оценивался объем генерируемых трассировочных данных. В таблице 3 представлено сравнение размера файлов трасс для пяти тестов из пакета NPB для 32 нитей.

Наилучшие результаты показывает профилировщик upc_trace. Однако собираемых им данных недостаточно для полноценного анализа. PPW и TM показывают результаты одного порядка. Использование улучшенного алгоритма (TM_Z в таблице 2) позволяет снизить размер трасс в несколько раз. К примеру, для теста IS сжатие «на лету» позволило сократить размер трассы в 6 раз – с 18ГБ до 3.1ГБ, тогда как размер трассы PPW для теста IS составляет 17ГБ.

Таблица 2

Сравнение объема трассировочных данных

Тест	upc_trace, МБ	PPW, МБ	TM, МБ	TM_Z, МБ
CG	940	12695	13429	2583
EP	542	66	72	16
FT	479	775	820	192
IS	506	17401	18427	3167
MG	674	183	176	33

Тестирование модуля анализа проводилось на трех приложениях: алгоритмах параллельной блочной сортировки (Integer Sort – IS) и быстрого преобразования Фурье (Fast Fourier Transform – FFT) из пакета NAS Parallel Benchmarks (NPB) и метода Якоби.

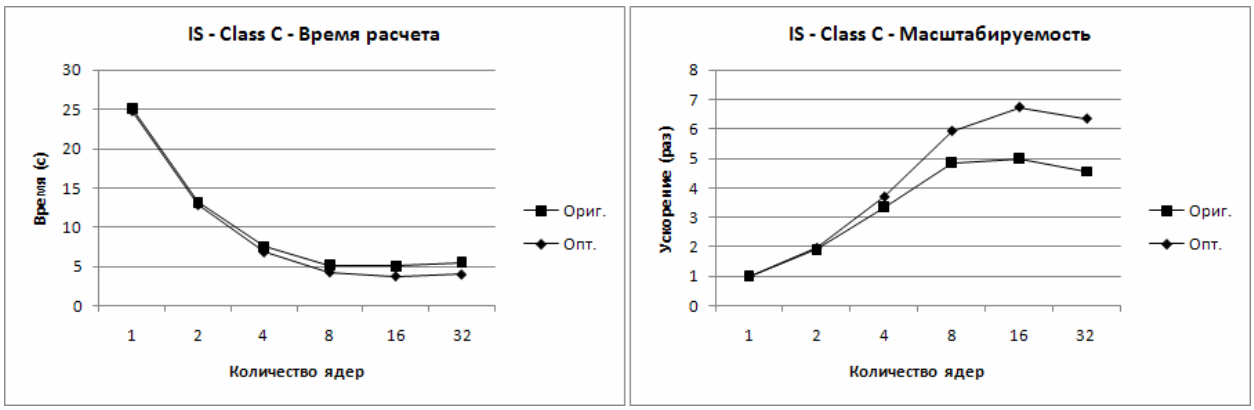

Рисунок 5. Результаты оптимизации блочной сортировки

Алгоритм **блочной сортировки** используется в задачах, основанных на методе частиц, и характерен для приложений физики, где частицы принадлежат ячейкам и могут перемещаться между ними. Сортировка используется для переназначения частиц соответствующим ячейкам. Алгоритм представляет собой интерес по двум причинам: во-первых, он используется в качестве основы для решения реальных задач физики «частиц в ячейках» и, во-вторых, пакет NPB используется для анализа эффективности работы кластеров и хорошо оптимизирован. Вероятность нахождения проблем производительности в подобных приложениях низкая и вызывает особые трудности.

Для анализа данного приложения использовались такие возможности инструмента, как: шаблон «Ожидание в операции динамического выделения памяти», шаблон «Ожидание в барьере», анализ разбалансировки нагрузки, а также профилировочные данные о количестве вызовов функций и времени, затраченного на операции перемещения данных.

В результате оптимизации алгоритма удалось добиться сокращения времени выполнения до 30 % в случае 32 нитей. На рисунке 6 представлены графики времени расчета и масштабируемости для оригинальной и оптимизированной версий программы.

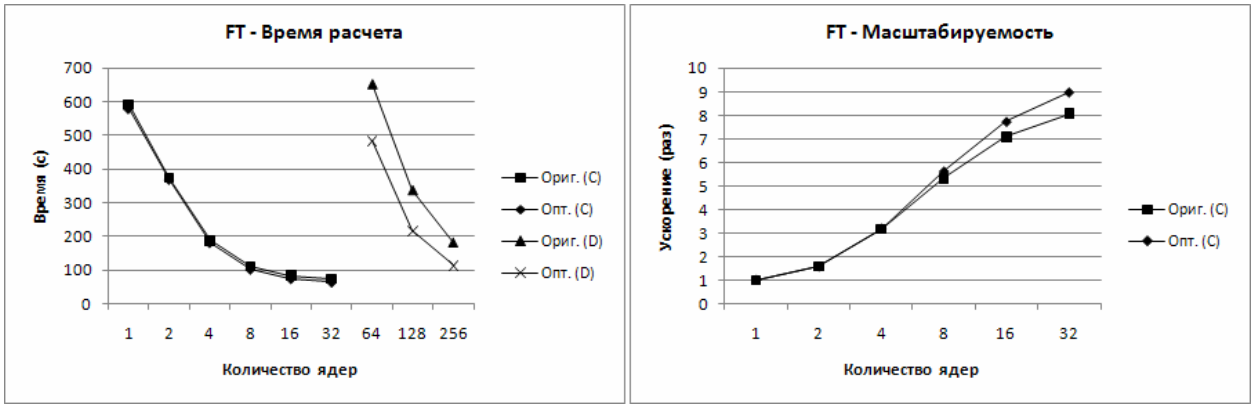

Рисунок 6. Результаты оптимизации преобразования Фурье

Второе приложение представляет собой решение трехмерного дифференциального уравнения при помощи **быстрых преобразований Фурье**. Преобразование Фурье используется во многих областях науки – в физике, статистике, криптографии, акустике, оптике и многих других. Поэтому анализ и оптимизация данного алгоритма для языка UPC, как и для любого другого, имеют большую значимость.

В данном примере использовались результаты срабатывания шаблона «Ожидание в барьере», анализа разбалансировки нагрузки и профилировки для крупных пересылок данных. Был сразу рассмотрен проблемный участок и выявлены причины неэффективного поведения. Последующая оптимизация программы позволила сократить время выполнения на 37 % для 256 нитей. На рисунке 6 представлены графики времени расчета и масштабируемости для оригинальной и оптимизированной версий программы. Первые две кривые с пометкой (C) соответствуют расчетам с размером матрицы 512×512×512

и 20 итерациями. Для кривых с пометкой (D) использовалась матрица размером 2048×1024×1024 и 25 итераций.

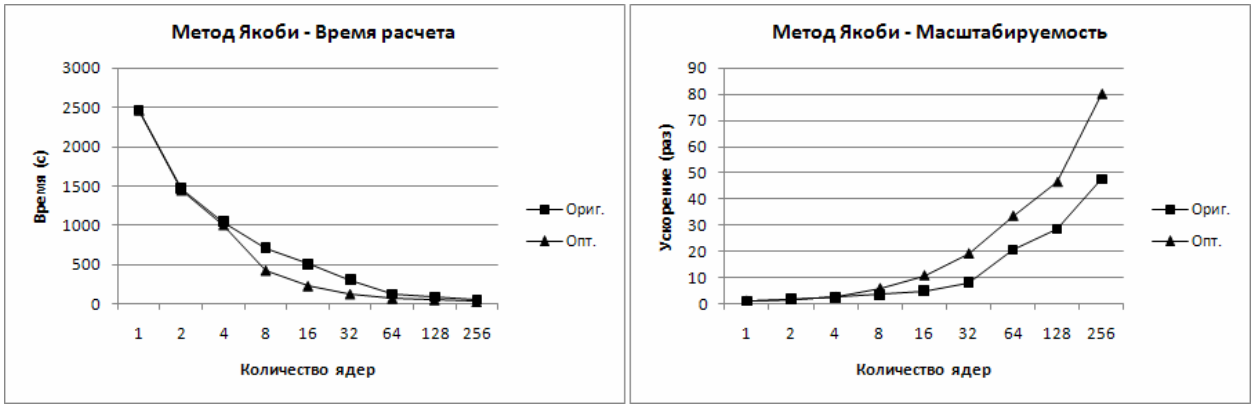

Рисунок 7. Результаты оптимизации метода Якоби

В качестве главного средства для анализа **метода Якоби** были использованы профилировочные данные о распределении времени выполнения между функциями. Были выявлены основные «горячие точки» приложения, которые требуют оптимизации. Дальше более подробное изучение кода программы позволило определить конкретные функции программы и инструкции языка UPC, ответственные за снижение производительности. В результате оптимизации удалось ускорить алгоритм на 40 % для 256 нитей. Максимальное ускорение составило 58 % для 32 нитей. На рисунке 7 представлены графики времени расчета и масштабируемости для оригинальной и оптимизированной версий программы. Оптимизированная версия для 256 нитей показывает масштабируемость почти в два раза выше оригинальной.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ РАБОТЫ

1. **Разработан перечень требований** к программным средствам анализа производительности на основании результатов проведенного обзора инструментов и методов. Выявлены достоинства и недостатки существующих подходов.

2. **Предложен набор шаблонов** неэффективного поведения для анализа производительности UPC-программ, позволяющий находить узкие места в автоматизированном режиме. Данная возможность реализуется за счет описания типовых проблем, присущих программной модели, и автоматической проверки программы на их наличие. Автору не известны другие работы по применению каких бы то ни было методов автоматизированного анализа к программной модели с разделенным глобальным адресным пространством.

3. **Разработан алгоритм** трассировки, который вносит меньшие накладные расходы по сравнению с существующими подходами, а также позволяет значительно сократить объем генерируемых выходных данных.

4. **Разработана модель** программного средства, удовлетворяющая сформулированным требованиям. В модели описаны компоненты инстру-

мента и связи между ними. Предложенное архитектурное решение может быть использовано для разработки эффективных инструментальных средств для других языков параллельного программирования.

5. На основе предложенных подходов, моделей и алгоритмов **реализовано инструментальное средство** для языка UPC, отличающееся от других средств поддержкой автоматизированного анализа и обладающее лучшими характеристиками трассировки. При помощи инструмента были проанализированы и оптимизированы реализации алгоритмов блочной сортировки, быстрого преобразования Фурье и метода Якоби, которые могут быть положены в основу эффективных параллельных приложений. Оптимизированные версии первых двух алгоритмов вошли в пакет NAS Parallel Benchmarks.

СПИСОК ОПУБЛИКОВАННЫХ РАБОТ ПО ТЕМЕ ДИССЕРТАЦИИ

Журналы, рекомендованные ВАК для представления основных научных результатов диссертации:

1. Андреев, Н.Е. Организация сбора и подготовки данных для анализа производительности UPC-программ / Н.Е. Андреев, К.Е. Афанасьев // Вестник Кемеровского государственного университета. – 2011. – Вып. 4 (44). – С. 4–10.

2. Андреев, Н.Е. Методы автоматизированного анализа производительности параллельных программ / Н.Е. Андреев // Вестник Новосибирского государственного университета. – 2009. – Т. 7, вып. 1. – С. 16–25.

3. Андреев, Н.Е. Реализация инструментального средства автоматизированного анализа производительности UPC-программ / Н.Е. Андреев, К.Е. Афанасьев // Вычислительные методы и программирование. – 2011. – Раздел 2. – С. 46–57 (http://num-meth.srcc.msu.ru/).

Публикации в других изданиях:

1. Андреев, Н.Е. Обзор методов автоматизированной оценки эффективности выполнения параллельных программ / Н.Е. Андреев. – Кемерово, 2009. – 34 с. – Деп. в ВИНИТИ 22.06.09, № 390-В2009.

2. Андреев, Н.Е. Продуктивный анализ производительности параллельных приложений на мультиядерных и многоядерных архитектурах / Н.Е. Андреев, К.Е. Афанасьев // Многоядерные процессоры и параллельное программирование: материалы региональной научно-практической конференции. – Бийск, 2011. – С. 28–33.

3. Андреев, Н.Е. Система автоматизированного поиска шаблонов неэффективного поведения UPC-программ / Н.Е. Андреев // Тезисы докладов Пятой Сибирской конференции по параллельным и высокопроизводительным вычислениям. – Томск, 2009. – С. 81–82.

4. Андреев, Н.Е. Автоматизированный анализ производительности параллельных программ как способ повышения продуктивности разработчика / Н.Е. Андреев, К.Е. Афанасьев // Информационные технологии и математическое моделирование (ИТММ-2010): материалы IX Всероссийской научно-

практической конференции с международным участием. – Анжеро-Судженск, 2010. – С. 121–125.

5. Андреев, Н.Е. Архитектура системы автоматизированного анализа UPC-программ / Н.Е. Андреев // Телематика'2010: телекоммуникации, веб-технологии, суперкомпьютинг: сборник статей участников Всероссийского конкурса научных работ студентов и аспирантов. – СПб., 2010. – С. 164–166.

6. Андреев, Н.Е. Один из подходов к трассировке параллельных программ в модели разделенного глобального адресного пространства / Н.Е. Андреев // Высокопроизводительные Параллельные Вычисления на Кластерных Системах: материалы Девятой международной конференции-семинара. – Владимир, 2009. – С. 16–20.

7. Андреев, Н.Е. Инструментальное средство автоматизированного анализа производительности UPC-программ / Н.Е. Андреев, К.Е. Афанасьев // Студент и научно-технический прогресс: материалы XLIX Международной научной студенческой конференции. – Новосибирск, 2011. – С. 219–219.

8. Андреев, Н.Е. Использование пакета Scalasca в качестве основы для анализа производительности UPC-программ / Н.Е. Андреев, К.Е. Афанасьев // Научное творчество молодежи: материалы XV Всероссийской научно-практической конференции. – Анжеро-Судженск, 2011. – С. 98–100.

Подписано к печати 17.05.2011. Формат $60\times84^{1}/_{16}$. Печать офсетная.
Бумага офсетная № 1. Усл. печ. л. 1,2. Тираж 100 экз. Заказ № 181.

ООО «Издательство «Кузбассвузиздат».
650043, г. Кемерово, ул. Ермака, 7. Тел.: (3842) 58-29-34