На правах рукописи

Кулагин Иван Иванович

СРЕДСТВА АРХИТЕКТУРНО-ОРИЕНТИРОВАННОЙ ОПТИМИЗАЦИИ ВЫПОЛНЕНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ ДЛЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ С МНОГОУРОВНЕВЫМ ПАРАЛЛЕЛИЗМОМ

Специальность: 05.13.15 – Вычислительные машины, комплексы и компьютерные сети

Автореферат
диссертации на соискание ученой степени
кандидата технических наук

Новосибирск – 2017

Работа выполнена в федеральном государственном бюджетном образовательном учреждении высшего образования «Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ).

Научный руководитель:
Курносов Михаил Георгиевич
доктор технических наук, доцент, федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ), заведующий Кафедрой вычислительных систем.

Официальные оппоненты:

Глинский Борис Михайлович
доктор технических наук, доцент, федеральное государственное бюджетное учреждение науки Институт вычислительной математики и математической геофизики Сибирского отделения Российской академии наук (ИВМиМГ СО РАН), заведующий лабораторией.

Шашев Дмитрий Вадимович
кандидат технических наук, акционерное общество «Научно-исследовательский институт полупроводниковых приборов» (АО «НИИПП»), младший научный сотрудник.

Ведущая организация – федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)» (СПбГЭТУ «ЛЭТИ»).

Защита состоится 15 февраля 2018 г. в 13-30 на заседании диссертационного совета Д 219.005.02 при федеральном государственном бюджетном образовательном учреждении высшего образования «Сибирский государственный университет телекоммуникаций и информатики», по адресу: 630102, г. Новосибирск, ул. Кирова, д. 86, ауд. 625.

С диссертацией можно ознакомиться в библиотеке СибГУТИ и на сайте: `http://www.sibsutis.ru/science/postgraduate/dis_sovets/`.

Автореферат разослан «___» _____ 2017 г.

Ученый секретарь
диссертационного совета Д 219.005.02
к.т.н., доцент  И.В. Нечта

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность темы исследования и степень ее разработанности. Архитектура современных высокопроизводительных вычислительных систем (ВС) характеризуется многоуровневым параллелизмом – на множестве иерархических уровней системы реализуются различные формы параллельной обработки информации. Развитие отечественных и зарубежных ВС идет по пути наращивания степени параллелизма на всех функциональных уровнях. Организация эффективного функционирования современных и перспективных ВС и достижение параллельными программами производительности близкой к пиковой требуют учета в системном инструментарии организации функционирования ВС форм параллелизма всех функциональных уровней.

Значительный вклад в развитие теории и практики вычислительных систем и средств организации их функционирования внесли выдающиеся ученые, среди которых В.С. Бурцев, В.А. Вальковский, А.П. Ершов, В.В. Воеводин, Э.В. Евреинов, А.В. Забродин, В.П. Иванников, М.Б. Игнатьев, А.В. Каляев, И.А. Каляев, Ю.Г. Косарев, В.В. Корнеев, Л.Н. Королев, А.О. Лацис, С.А. Лебедев, В.К. Левин, И.И. Левин, Г.И. Марчук, В.А. Мельников, Н.Н. Миренков, Д.А. Поспелов, И.В. Поттосин, И.В. Прангишвили, Д.В. Пузанков, Г.Г. Рябов, В.Б. Смолов, А.Н. Томилин, В.Г. Хорошевский, Б.Н. Четверушкин, Н.Н. Яненко, W. Carlson, B. Chamberlain, D. Culler, J. Dongarra, D. Grove, M. Herlihy, L. Lamport, T. Knight, T. Riegel, N. Shavit.

Анализ тенденций развития мощнейших высокопроизводительных ВС мира показывает, что развитие архитектуры ВС идет по пути увеличения степени параллелизма на всех уровнях иерархии системы: числа процессоров в элементарных машинах (ЭМ), числа ядер в процессорах, а также количества параллельно работающих скалярных и векторных АЛУ в процессорных ядрах. Эффективное использование всех ресурсов ВС, достижение (суб)пиковой устойчивой производительности требуют разработки инструментальных средств (математических модели, методов и алгоритмов) и параллельных программ, учитывающих многоуровневый параллелизм ВС.

С развитием мультиархитектуры ВС активно развиваются высокоуровневые средства параллельного программирования, ориентированные на сокращение трудозатрат при разработке параллельных программ (high productivity computing). В частности, языки параллельного программирования Unified Parallel C, Coarray Fortran, IBM X10, Cray Chapel, DVM, ParJava, T-система. Значительная часть таких языков реализуют модель разделенного глобального адресного пространства (partitioned global address space, PGAS), которая позволяет абстрагироваться от явного использования в коде программ низкоуровневых операций передачи сообщений при обращении к объектам в памяти удаленных процессов (эле-

ментарных машин). Масштабируемость параллельных PGAS-программ на ВС во многом определяется эффективностью алгоритмов, реализованных в PGAS-компиляторах и их runtime-системах. В частности, остро стоят задачи, связанные с оптимизацией времени доступа в PGAS-программах к совместно используемым структурам данных, находящимся в памяти удаленных ЭМ системы. Решение этих задач связано с разработкой методов оптимизации информационных обменов в PGAS-языках на уровне ВС.

На уровне отдельного многопроцессорного вычислительного узла с общей памятью требуют своего решения задачи сокращения времени доступа ветвей параллельных программ к разделяемым структурам данных. Классические методы синхронизации типа «мьютекс» (взаимное исключение) и «семафор» подразумевают блокирование одновременного выполнения участка кода программы множеством потоков, а не защиту совместно используемой области памяти. Последнее, для значительного класса задач, приводит к образованию очередей ожидания доступа в критическую секцию и снижает масштабируемость программ. Развиваются альтернативные подходы: алгоритмы, свободные от блокировок (lock-free algorithms), и программная транзакционная память (ПТП, software transactional memory, STM). Модель транзакционной памяти получила как аппаратурную реализацию в процессорах IBM BlueGene/Q PowerPC, Intel Haswell (Intel TSX), Oracle Rock (SPARC v9), так и программную реализацию в современных компиляторах и runtime-библиотеках: GCC TM, LazySTM, TinySTM, DTMC, RSTM, STMX, STM Monad. В программных реализациях транзакционной памяти актуальными являются задачи разработки методов и структур данных обнаружения конкурентного доступа потоков программы к разделяемым объектам в памяти.

Эффективное использование ресурсов ВС также требует учета параллелизма на уровнях суперскалярного ядра процессора и векторных арифметико-логических устройств. В связи с активным развитием наборов векторных SIMD-инструкций (Intel AVX, IBM AltiVec, ARM NEON/SVE, MIPS MSA) новую актуальность получили задачи (полу)автоматической векторизации циклов в параллельных программах на процессорах с короткими векторными регистрами.

Цель работы и задачи исследования. Целью работы является разработка и исследование средств архитектурно-ориентированной оптимизации выполнения параллельных программ для ВС с многоуровневым параллелизмом. В соответствии с целью определены следующие задачи исследования.

1. Для ВС с многоуровневым параллелизмом разработать средства оптимизации циклического доступа к информационным массивам в параллельных программах на базе модели разделенного глобального адресного пространства.

2. Для многопроцессорных вычислительных узлов с общей памятью

разработать и исследовать методы сокращения времени выполнения транзакционных секций многопоточных программ в модели программной транзакционной памяти.

3. Разработать средства анализа эффективности векторизации циклов на архитектурах процессоров с короткими векторными регистрами.

4. Разработать пакет программ оптимизации функционирования ВС и использования их многоуровневого параллелизма для решения параллельных задач.

Научная новизна полученных результатов определяется учетом в созданных методах и алгоритмах форм параллелизма основных функциональных уровней ВС и динамических характеристик параллельных задач.

1. Оригинальность созданного алгоритма *Array Preload* трансформации конструкций циклической передачи потока управления подчиненным элементарным машинам заключается в возможности его применения к PGAS-программам, в которых на этапе компиляции неизвестно множество используемых элементов массивов. В отличие от известных, время выполнения кода, формируемого алгоритмом, не зависит от количества итераций циклов.

2. Новизна метода сокращения числа ложных конфликтов в многопоточных программах на базе программной транзакционной памяти заключается в (суб)оптимальной настройке таблиц обнаружения конфликтов с учетом шаблона доступа к памяти из параллельных программ.

3. В отличие от известных работ полученные классы трудно векторизуемых циклов из тестового набора ETSVC сформированы с учетом микроархитектуры современных наборов команд Intel AVX и AVX-512.

4. Созданный инструментарий анализа эффективности использования микроархитектурных возможностей ядер суперскалярных процессоров ВС позволяет анализировать загрузку суперскалярного конвейера архитектуры Intel 64 потоком инструкций с точностью до нескольких машинных команд.

Теоретическая и практическая значимость работы состоит в том, что разработанные в диссертации методы и алгоритмы реализованы в виде системного программного обеспечения анализа и оптимизации использования в параллельных программах многоуровневого параллелизма ВС.

1. Созданный алгоритм *Array Preload* трансформации конструкций циклической передачи потока управления подчиненным элементарным машинам реализован в виде расширения компилятора языка IBM X10. По сравнению со стандартными алгоритмами IBM X10 предложенный позволяет на кластерных ВС с сетями связи Gigabit Ethernet и InfiniBand QDR сократить время выполнения циклического доступа к элементам массивов в 1,2–2,5 раз.

2. Разработанный метод сокращения числа ложных конфликтов в многопоточных программах на базе программной транзакционной памяти реализован в расширении компилятора GCC. Реализация включает модуль

инструментации кода целевой программы и модуль настройки параметров runtime-системы реализации ПТП по результатам профилирования программы. Использование предложенного метода позволяет сократить время выполнения параллельных программ в среднем на 20%, что экспериментально показано на тестовых программах из пакета STAMP.

3. Создан инструментарий анализа эффективности использования микроархитектурных возможностей ядер суперскалярных процессоров ВС. Предложенные средства позволяют анализировать загрузку суперскалярного конвейера архитектуры Intel 64 потоком инструкций с точностью до нескольких команд ассемблера.

4. На процессорах с поддержкой наборов векторных инструкций Intel AVX и AVX-512 экспериментально установлены классы трудно векторизуемых циклов из тестового набора ETSVC. Построенное подмножество циклов составляет базисный набор для анализа эффективности ядер автовекторизаторов оптимизирующих компиляторов для векторных процессоров класса «регистр-регистр».

5. Программные средства внедрены в действующую мультикластерную ВС Центра параллельных вычислительных технологий СибГУТИ и Лаборатории вычислительных систем Института физики полупроводников им. А.В. Ржанова СО РАН (ИФП СО РАН).

Основные этапы исследования выполнены в ходе осуществления работ по проектам Российского фонда фундаментальных исследований №№ 15-07-00653, 15-37-20113; Совета Президента РФ по поддержке ведущих научных школ № НШ-2175.2012.9 (руководитель – чл.-корр. РАН В.Г. Хорошевский); грантов Новосибирской области для молодых ученых, стипендии Президента РФ для аспирантов.

Получено три свидетельства о государственной регистрации программ для ЭВМ. Результаты работы внедрены в учебный процесс СибГУТИ, в систему параллельного мультипрограммирования пространственно-распределенной ВС, что подтверждается соответствующими актами.

Методология и методы исследования. Для решения поставленных задач использовались методы теории вычислительных систем, теории алгоритмов, теории графов. Экспериментальные исследования осуществлялись путем моделирования на вычислительных кластерах с многоуровневым параллелизмом. Работа основана на результатах ведущей научной школы в области анализа и организации функционирования большемасштабных ВС (руководитель – чл.-корр. РАН В.Г. Хорошевский).

Положения и результаты, выносимые на защиту.

1. Алгоритм *Array Preload* трансформации конструкций циклической передачи потока управления подчиненным элементарным машинам, сокращающий время выполнения программ за счет опережающего копирования информационных массивов. В отличие от известных, разработанный метод применим к PGAS-программам, в которых на этапе компиляции неизвест-

но множество используемых элементов массивов.

2. Программная реализация алгоритма *Array Preload*, обеспечивающая на кластерных ВС с сетями связи Gigabit Ethernet и InfiniBand QDR сокращение в параллельных программах на языке IBM X10 времени выполнения циклического доступа к элементам массивов в 1,2–2,5 раз.

3. Метод сокращения числа ложных конфликтов в многопоточных программах на базе программной транзакционной памяти, основанный на подборе (суб)оптимальных параметров таблиц обнаружения конфликтов по результатам предварительного профилирования параллельных программ.

4. Программная реализация метода сокращения числа ложных конфликтов в расширении компилятора GCC, обеспечивающая сокращение времени выполнения транзакционных секций в C/C++-программах в среднем на 20%.

5. Экспериментально построенные на архитектурах с поддержкой наборов инструкций Intel AVX/AVX-512 классы трудно векторизуемых циклов из тестового набора ETSVC. Полученное подмножество циклов составляет базисный набор для анализа эффективности ядер автовекторизаторов оптимизирующих компиляторов для векторных процессоров класса «регистр-регистр».

6. Инструментарий анализа эффективности использования микроархитектурных возможностей ядер суперскалярных процессоров ВС, позволяющий анализировать загрузку суперскалярного конвейера архитектуры Intel 64 потоком инструкций с точностью до нескольких команд ассемблера.

7. Инструментарий параллельного мультипрограммирования кластерной ВС, расширенный созданными автором пакетами оптимизации использования многоуровневого параллелизма в параллельных программах.

Личный вклад. Выносимые на защиту результаты получены соискателем лично. В совместных работах постановки задач и разработка методов их решения осуществлялись при непосредственном участии соискателя.

Степень достоверности и апробация результатов подтверждаются проведенными экспериментами и моделированием, согласованностью с данными, имеющимися в отечественной и зарубежной литературе, а также прошедшими экспертизами работы при получении грантов.

Основные результаты работы докладывались и обсуждались на международных, всероссийских и региональных научных конференциях, в их числе: международные конференции «Parallel Computing Technologies» (PaCT-2015, Petrozavodsk), «Параллельные вычислительные технологии» (ПаВТ-2016, г. Архангельск), «Суперкомпьютерные технологии: разработка, программирование, применение» (СКТ-2014, СКТ-2016, г. Геленджик), «Открытая конференция по компиляторным технологиям» (2015 г., г. Москва); российские конференции: «Актуальные проблемы вычислительной и прикладной математики» (АПВПМ-2015, г. Новосибирск), «Новые информационные технологии в исследовании сложных структур» (ICAM-2014,

г. Томск), Сибирская конференция по параллельным и высокопроизводительным вычислениям (2015 г., г. Томск).

Публикации. По теме диссертации опубликовано 23 работы. Из них 4 – в журналах из перечня ВАК РФ; 2 – в изданиях, индексируемых Scopus и Web of Science. Получено 3 свидетельства о государственной регистрации программ для ЭВМ.

Структура и объем диссертации. Диссертация состоит из введения, четырех глав, заключения, списка литературы и приложений, изложенных на 155 страницах.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ

Во введении изложена актуальность темы исследования и степень ее разработанности, сформулированы цели и задачи работы, отражена ее научная новизна, теоретическая и практическая значимость; представлены положения, выносимые на защиту, а также степень достоверности и апробация результатов.

В первой главе дается понятие распределенных ВС с многоуровневым параллелизмом, рассматривается программно-аппаратный стек технологий для организации параллельных вычислений на таких системах.

Распределенные ВС имеют иерархическую структуру и обладают многоуровневым параллелизмом: параллелизм процессов на уровне вычислительных узлов, параллелизм потоков на уровне процессорных ядер, параллелизм команд на уровне АЛУ и параллелизм данных на уровне векторных АЛУ. Реализация параллельных алгоритмов для таких систем требует задействования всего программного стека. На уровне ЭМ: библиотеки стандарта MPI (message passing interface) – MPICH2, Open MPI, MVAPICH2, Cray MPI, IBM MPI, Intel MPI; библиотеки семейства SHMEM; языки модели разделенного глобального адресного пространства (partitioned global address space – PGAS) – Unified Parallel C, Coarray Fortran, IBM X10, Cray Chapel, DVM, ParJava и др. На уровне многопроцессорного SMP/NUMA-узла с общей памятью разработка ведется при помощи средств, реализующих модель многопоточного программирования: стандарты OpenMP; библиотека потоков уровня ядра операционной системы PThreads, и др. Для использования параллелизма уровней суперскалярного вычислительного ядра процессора и векторных АЛУ современные компиляторы реализуют алгоритмы автоматической векторизации и конвейеризации циклов, планирования команд и другие техники оптимизирующей компиляции.

Рассмотрены модели параллельных вычислений (LogP, LogGP, PLogP, LogGPS, LogPC LogfP и др.), связывающие архитектурные свойства ВС и характеристики параллельных программ.

Вторая глава посвящена оптимизации параллельных программ, разработанных в модели PGAS, для распределенных ВС с многоуровневым параллелизмом.

Модель разделенного глобального адресного пространства (partitioned global address space – PGAS) – это модель параллельного программирования, реализующая общее адресное пространство над распределенной памятью ВС. В модели PGAS доступ из любой ЭМ к глобальному адресному пространству осуществляется как к локальной памяти при помощи высокоуровневых операторов присваивания «=». То есть, модель PGAS позволяет создавать программы для распределенных ВС, используя семантику обращения к памяти из ВС с общей памятью. Достоинство данной модели в том, что, в отличие от модели передачи сообщений, в ней отсутствуют коммуникационные операции. Программа, разработанная в модели PGAS, называется – *PGAS-программой*, а язык программирования, реализующий эту модель, – *PGAS-языком*.

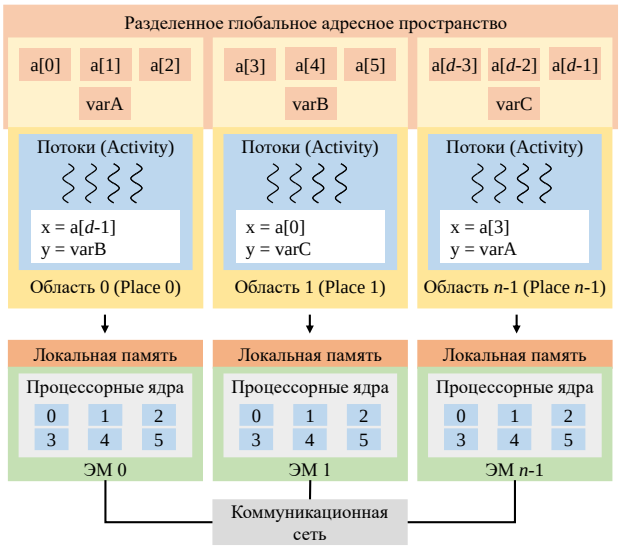


Рисунок 1 – Отображение модели PGAS на распределенную ВС

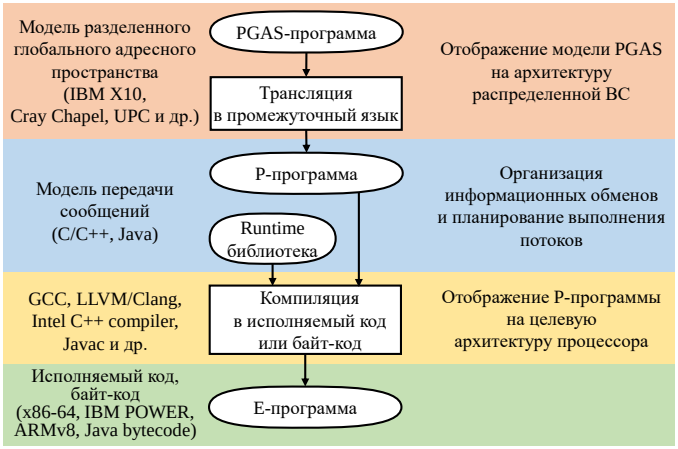


Рисунок 2 – Двухэтапный процесс компиляции PGAS-программы

На рисунке 1 представлена модель разделенного глобального адресного пространства и ее отображение на распределенную ВС из n ЭМ. Ключевым понятием в модели PGAS является *область* (в терминах языка IBM X10 – *Place*), которая представляет абстракцию ЭМ с разделяемой памятью на базе многоядерного процессора. В рамках областей выполняются потоки (в терминах языка IBM X10 – *Activity*). Данные, над которыми потоки выполняют операции, могут храниться либо в памяти текущей области (*локальной памяти*), либо в памяти другой области (*удаленной памяти*). Память множества областей формирует разделенное глобальное адресное пространство. Доступ ко всем данным в глобальном адресном пространстве осуществляется одинаково без явного обращения в PGAS-программе к коммуникационным операциям, вне зависимости от того, где они хранятся, в локальной памяти или удаленной. Каждая область закреплена за определенной ЭМ. Локальная память области является физической памятью ЭМ, а ее потоки выполняются процессорными ядрами ЭМ.

Компиляторы основных языков программирования семейства PGAS являются *транспилерами* (*source-to-source compiler*), или трансляторами, преобразующими код программы, написанной на одном языке, в аналогичный код, но на другом языке. Процесс компиляции PGAS-программы содержит 2 этапа. На первом этапе выполняется трансляция PGAS-программы в промежуточный язык. В процессе трансляции компилятор преобразует языковые конструкции в последовательность вызовов функций runtime-системы, которые содержат обращения к коммуникационным операциям. В результате генерируется *P-программа* в модели передачи сообщений на промежуточном языке программирования, как правило, C/C++ или Java.

На втором этапе выполняется отображение сгенерированной *P-программы* на целевую архитектуру процессора ЭМ (*пост-компиляция*). Во время пост-компиляции применяются методы оптимизации и автоматической векторизации кода. В результате, генерируется *E-программа*, представленная исполняемым кодом или интерпретируемым байт-кодом. В роли пост-компилятора могут выступать промышленные компиляторы GCC, Intel C++ compiler, LLVM/Clang, Javac, PGI, PathScale EKO Compiler Suite и др.

Одним из часто используемых шаблонов в PGAS-программах является циклическая конструкция передачи потока управления подчиненным ЭМ. В этой конструкции содержится цикл из r итераций, на каждой из которых осуществляется передача управления подчиненным ЭМ из множества M при помощи конструкции `at`. На каждой итерации передача управления происходит $m = |M|$ элементарным машинам. В теле конструкции `at` выполняются операции над элементами l массивов $a_0, ..., a_{l-1}$. Предполагается, что операции над элементами массива a_i объединены в функцию f_i, таким образом, последовательность функций $f_0(a_0), ..., f_{l-1}(a_{l-1})$ содержит все операции, выполняющиеся над элементами массивов $a_0, ..., a_{l-1}$. В листинге 1 представлен пример такого шаблона на языке IBM X10. В этом примере функция `f` осуществляет доступ к элементам одного удаленного массива `a` (листинг 1, строка 7). Этот массив копируется runtime-системой из памяти главной ЭМ в память подчиненной ЭМ с номером `j` при выполнении конструкции `at` (листинг 1, строка 6).

Листинг 1 – Доступ подчиненных ЭМ к удаленному массиву a на языке $IBM\ X10$

```
1  val a = new Array[Long](d);
2  val M = Place.places();
3  ...
4  for (i in 0..(r - 1)) {
5      for (j in M) {
6          at (j) {
7              f(a);
8          }
9      }
10 }
```

Для доставки элементов массивов в память подчиненных ЭМ компилятор PGAS-языка преобразует циклическую конструкцию передачи потока управления в последовательность обращений к коммуникационным операциям в *P-программе*. Объем передаваемых данных определяется эффективностью алгоритма трансформации таких конструкций. Недостаток алгоритма *By-Iterative Copying*, являющегося стандартным для языка IBM X10, заключается в избыточном копировании всего массива на каждой итерации цикла. Алгоритм *Scalar Replacement* копирует только используемые элементы массива, однако, он не применим в случаях, когда на этапе компиляции неизвестно множество используемых индексов элементов массива. Для эффективного выполнения таких конструкций в диссертации предложен алгоритм *Array Preload*, являющийся оптимизационным проходом компилятора.

Алгоритм *Array Preload* опережающего копирования массивов. Идея алгоритма заключается в преобразовании конструкции циклической передачи потока управления подчиненным ЭМ таким образом, чтобы при выполнении PGAS-программы копирование используемых данных в память удаленных ЭМ осуществлялось только один раз перед циклом, а не на каждой итерации i. Такой подход принято называть *опережающим копированием массивов* (*preemptive copying*).

Опережающее копирование массивов в удаленную память подчиненных ЭМ выполняется прологом цикла. Основные преобразования, выполняемые алгоритмом *Array Preload*, направлены на формирование пролога цикла для отправки подчиненным ЭМ используемых массивов и создания их локальных копий в памяти ЭМ, а также на преобразование тела конструкции передачи управления. В ходе преобразований тела конструкции обращения к массиву, расположенному в памяти главной ЭМ, заменяются на обращения к его локальной копии в памяти подчиненных ЭМ.

На вход алгоритма подается *абстрактное синтаксическое дерево* (АСД) C конструкции циклической передачи потока управления подчиненным ЭМ из множества M. На выходе алгоритм формирует АСД C' конструкции, АСД P функции BodyAt и АСД H, представляющее функцию для создания локальных копий массивов в памяти подчиненных ЭМ. Алгоритм состоит из 9 шагов.

Шаг 1. Поиск в АСД C поддерева B, которому сопоставлено тело внутреннего цикла, на итерациях которого осуществляется передача потока управления подчиненным ЭМ при помощи конструкции at.

Шаг 2. Поиск в АСД C узла r, которому сопоставлен номер удаленной ЭМ $j \in M$.

Шаг 3. Поиск в АСД B и формирование последовательности узлов $(a_0, ..., a_{l-1})$, которые отображают массивы, чьи элементы используются внутри тела конструкции at.

Шаг 4. Поиск в АСД B и формирование последовательности узлов

$(f_0, ..., f_{l-1})$, которым соответствуют функции, содержащие операции работы с элементами массивов $a_0, ..., a_{l-1}$.

Шаг 5. Построение АСД H функции, выполняющей создание локальных копий массивов $a_0, ..., a_{l-1}$ в памяти подчиненных ЭМ.

Шаг 6. Поиск последовательности массивов $(a'_0, ..., a'_{l-1})$, созданных в локальной памяти подчиненных ЭМ.

Шаг 7. Построение АСД S, соответствующего прологу цикла, который выполняет отправку сообщения подчиненным ЭМ для создания в их памяти локальных копий используемых массивов.

Шаг 8. Построение АСД P функции BodyAt, соответствующей телу конструкции передачи управления подчиненным ЭМ.

Шаг 9. Преобразования АСД B циклической конструкции передачи управления подчиненным ЭМ в АСД B', содержащее узлы для отправки сообщения.

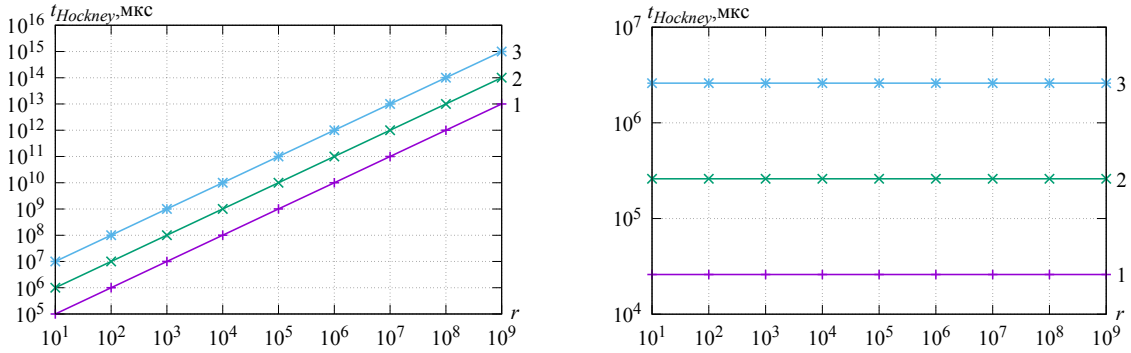
Для генерируемого алгоритмом *Array Prelaod* кода *P-программы* построены оценки времени выполнения в моделях Hockney, LogP и LogGP, а также оценка объема V передаваемых данных главной ЭМ на каждой итерации: $V = m \cdot b \cdot \sum_{i=0}^{l-1} s_i$, где b – размер одного элемента массива в байтах, а s_i – размер массива a_i. Время t выполнения пролога цикла и самой конструкции в моделях LogP, LogGP и Hockney имеют следующий вид:

$$t_{LogP} = gm - g + L + 2o; \qquad t_{LogGP} = ((V/m - 1)G + g)m - g + L + 2o;$$

$$t_{Hockney} = m(\alpha + \beta V/m).$$

Выполнен анализ эффективности известных алгоритмов *By-Iterative Copying*, *Scalar Replacement* и предложенного *Array Preload* преобразования циклических конструкций передачи потока управления подчиненным ЭМ в модели Hockney. На рисунках 3а, 3б показана зависимость времени выполнения циклической конструкции в *P-программе*, полученной алгоритмами *Scalar Replacement* и *Array Preload* соответственно, от числа r итераций при различных значениях m количества подчиненных ЭМ. Предполагалось, что параллельная PGAS-программа содержит одну циклическую конструкцию передачи потока, в теле которой подчиненные ЭМ использовали только 1 элемент массива с типом `double` ($b = 8$ байт). Алгоритм *Scalar Replacement* демонстрирует линейную зависимость времени выполнения конструкции от числа r итераций в цикле, в то время как алгоритм *Array Preload* – константную.

Алгоритм *Array Preload* реализован автором в виде расширения компилятора языка IBM X10. Проведенные эксперименты показали его эффективность. По сравнению со стандартным алгоритмом предложенный позволяет сократить время выполнения циклического доступа ко всем элементам массивов в 1,2–2,5 раз на кластерных ВС с сетями связи Gigabit Ethernet



а) Scaral Replacement *б) Array Preload*

1 – $m = 10^4$; 2 – $m = 10^5$; 3 – $m = 10^6$;

Рисунок 3 – Зависимость времени выполнения копирования массивов в память подчиненных ЭМ при выполнении *P-программы* от числа r итераций в циклической конструкции передачи потока управления $b = 8$ байт, сеть связи – InfiniBand QDR ($\alpha = 1$ мкс, $\beta = 2 \cdot 10^{-4}$ мкс)

и InfiniBand QDR.

Третья глава посвящена методам оптимизации выполнения параллельных программ на многопроцессорных ВС с общей памятью.

При выполнении многопоточной программы на ресурсах многопроцессорной ВС с общей памятью может возникнуть *состояние гонки за данными* (*data race*) – ситуация, при которой множество потоков пытаются одновременно получить доступ к разделяемому ресурсу, к одной области памяти. Возникновение состояния гонки за данными является результатом ошибки при разработке параллельного алгоритма и приводит к некорректной работе программы.

Основными подходами к созданию потокобезопасных программ, не приводящих к возникновению состояния гонки за данными, являются: синхронизация потоков операциями блокировок, разработка неблокирующих алгоритмов и структур данных, использование транзакционной памяти.

Транзакционная память (*transactional memory*) – это подход к созданию потокобезопасных программ, позволяющий предотвратить возникновение состояния гонок за данными без использования блокировок. В отличие от примитивов синхронизации на основе блокировок, транзакционная память позволяет защитить *область памяти программы* от конкурентного доступа множества потоков, а не участок кода от одновременного выполнения. Кроме того, ее использование не требует глубокой переработки кода программы, как в случае использования неблокирующих алгоритмов и структур данных. Существуют как программные реализации транзакционной памяти (*software transactional memory – STM*): LazySTM, TinySTM, GCC TM, DTMC, RSTM, STMX, STM Monad, так и аппаратные реализации в процессорах (*hardware transactional memory – HTM*): Intel TSX, AMD ASF, Oracle Rock, IBM POWER8, IBM PowerPC A2.

Программная транзакционная память предоставляет языковые конструкции или API, позволяющие выделять в программе участки кода – *транзакционные секциии* (*transactional section*), в которых осуществляется защита совместно используемых областей памяти runtime-системой языка, возможно, с использованием возможностей HTM. Для выполнения транзакционных секций runtime-среда создает транзакции. Если во время выполнения транзакции одним потоком произошло изменение защищаемой ею области памяти другими потоками, то транзакция считается некорректной, и ее выполнение прерывается при помощи аппаратных возможностей процессора либо runtime-системой. Все модифицированные в рамках нее области памяти восстанавливаются в исходное состояние, и поток начинает выполнять ее заново.

Проверка корректности выполнения транзакций реализуется путем поддержки runtime-системой метаданных о состоянии защищаемых регионов памяти. Линейное адресное пространство процесса разбивается на фиксированные блоки размера B, каждый из которых сопровождается метаданными о состоянии (подход, подобный прямому отображению физических адресов на кэш-память процессора). Метаданные хранятся в таблице размера S. При такой организации хранения информации о состоянии адресного пространства множеству областей памяти соответствуют одни метаданные, что является источником возникновения ложных конфликтов (рисунок 4). *Ложный конфликт* (*false conflict*) – ситуация, при которой два или более потока во время выполнения транзакции обращаются к разным участкам линейного адресного пространства, но отображаемым на одни и те же метаданные. Поэтому runtime-система воспринимает такую ситуацию как конфликт (data race), хотя на самом деле таковой отсутствует. Автором предложен метод, позволяющий сократить число ложных конфликтов в STM-программах.

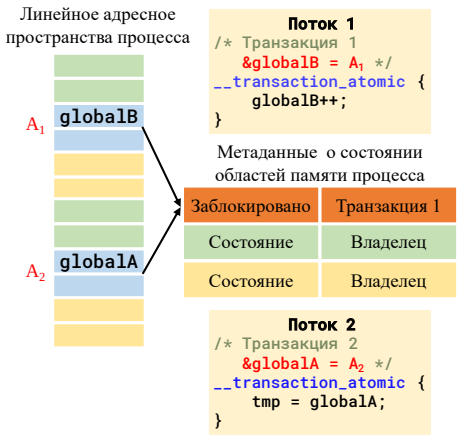


Рисунок 4 – Пример возникновения ложного конфликта при выполнении двух транзакций (GCC 4.7+)

Метод сокращения ложных конфликтов выполняет автоматическую

настройку параметров S и B таблицы метаданных под динамические характеристики конкретной STM-программы. Метод включает три этапа.

Этап 1. Внедрение функций библиотеки профилирования в транзакционные секции. На первом этапе выполняется компиляция C/C++ STM-программы с использованием разработанного модуля анализа транзакционных секций и внедрения вызова функций библиотеки профилирования (модуль расширения GCC).

Этап 2. Профилирование программы. На данном этапе выполняется запуск STM-программы в режиме профилирования. В результате формируется протокол (trace), содержащий информацию о ходе выполнения транзакционных секций.

Этап 3. Настройка параметров таблицы. По протоколу определяется средний размер W читаемой/записываемой области памяти во время выполнения транзакций. По значению W подбираются субоптимальные параметры B и S таблицы, с которыми STM-программа компилируется.

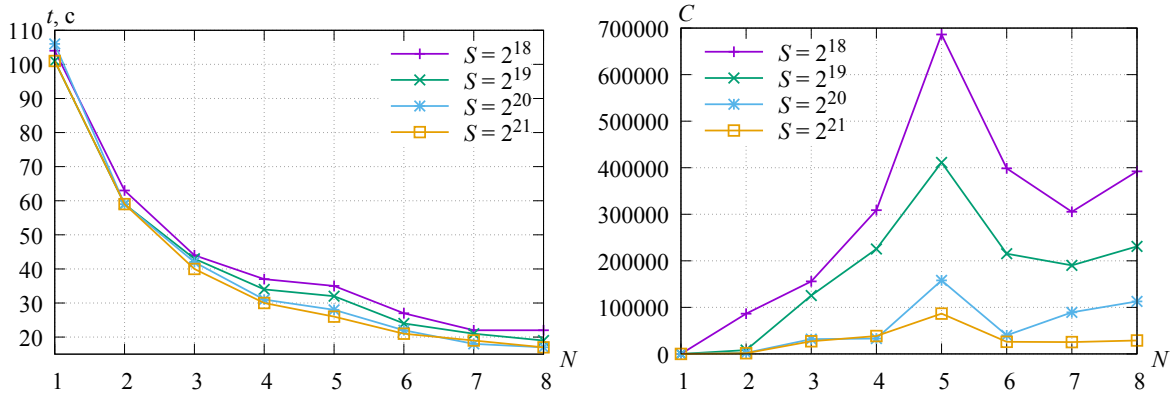
Значение параметра B выбирается следующим образом:
– если $W = 1$ байт, то $B = 2^4$ байт;– если $W = 8$ байт, то $B = 2^7$ байт;
– если $W = 4$ байт, то $B = 2^6$ байт; – если $W >= 64$ байт, то $B = 2^8$ байт.

Экспериментальное исследование проводилось на ВС, оснащенной двумя четырехъядерными процессорами Intel Xeon E5420. В данных процессорах отсутствует поддержка аппаратной транзакционной памяти (Intel TSX). В качестве тестовых программ использовались многопоточные STM-программы из пакета STAMP. Тесты собирались компилятором GCC 5.1.1. Операционная система – GNU/Linux Fedora 21 x86_64.

В рамках экспериментов измерялись значения двух показателей: время t выполнения STM-программы; количество C ложных конфликтов в программе. На рисунках 5б, 5а показана зависимость количества C ложных конфликтов и времени t выполнения теста от числа потоков при различных значениях параметров B и S. Результаты приведены для программы genome из пакета STAMP. В ней порядка 10 транзакционных секций, реализующих операции над хеш-таблицей и связными списками. Видно, что увеличение значений параметров S и B приводит к уменьшению числа возможных коллизий (ложных конфликтов), возникающих при отображении адресов линейного адресного пространства процесса на записи таблицы. При размере таблицы 2^{21} записей, на каждую из которых отображается 2^6 адресов линейного адресного пространства, достигается минимум времени выполнения теста genome, а также минимум числа ложных конфликтов.

Время выполнения теста genome удалось сократить в среднем на 20% за счет минимизации числа ложных конфликтов.

Анализ эффективности подсистем автоматической векторизации циклов в открытых компиляторах. Одним из уровней параллелизма ВС является параллелизм данных на уровне векторных АЛУ вычислительного ядра. Наборы команд практически всех архитектур современных процес-



а) $B = 2^6$ *б)* $B = 2^6$

Рисунок 5 – Зависимость времени t выполнения теста (*а*) и числа C ложных конфликтов (*б*) от числа N потоков

соров включают поддержку векторных команд: Intel SSE/AVX/AVX-512, ARM NEON SIMD, IBM AltiVec, MIPS MSA. Процессоры, реализующие поддержку таких команд, содержат одно или несколько параллельно работающих векторных АЛУ и совокупность векторных регистров. В отличие от векторных систем 1990-х годов, современные процессоры поддерживают выполнение операций с векторами относительно небольшой длины (64 – 512 бит), предварительно загруженными из оперативной памяти в векторные регистры (класс векторных систем «регистр-регистр»).

Основная сфера применения векторных инструкций – сокращение времени работы с одномерными массивами. При векторизации происходит трансформация выполнения итераций обработки массивов данных в векторные инструкции, выполняющиеся одновременно над несколькими экземплярами данных. Как правило, ускорение, достигаемое при использовании векторных инструкций, в первую очередь определяется количеством элементов массива, помещающихся в векторный регистр.

При разработке прикладных программ наиболее востребованы следующие способы использования векторных инструкций: ассемблерные вставки; интринсики (intrinsics); SIMD-директивы компиляторов, стандартов OpenMP, OpenACC; языковые расширения и библиотеки; автоматическая векторизация компилятором. Автоматическая векторизация циклов компиляторами является одной из наиболее значимых методик их оптимизации. Такой способ векторизации не требует значительной модификации прикладных программ и обеспечивает их переносимость на уровне исходного кода между разными архитектурами процессоров.

Выполнена оценка эффективности алгоритмов автоматической векторизации в компиляторах GCC C/C++ и LLVM/Clang на тестах из пакета Extended Test Suite for Vectorizing Compilers (ETSVC), содержащего основные классы циклов, встречающихся в научных приложениях на языке C. Исходная версия пакета была разработана в конце 1980-х годов груп-

пой Дж. Донгарры и содержала 122 цикла на языке Fortran для оценки эффективности компиляторов векторных ВС. В 2011 году группа Д. Падуа транслировала пакет TSVC на язык программирования C и дополнила его новыми циклами. Расширенная версия пакета содержит 151 цикл. Циклы разделены на категории: анализ зависимостей по данным (36 циклов), анализ потока управления и трансформация циклов (52 цикла), распознавание идиоматических конструкций (редукции, рекуррентности и т.п., 27 циклов), полнота понимания языка программирования (23 цикла). Кроме этого, в набор включены 13 контрольных циклов – тривиальные циклы, с векторизацией которых должен справиться каждый векторизующий компилятор.

Экспериментальное исследование алгоритмов в открытых компиляторах GCC C/C++ 6.3.0, LLVM/Clang 3.9.1 проводилось на системе, представляющей собой сервер на базе двух процессоров Intel Xeon E5-2620 v4 (архитектура Intel 64, микроархитектура Broadwell, 8 ядер, Hyper-Threading включен, поддержка набора векторных инструкций AVX 2.0), 64 Гбайта оперативной памяти DDR4, операционная система GNU/Linux CentOS 7.3 x86-64 (ядро linux 3.10.0-514.2.2.el7).

Глобальные массивы в пакете ETSVC были выравнены на границу 32 байта. Эксперименты выполнены для массивов с элементами типов `double`, `float`, `int` и `short int`. На рисунке 6 представлены результаты для данных типа `double`. Для GCC C/C++ общее количество векторизованных циклов составляет 79. При этом 34 из них были векторизованы только им. LLVM/Clang векторизовал 51 цикл, 6 из которых смог векторизовать только он. Количество невекторизованных циклов ни одним из компиляторов составило 66. Результаты векторизации для массивов с элементами типов `float`, `int` и `short int` аналогичны `double` для обоих компиляторов.

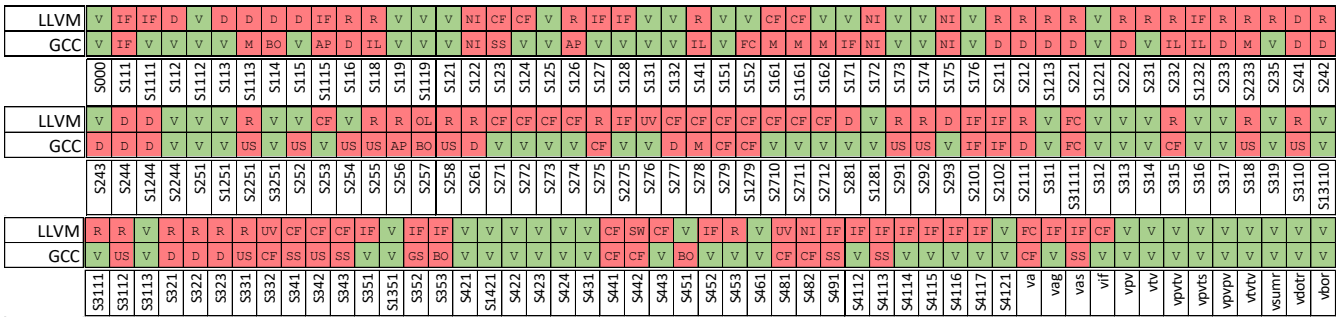


Рисунок 6 – Результаты векторизации циклов (архитектура Intel 64, тип данных `double`)

Таблица 1 содержит расшифровку сокращенных обозначений в результатах векторизации на рисунке 6. Множество трудно векторизуемых циклов, которые не были векторизованы ни одним компилятором, распределено по всем категориям: «Анализ зависимостей по данным» – 14 циклов,

«Анализ потока управления и трансформация циклов» – 29 циклов, «Распознавание идиоматических конструкций» – 15 циклов, «Полнота понимания языка программирования» – 6 циклов.

Таблица 1 – Сокращенные обозначения результатов векторизации

V	Цикл векторизован полностью
IF	Векторизация возможна, но не эффективна
D	Зависимость по данным препятствует векторизации
M	Сгенерировано несколько версий цикла, из которых в процессе выполнения программы будет выбрана невекторизованная (multiversioning)
BO	Неподходящая операция или неподдерживаемая форма границы цикла
AP	Сложный шаблон доступа к элементам массива
R	Значение, которое не может быть идентифицировано как результат редукции, используется вне цикла (наличие индуктивных переменных)
IL	Переменная-счетчик внутреннего цикла не является инвариантом
NI	Невозможно вычислить количество итераций
CF	Невозможно определить направление потока управления
SS	Цикл не подходит для векторной записи по несмежным адресам
FC	Цикл содержит вызовы функций или данные, которые невозможно проанализировать
UV	Векторизатор не может понять поток управления в цикле
SW	Наличие оператора `switch` в цикле
US	Неподдерживаемое использование в выражении
GS	В базовом блоке нет сгруппированных операций записи

Максимальное ускорение, полученное при векторизации компилятором GCC C/C++, составило 4.06, 8.1, 12.01 и 24.48 для типов `double`, `float`, `int` и `short int`, соответственно. Компилятором LLVM/Clang получены следующие значения максимального ускорения: 5.12 (`double`), 10.22 (`float`), 4.55 (`int`) и 14.57 (`short int`). Ускорение измерялось как отношение времени выполнения скалярного кода к времени выполнения векторизованного. При этом учитывались только значения ускорения, большие 1.15. Как показал анализ, значения максимального ускорения соответствуют циклам, выполняющим операции редукции (сумма, произведение, поиск минимального или максимального элемента) над элементами одномерных массивов всех типов данных. Эти циклы относятся в ETSVC к категории «Распознавание идиоматических конструкций».

Инструментарий анализа эффективности использования функциональных устройств вычислительного ядра. Архитектурно-ориентированная оптимизация программ требует использования средств анализа эффективности использования микроархитектурных возможностей для ускорения вычислений, собирающих количественную информацию о ходе выполнения всей программы или определенного ее участка.

Современные процессоры реализуют технологии, позволяющие выпол-

нять глубокий анализ эффективности кода программы. В частности, в архитектуру процессоров фирмы Intel внедрены специальные счетчики мониторинга производительности, реализована технология Intel Processor Trace.

В процессе профилирования и анализа эффективности программ очень важно уменьшить влияние ОС на выполнение кода. Полностью исключить активность ОС невозможно, так как неизбежно происходят прерывания и исключения, обработка системных вызовов, работа планировщика процессов и многое другое. Активность ОС, которую принято называть *шумом ОС*, затрудняет анализ эффективности кода программ.

В диссертации предложен инструментарий профилирования программ позволяющий получить значения счетчиков производительности при выполнении заданного участка для конкретной микроархитектуры. Реализована поддержка микроархитектур Ivy Bridge, Haswell и Broadwell.

Четвертая глава посвящена развитию мультикластерной ВС с многоуровневым параллелизмом, в конфигурировании которой автор принимал активное участие. Выполнено описание стека программного и аппаратного обеспечения подсистем ВС. Стек программного обеспечения, содержащий необходимые средства для использования параллелизма каждого уровня, расширен разработанным инструментарием оптимизации и профилирования выполнения параллельных программ.

ЗАКЛЮЧЕНИЕ

В диссертации предложены архитектурно-ориентированные методы и алгоритмы организации функционирования ВС и оптимизации выполнения параллельных программ на них.

1. Для ВС с многоуровневым параллелизмом разработаны средства оптимизации циклического доступа к информационным массивам в параллельных программах на базе модели разделенного глобального адресного пространства.

1.1. Предложен алгоритм *Array Preload* трансформации конструкций циклической передачи потока управления подчиненным элементарным машинам, сокращающий время выполнения программ за счет опережающего копирования информационных массивов. В отличие от известных, разработанный алгоритм применим к PGAS-программам, в которых на этапе компиляции неизвестно множество используемых элементов массивов.

1.2. В моделях параллельных вычислений LogP, LogGP и Hockney построены оценки эффективности выполнения формируемого алгоритмом *Array Preload* кода, показывающие отсутствие функциональной зависимости времени его выполнения от количества итераций циклов.

1.3. Выполнена реализация алгоритма в виде расширения компилятора языка IBM X10. По сравнению со стандартным алгоритмом предложенный позволяет на кластерных ВС с сетями связи Gigabit Ethernet и InfiniBand

QDR сократить время выполнения циклического доступа к элементам массивов в 1,2–2,5 раз.

2. Для многопроцессорных вычислительных узлов с общей памятью разработаны и исследованы методы оптимизации выполнения многопоточных программ.

2.1. Предложен метод сокращения числа ложных конфликтов в многопоточных программах на базе программной транзакционной памяти. В основе метода лежит подбор (суб)оптимальных параметров таблиц обнаружения конфликтов в реализации транзакционной памяти по результатам предварительного профилирования целевой программы.

2.2. Выполнена программная реализация метода сокращения числа ложных конфликтов в расширении компилятора GCC. Использование предложенного метода позволяет сократить время выполнения параллельных программ в среднем на 20%, что экспериментально показано на тестовых программах из пакета STAMP.

2.3. Для известных алгоритмов автоматической векторизации циклов в открытых компиляторах GCC и LLVM/Clang выявлены классы трудно векторизуемых циклов из тестового набора ETSVC. Установлено, что на архитектуре Intel 64 известные алгоритмы способны векторизовать от 34% до 52% циклов пакета ETSVC. Построенное подмножество циклов составляет базисный набор для анализа эффективности ядер автовекторизаторов оптимизирующих компиляторов для векторных процессоров класса «регистр-регистр».

2.4. Создан инструментарий анализа эффективности использования микроархитектурных возможностей ядер суперскалярных процессоров ВС. В отличие от известных пакетов, предложенные программные средства позволяют анализировать загрузку суперскалярного конвейера архитектуры Intel 64 потоком инструкций с точностью до нескольких команд ассемблера.

3. Выполнено развитие программно-аппаратной конфигурации мультикластерной ВС. В состав системы введены сегменты на базе вычислительных узлов с сопроцессорами Intel Xeon Phi и графическими процессорами NVIDIA. Программный инструментарий системы расширен разработанными автором пакетами оптимизации использования многоуровневого параллелизма ВС в параллельных программах.

Рекомендации и перспективы дальнейшей разработки темы.

1. Разработка методов организации отказоустойчивого (живучего) мультипрограммного функционирования большемасштабных мультиархитектурных ВС при выполнении PGAS-программ.

2. Развитие оценочных моделей для расстановки приоритетов (полу)автоматического совместного использования различных форм параллелизма в программах: передача сообщений, многопоточность, векторизация кода.

ОПУБЛИКОВАННЫЕ РАБОТЫ АВТОРА ПО ТЕМЕ ДИССЕРТАЦИИ

Статьи в журналах из перечня ВАК РФ

1. **Кулагин, И.И.** О спекулятивном выполнении критических секций на вычислительных системах с общей памятью / И.И. Кулагин, М.Г. Курносов // Известия Южного федерального университета. Технические науки. – 2016. – № 11. – С. 54-64.

2. **Кулагин, И.И.** Оптимизация обнаружения конфликтов в параллельных программах с транзакционной памятью / И.И. Кулагин, М.Г. Курносов // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. – 2016. – Т. 5, № 4. – С. 46-60.

3. **Кулагин, И.И.** Инструментация и оптимизация выполнения транзакционных секций многопоточных программ / И.И. Кулагин, М.Г. Курносов // Труды Института системного программирования РАН. – 2015. – Том 27. Выпуск 6. – С. 135-150.

4. **Кулагин, И.И.** Эвристические алгоритмы оптимизации информационных обменов в параллельных PGAS-программах / И.И. Кулагин, А.А. Пазников, М.Г. Курносов // Вестник СибГУТИ. – № 3. – 2014. – С. 52-66.

Публикации в изданиях, индексируемых Scopus и Web of Science

5. **Kulagin, I.** Optimization of Conflict Detection in Parallel Programs with Transactional Memory / I. Kulagin, M. Kurnosov // Proc. of 10th Annual International Scientific Conference on Parallel Computing Technologies (PCT-2016). – CEUR-WS, 2016 – Vol. 1576. – P. 582-594.

6. **Kulagin, I.**. Heuristic Algorithms for Optimizing Communications in Parallel PGAS-programs / I. Kulagin, A. Paznikov, M. Kurnosov // Proc. of the 13th International Conference on Parallel Computing Technologies (PaCT-2015). – Springer LNCS, 2015. – Vol. 9251. – P. 405-409.

Свидетельства о государственной регистрации программ для ЭВМ

7. Свидетельство 2017660065 РФ. Программа детального анализа производительности выполнения кода на архитектуре Intel 64 : свидетельство об официальной регистрации программы для ЭВМ / **И.И. Кулагин**, М.Г. Курносов; заявитель и патентообладатель СибГУТИ; заявл. 17.07.2017, опубл. 14.09.2017.

8. Свидетельство 2016660098 РФ. Программа оптимизации выполнения транзакционных секций в параллельных программах для вычислительных систем с общей памятью : свидетельство об официальной регистрации

программы для ЭВМ / **И.И. Кулагин**, М.Г. Курносов; заявитель и патентообладатель СибГУТИ; заявл. 25.07.2016, опубл. 06.09.2016.

9. Свидетельство 2015619554 РФ. Программа компиляторной оптимизации доступа к удаленным массивам в программах на языке IBM X10 : свидетельство о государственной регистрации программы для ЭВМ / **И.И. Кулагин**, М.Г. Курносов; заявитель и патентообладатель СибГУТИ; заявл. 14.07.2015, опубл. 08.09.2015.

Публикации в сборниках трудов и материалах конференций

10. Молдованова, О.В. Векторизация циклов в открытых компиляторах для архитектур с короткими векторными регистрами / О.В. Молдованова, **И.И. Кулагин**, М.Г. Курносов // Сборник трудов тринадцатой международной Азиатской школа-семинар «Проблемы оптимизации сложных систем» в рамках международной мультиконференции IEEE SIBIRCON-2017. – 2017. – С. 70.

11. **Кулагин, И.И.** Подход к архитектурно-ориентированной оптимизации программ для архитектуры Intel 64 // Материалы Российской научно-технической конференции «Обработка информации и математическое моделирование», 2017. – Новосибирск: СибГУТИ, 2017. – С. 300-310.

12. **Кулагин, И.И**. Оптимизация обнаружения конфликтов в параллельных программах с транзакционной памятью / И.И. Кулагин, М.Г. Курносов // Параллельные вычислительные технологии (ПаВТ'2016): труды международной научной конференции (28 марта – 1 апреля 2016 г., г. Архангельск). – Челябинск: Издательский центр ЮУрГУ, 2016. – С. 582-594.

13. **Кулагин, И.И.** О подходах к реализации программной транзакционной памяти / И.И. Кулагин, М.Г. Курносов // Материалы Российской научно-технической конференции «Обработка информации и математическое моделирование», 2016. – Новосибирск: СибГУТИ, 2016. – С. 331-338.

14. **Кулагин, И.И.** О спекулятивном выполнении критических секций на вычислительных системах с общей памятью / И.И. Кулагин, М.Г. Курносов // Материалы Всероссийской научно-технической конференции «Суперкомпьютерные технологии» (СКТ-2016), 2016. – Т. 1. – С. 170-174.

15. **Кулагин, И.И.** Подход к сокращению ложных конфликтов в параллельных программах на базе транзакционной памяти / И.И. Кулагин, М.Г. Курносов // Тезисы докладов Сибирской конференции по параллельным и высокопроизводительным вычислениям (СКПВВ-2015), Томск, 2015. – С. 48.

16. **Кулагин, И.И.** Анализ обнаружения ложных конфликтов в приложениях с программной транзакционной памятью / И.И. Кулагин, М.Г. Курносов // Материалы Российской научно-технической конференции «Обработка информации и математическое моделирование», 2015. – С. 335-337.

17. **Кулагин, И.И.** Повышение эффективности циклического доступа к удаленным массивам в программах на языке IBM X10 // Сборник статей Всероссийской научно-практической конференции: Многоядерные процессоры, параллельное программирование, ПЛИС, системы обработки сигналов. – Изд-во Алт. ун-та, 2015. – С. 129-136.

18. **Кулагин, И.И.** Алгоритмы оптимизации ложных конфликтов в параллельных программах на базе транзакционной памяти / И.И. Кулагин, М.Г. Курносов // Материалы международной конференции «Актуальные проблемы вычислительной и прикладной математики» (АПВПМ-2015, AMCA-2015), Новосибирск, 2015. – С. 416-422.

19. **Кулагин, И.И.** Оптимизация доступа к удаленным массивам в программах на языке IBM X10 // Материалы 52-й Международной научной студенческой конференции МНСК-2014: Информационные технологии / Новосиб. гос. ун-т. – Новосибирск, 2014. С. 168.

20. **Кулагин, И.И.** Оптимизация информационных обменов в параллельных PGAS-программах / И.И. Кулагин, А.А. Пазников, М.Г. Курносов // Материалы 3-й Всероссийской научно-технической конференции «Суперкомпьютерные технологии» (СКТ-2014), 2014. – Т.1 – С. 158-162.

21. **Кулагин, И.И.** Методы оптимизации передачи массивов в параллельных программах на языке IBM X10 / И.И. Кулагин, М.Г. Курносов // Материалы докладов 10-ой Российской конференции с международным участием «Новые информационные технологии в исследование сложных структур» (ICAM-2014). – Томск: НТЛ, 2014. – С. 5-6.

22. **Кулагин, И.И.** Исследование эффективности доступа к распределенным массивам в программах на языке IBM X10 / И.И. Кулагин, М.Г. Курносов // Материалы Российской научно-технической конференции «Обработка информационных сигналов и математическое моделирование». – Новосибирск, 2014. – С. 77-79.

23. **Кулагин, И.И.** Оптимизация выполнения MPI-программ по результатам их профилирования / М.Г. Курносов, И.И. Кулагин // Материалы Российской научно-технической конференции «Обработка информационных сигналов и математическое моделирование». Новосибирск, 2012. – С. 159-160.

Автореферат

Кулагин Иван Иванович

СРЕДСТВА АРХИТЕКТУРНО-ОРИЕНТИРОВАННОЙ ОПТИМИЗАЦИИ ВЫПОЛНЕНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ ДЛЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ С МНОГОУРОВНЕВЫМ ПАРАЛЛЕЛИЗМОМ

Автореферат диссертации на соискание ученой степени кандидата технических наук

Подписано в печать «27» ноября 2017 г.
Бумага офсетная. Печать цифровая.
Усл. печ. л. 1,38.
Тираж 150. Заказ 15.

Отпечатано в ООО «Параллель».
630090, Новосибирск, ул. Институтская, 4/1. Тел. (383) 330-26-98.