ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОУ ВПО “СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ”

На правах рукописи

Курносов Михаил Георгиевич

МОДЕЛИ И АЛГОРИТМЫ ВЛОЖЕНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ В РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Специальность: 05.13.15 – Вычислительные машины и системы

Д И С С Е Р Т А Ц И Я

на соискание

ученой степени кандидата технических наук

Научный руководитель –

доктор технических наук

профессор

член-корреспондент РАН

В. Г. Хорошевский

Новосибирск – 2008

СОДЕРЖАНИЕ

СПИСОК ОСНОВНЫХ СОКРАЩЕНИЙ

АЛУ – арифметико-логическое устройство.

БИС – большая интегральная схема.

ВМ – вычислительный модуль.

ВС – вычислительная система.

ГОУ ВПО “СибГУТИ” – Государственное образовательное учреждение высшего профессионального образования “Сибирский государственный университет телекоммуникаций и информатики”.

ИФП СО РАН – Учреждение Российской академии наук Институт физики полупроводников им. А. В. Ржанова Сибирского отделения РАН.

ЛК – локальный коммутатор.

ЛП – локальная память.

ОС – операционная система.

ПК – персональный компьютер.

ПО – программное обеспечение.

СО АН СССР – Сибирское отделение Академии наук Союза Советских Социалистических Республик.

СУ – системное устройство.

ЦПВТ – Центр параллельных вычислительных технологий.

ЭВМ – электронная вычислительная машина.

ЭМ – элементарная машина.

ЭП – элементарный процессор.

MPI – Message Passing Interface.

ВВЕДЕНИЕ

Актуальность работы. Возрастающая потребность в решении сложных задач науки и техники привела к созданию распределенных вычислительных систем (ВС) [27, 95]. В архитектурном плане распределенная ВС представляется множеством взаимодействующих элементарных машин, оснащенных средствами коммуникаций и внешними устройствами. Элементарная машина (ЭМ) – это основной функциональный и структурный элемент ВС; конфигурация ЭМ допускает варьирование в широких пределах – от процессорного ядра до ЭВМ. Все основные ресурсы распределенных ВС (арифметико-логические устройства, память, средства управления и коммуникаций) являются логически и технически рассредоточенными. Число ЭМ в распределённых ВС допускает варьирование от нескольких единиц до сотен тысяч (например, в МВС-15000ВМ число процессоров равно 1148, в IBM Roadrunner – 122400, а в системах IBM BlueGene второго поколения до 884736).

Современные распределенные ВС являются мультиархитектурными [92, 94, 95]. В зависимости от уровня рассмотрения их функциональных структур, они могут выглядеть и как MISD, и как SIMD, и как MIMD системы. Для таких систем характерны иерархическая организация и различные пропускные способности каналов связи между их ресурсами (вычислительными узлами, ЭМ, процессорами и их ядрами).

Время выполнения параллельных программ на распределенных ВС существенно зависит от того насколько они эффективно вложены в систему [40-45, 90, 93, 106-107, 116, 133, 144, 164]. Под эффективным вложением понимается такое распределение ветвей параллельной программы между ЭМ системы, при котором достигаются минимумы накладных расходов на межмашинные обмены информацией и дисбаланса загрузки ЭМ.

При организации эффективного функционирования распределенных ВС актуальной является задача разработки моделей и алгоритмов вложения па-

раллельных программ, учитывающих архитектурные особенности современных систем.

Исследования в области распредёленных ВС ведутся с середины XX столетия. В нашей стране и за рубежом выполнен ряд фундаментальных работ, посвящённых проблемам организации высокопроизводительных вычислительных средств: проведены исследования по теории функционирования и построению оптимальных (макро)структур ВС, проработаны многие аспекты создания программного обеспечения, исследован широкий круг задач, допускающих эффективную реализацию на распределённых ВС. Построены отечественные вычислительные системы: “Минск-222”, СУММА, МИНИМАКС, МИКРОС, МВС и др.

Фундаментальный вклад в теорию и практику вычислительных систем и параллельных вычислительных технологий внесли выдающиеся учёные, среди которых: Е. П. Балашов, В. Б. Бетелин, В. С. Бурцев, В. В. Васильев, В. В. Воеводин, В. М. Глушков, В. Ф. Евдокимов, Э. В. Евреинов, А. В. Забродин, В. П. Иванников, М. Б. Игнатьев, А. В. Каляев, И. А. Каляев, Л. Н. Королев, В. Г. Лазарев, С. А. Лебедев, В. К. Левин, Г. И. Марчук, В. А. Мельников, Ю. И. Митропольский, Д. А. Поспелов, И. В. Прангишвили, Д. В. Пузанков, Г. Е. Пухов, А. Д. Рычков, Г. Г. Рябов, А. А. Самарский, В. Б. Смолов, А. Н. Томилин, Я. А. Хетагуров, В. Г. Хорошевский, Б. Н. Четверушкин, Ю. И. Шокин, Н. Н. Яненко, S. Cray, M. Flynn, I. Foster, A. Gara, D. Grice, D. Hillis, C. Kesselman, D. L. Slotnick и другие.

При решении проблем оптимизации вложения параллельных программ в распределенные ВС большую роль сыграли фундаментальные работы по исследованию операций и оптимальному управлению выдающихся ученых: В. Л. Береснева, Э. Х. Гимади, В. Т. Дементьева, С. В. Емельянова, Ю. И. Журавлева, А. А. Корбут, С. К. Коровина, Ю. С. Попкова, К. В. Рудакова, D. P. Agrawal, R. Baraglia, S. H. Bokhari, P. Bouvry, A. Gara,

G. Karypis, B. W. Kernighan, V. Kumar, S. Lin, C. H. Papadimitriou, R. Perego, K. Steiglitz и др.

В диссертации предложены модели и алгоритмы, позволяющие осуществлять субоптимальное вложение в распределенные ВС параллельных программ с целью минимизации времени их выполнения. На основе полученных результатов созданы программные средства оптимизации вложения параллельных MPI-программ в распределенные ВС.

Цель работы и задачи исследования. Целью диссертационной работы является разработка и исследование моделей и алгоритмов оптимизации вложения параллельных программ в распределенные вычислительные системы.

В соответствии с целью определены нижеследующие задачи исследования.

1. Анализ архитектурных особенностей современных ВС и методов вложения в них параллельных программ.

2. Разработка методов и алгоритмов вложения параллельных программ в ВС с иерархической организацией коммуникационных сред.

3. Создание методов и алгоритмов вложения параллельных программ в пространственно-распределенные ВС.

4. Разработка программного инструментария оптимизации вложения параллельных MPI-программ в ВС на базе многоядерных процессоров.

5. Формирование средств анализа производительности параллельных MPI-программ.

Методы исследования. Для достижения поставленной цели и решения сформулированных в диссертационной работе задач использовались методы теории вычислительных систем, исследования операций, теории графов и теории алгоритмов. Экспериментальные исследования осуществлялись с помощью параллельного моделирования на пространственно-распределенной мультикластерной ВС.

Научная новизна работы. Разработаны нетрудоемкие средства субоптимального вложения параллельных программ в распределенные вычислительные системы:

1. Предложены математическая модель коммуникационной среды ВС с иерархической организацией и алгоритмы вложения параллельных программ в такие системы.

2. Разработаны эвристические алгоритмы формирования в пределах ВС подсистем, обеспечивающих эффективную реализацию основных схем межмашинных обменов, и вложения в них параллельных программ.

3. Создан стохастический алгоритм вложения параллельных программ в пространственно-распределенные ВС; предложена его параллельная версия для большемасштабных систем.

4. Реализованы алгоритм формирования в пространственно-распределенных ВС гомогенных подсистем и вложения в них параллельных программ.

5. Осуществлено параллельное моделирование разработанных алгоритмов на мультикластерной ВС Центра параллельных вычислительных технологий ГОУ ВПО “Сибирский государственный университет телекоммуникаций и информатики” (ЦПВТ ГОУ ВПО “СибГУТИ”) и Лаборатории вычислительных систем Института физики полупроводников им. А. В. Ржанова СО РАН (ИФП СО РАН).

Практическая ценность работы. Разработанные в диссертации модели и алгоритмы в композиции с известными средствами управления ресурсами распределенных ВС составляют базу для организации их эффективного функционирования.

Созданные алгоритмы легли в основу пакета оптимизации вложения параллельных MPI-программ в кластерные ВС. Применение реализованных средств позволяет сократить время выполнения параллельных программ.

Разработан инструментарий анализа производительности MPI-программ, который позволяет получать информацию о структуре информационных обменов между ветвями параллельных программ и использовать ее при оптимизации их вложения.

Программные средства внедрены в действующую пространственно-распределенную мультикластерную вычислительную систему ЦПВТ ГОУ ВПО “СибГУТИ” и Лаборатории вычислительных систем ИФП СО РАН.

Реализация и внедрение результатов работы. Основные результаты диссертационной работы нашли применение в работах по созданию и развитию пространственно-распределенной мультикластерной вычислительной системы ЦПВТ ГОУ ВПО “СибГУТИ” и Лаборатории вычислительных систем ИФП СО РАН.

Диссертационные исследования выполнялись в рамках проекта № 4.6.1.1 “Методы и алгоритмы анализа и организации функционирования распределенных вычислительных систем, параллельное мультипрограммирование и аппаратурно-программный инструментарий для моделирования технологий обработки информации” (программа 4.6.1 фундаментальных исследований СО РАН). Работа поддержана грантами Российского фонда фундаментальных исследований № 08-07-00018 (научный руководитель – Курносов М.Г.), 08-07-00022, 06-07-89089, 05-07-90009, грантами Президента РФ по поддержке ведущих научных школ № НШ-9505.2006.9, НШ-2121.2008.9, стипендиальными грантами компаний Intel и Alcatel, а также грантом по Программе “У.М.Н.И.К.” Фонда содействия развитию малых форм предприятий в научно-технической сфере.

Результаты диссертации внедрены в учебный процесс. Они использовались при чтении курсов лекций на Кафедре вычислительных систем ГОУ ВПО “СибГУТИ” по дисциплинам “Теория функционирования распределенных вычислительных систем” и “Высокопроизводительные вычислительные системы”.

Внедрение результатов диссертационных исследований подтверждено соответствующими актами.

Достоверность полученных результатов подтверждается проведенными экспериментами и моделированием, согласованностью с данными имеющимися в отечественной и зарубежной литературе, а также экспертизами работы, прошедшими при получении грантов.

Апробация работы. Основные результаты работы докладывались и обсуждались на Международных, Всероссийских и региональных научных конференциях, в том числе:

– Международной научной конференции “Моделирование-2008” (“Simulation-2008”, г. Киев, Украина, 2008 г.).

– Международной научной конференции “Информационные технологии и математическое моделирование систем” (г. Майорка, Испания, 2008 г.).

– Всероссийской конференции с международным участием “Новые информационные технологии в исследовании сложных структур” (“ICAM-2008”, г. Томск, 2008 г.).

– Международной научно студенческой конференции “Студент и научно-технический прогресс (г. Новосибирск, 2006, 2007, 2008 гг.).

– Всероссийской научно-технической конференции “Информатика и проблемы телекоммуникаций” (г. Новосибирск, 2006, 2007, 2008 гг.).

– Всероссийской научной конференции “Наука. Технологии. Инновации” (г. Новосибирск, 2007 г.).

– Сибирской школе-семинаре по параллельным и высокопроизводительным вычислениям (г. Томск, 2007 г.).

Публикации. По теме диссертации опубликована 31 работа, включая 2 статьи в рецензируемых изданиях. Результаты исследований отражены в отчетах по грантам и НИР.

Личный вклад. Все основные результаты диссертационной работы получены автором лично.

Основные положения диссертации, выносимые на защиту.

1. Метод и алгоритмы вложения параллельных программ в ВС с иерархической организацией коммуникационных сред.

2. Эвристические алгоритмы формирования в пределах ВС подсистем, обеспечивающих эффективную реализацию основных схем межмашинных обменов, и вложения в них параллельных программ.

3. Последовательный и параллельный стохастические алгоритмы вложения параллельных программ в пространственно-распределенные ВС.

4. Алгоритмы формирования в пространственно-распределенных ВС однородных подсистем и вложения в них параллельных программ.

5. Программный инструментарий оптимизации вложения параллельных MPI-программ в ВС на базе многоядерных процессоров.

6. Средства анализа производительности параллельных MPI-программ.

7. Функциональная структура пространственно-распределенной мультикластерной ВС, оснащенная средствами оптимизации вложения параллельных программ.

Структура и объем диссертации. Диссертационная работа состоит из введения, четырех глав, заключения и списка используемых литературных источников, изложенных на 170 страницах, а также приложений на 7 страницах.

Содержание работы.

В первой главе описывается архитектура распределенных ВС с программируемой структурой, приводятся основные понятия параллельного программирования, дается описание основных принципов построения и архитектурных особенностей современных кластерных, мультикластерных ВС и GRID систем. Рассмотрена задача вложения параллельных программ в распределенные ВС и основные режимы их функционирования, произведен анализ известных методов и алгоритмов вложения параллельных программ.

Во второй главе предложена математическая модель коммуникационной среды ВС с иерархической организацией, сформулирована задача оптимального вложения параллельных программ в такие системы, описаны алгоритмы приближенного решения задачи. Приведены эвристические алгоритмы формирования в пределах ВС подсистем, допускающих эффективную реализацию основных схем межмашинных обменов, и вложения в них параллельных программ.

В третьей главе рассмотрена задача оптимального вложения параллельных программ в пространственно-распределенные ВС, предложены последовательный и параллельный стохастические алгоритмы решения задачи. Описаны алгоритмы формирования однородных подсистем ЭМ и вложения в них параллельных программ.

В четвертой главе описана архитектура пространственно-распределенной мультикластерной вычислительной системы, в разработке которой диссертант принимал непосредственное участие. Приведено описание реализованных на базе предложенных алгоритмов средств анализа производительности и оптимизации вложения параллельных MPI-программ в ВС на базе многоядерных процессоров. Отражены результаты моделирования работы алгоритмов.

В заключении сформулированы основные результаты диссертационной работы.

В приложениях приведены сводные данные о предложенных алгоритмах и описание структурной организации сегментов мультикластернной ВС.

ГЛАВА 1. РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ С ПРОГРАММИРУЕМОЙ СТРУКТУРОЙ

1.1. Понятие о распределенных ВС с программируемой структурой

1.1.1. Модель коллектива вычислителей. Классификация ВС

Существует два основных подхода к созданию вычислительных средств [22-27, 95]:

1) построение электронных вычислительных машин (ЭВМ), моделирующих процесс выполнения алгоритма одиночным человеком-вычислителем;

2) создание вычислительных систем, моделирующих процесс выполнения алгоритма коллективом вычислителей.

Модель вычислителя сформулирована в [25] и наиболее полно по отношению к вычислительным машинам отражена в [27]. Для этой модели характерны последовательное выполнение операций, фиксированная структура, неоднородность связей и функциональных элементов. Использование модели вычислителя для построения высокопроизводительных вычислительных средств ограничено теоретическими и техническими пределами скорости выполнения операций (возможностями элементной базы и фундаментальными физическими законами) [95].

Модель коллектива вычислителей сформулирована в работах [27] и получила дальнейшее развитие в работах [22, 32, 95].

Под коллективом вычислителей понимается совокупность вычислительных машин, программно-аппаратурным способом настраиваемая на решение общей задачи.

В основу коллектива вычислителей положены три основополагающих архитектурных принципа [95]:

1) *параллелизм* (*parallelism*, *concurrency*) при обработке информации;

2) *программируемость* (*programmability*, *adoptability*) *структуры* (настраиваемости структуры сети межмашинных связей между вычислителями, достигаемой программными средствами);

3) *однородность* (*homogeneity*) *конструкции* (однородности вычислителей и структуры).

Как видно, модели коллектива вычислителей опирается на положения, противоположные принципам, лежащим в основе конструкции вычислителя [27, 95]. Принцип программируемости структуры является не менее фундаментальным в области архитектуры средств обработки информации, чем предложение Дж. фон Неймана по хранению программы работы ЭВМ в ее памяти и модификации программы с помощью самой же машины. Требования принципа программируемости структуры сводятся к тому, чтобы в коллективе вычислителей была заложена возможность хранения описания изначальной физической структуры, априорной автоматической (программной) настройки проблемно-ориентированных (виртуальных) конфигураций и их перенастройки в процессе функционирования с целью обеспечения адекватности структурам и параметрам решаемых задач и достижения эффективности при заданных условиях эксплуатации.

Конструктивная однородность модели коллектива вычислителей заключается в формировании его из совокупности одинаковых вычислителей, регулярно соединенных между собой.

В отличие от одиночного вычислителя, коллектив машин-вычислителей обладает теоретически неограниченной производительностью, обусловленной отсутствием ограничений на увеличение их числа. Кроме того, по сравнению с ЭВМ, построенное на модели коллектива вычислительное средство способно обладать заданной надежностью и живучестью [21, 27, 93-95, 98], т.е. возможностью функционировать при отказах элементов, а также допускает простой способ наращивания производительности.

Вычислительное средство, базирующееся на модели коллектива вычислителей, называется *вычислительной системой* [95].

При достаточно общей трактовке под ВС понимается совокупность взаимосвязанных и одновременно функционирующих аппаратурно-программных вычислителей, которая способна не только реализовать (параллельный) процесс решения сложной задачи, но и в процессе работы автоматически настраиваться и перестраиваться с целью достижения адекватности между своей структурно-функциональной организацией и структурой и характеристиками решаемой задачи.

Классификация вычислительных систем. Существуют различные подходы к классификации архитектур вычислительных средств [2-5, 10-11, 13-14, 16-17]. Наибольшее распространение получила классификация, предложенная в 1966 году М. Дж. Флинном (M. J. Flynn) [124-125], в которой выделяют четыре класса архитектур вычислительных средств: SISD (Single Instruction stream / Single Data stream), SIMD (Single Instruction stream / Multiple Data stream), MISD (Multiple Instruction stream / Single Data stream), MIMD (Multiple Instruction stream / Multiple Data stream). В основе классификации лежит разделение архитектур вычислительных средств по количеству обрабатываемых ими потоков команд и данных.

Первый класс архитектуры – SISD или ОКОД (Одиночный поток Команд и Одиночный поток Данных) относится к ЭВМ. Под потоком команд понимается любая их последовательность, поступающая для исполнения вычислительным средством (ЭВМ или процессором, в случае SISD-архитектуры). При выполнении команд потока требуются операнды (данные), следовательно, поток команд “порождает” поток данных.

Архитектуры MISD, SIMD, MIMD относятся к вычислительным системам. В этих архитектурах имеет место “множественность” потоков или (и) команд, или (и) данных. Множественность характеризуется количеством одновременно реализуемых потоков команд или (и) данных.

Архитектура MISD (Multiple Instruction stream / Single Data stream) или МКОД (Множественный поток Команд и Одиночный поток Данных) позволяет нескольким потокам команд обрабатывать один поток данных. Архитектура SIMD (Single Instruction stream / Multiple Data stream) или ОКМД (Одиночный поток Команд и Множественный поток Данных) предоставляет возможность одному потоку команд обрабатывать несколько потоков данных. Архитектура MIMD (Multiple Instruction stream / Multiple Data stream) или МКМД (Множественный поток Команд и Множественный поток Данных) допускает обработку несколькими потоками команд нескольких потоков данных.

В архитектурах классов MISD, SIMD, MIMD допустимо построение нескольких типов вычислительных систем [27, 95], среди которых наибольший интерес представляют: 1) конвейерные ВС, 2) матричные ВС, 3) мультипроцессорные ВС, 4) распределенные ВС, 5) ВС с программируемой структурой, 6) кластерные ВС, 7) пространственно-распределенные мультикластерные ВС

Конвейерные ВС – это системы, архитектура которых является предельным вариантом эволюционного развития последовательной ЭВМ и простейшей версией модели коллектива вычислителей. В основе таких систем лежит конвейерный способ обработки информации, а их функциональная структура представляется в виде “последовательности” связанных элементарных блоков обработки информации [30, 90]. Все блоки работают параллельно, но каждый из них реализует лишь свою операцию над данными одного и того же потока. Конвейерные ВС принадлежат классу MISD-систем.

Матричные ВС основываются на принципе массового параллелизма, в них обеспечивается возможность одновременной реализации большого числа операций на элементарных процессорах (ЭП), “объединенных” в матрицу. Каждый ЭП – композиция из арифметико-логического устройства (АЛУ) и локальной памяти (ЛП); последняя предназначается для хранения части дан-

ных. Поток команд на матрицу ЭП формируется устройством управления (следовательно, оно имеет в своём составе память для хранения программ обработки данных). Такие ВС имеют SIMD-архитектуру в классическом виде.

Мультипроцессорные ВС – обширная группа систем, в которую, в частности, могут быть включены конвейерные и матричные ВС. Однако принято к мультипроцессорным ВС относить системы с MIMD-архитектурой, которые состоят из множества (не связанных друг с другом) процессоров и общей (возможно и секционированной, модульной) памяти; взаимодействие между процессорами и памятью осуществляется через коммутатор (общую шину и т.п.), а между процессорами – через память.

Распределенные ВС – мультипроцессорные ВС с MIMD-архитектурой, в которых нет единого ресурса (общей памяти). Основные компоненты распределенной ВС (такие, как коммутатор, устройство управления, арифметико-логическое устройство или процессор, память) допускают представление в виде композиции из одинаковых элементов (локальных коммутаторов и устройств управления, локальных процессоров и модулей памяти).

Вычислительные системы с программируемой структурой полностью основываются на модели коллектива вычислителей и являются композицией взаимосвязанных элементарных машин. Каждая ЭМ в своем составе обязательно имеет локальный коммутатор (ЛК), процессор и память; может иметь также внешние устройства. Локальная память ЭМ предназначается для хранения и части данных, и, главное, ветви параллельной программы. Архитектура ВС с программируемой структурой относится к типу MIMD. Такие ВС по своим потенциальным архитектурным возможностям не уступают ни одному из перечисленных выше классов систем.

В вычислительных системах с программируемой структурой выделяют пространственно сосредоточенные и распределенные ВС. Характерной особенностью *сосредоточенных ВС* является компактное пространственное раз-

мещение средств обработки и хранения информации, при котором среднее время передачи слова между функциональными модулями (процессорами, модулями памяти, ЭВМ и др.) соизмеримо со средним временем выполнения одной операции в процессоре.

К *пространственно-распределенным ВС* относят макросистемы – системы сложной конфигурации, в которых в качестве функциональных элементов выступают пространственно-рассредоточенные вычислительные средства, основанные на моделях вычислителя и коллектива вычислителей, и сети связи, обеспечивающие взаимный теледоступ между средствами обработки информации.

Пространственно-распределенные ВС, в которых выполняется условие программной совместимости ЭВМ, будем относить к *гомогенным ВС*.

Концепция вычислительных систем с программируемой структурой была сформулирована в Сибирском отделении АН СССР, первая система (“Минск-222”) была построена в 1965 – 1966 гг. [27].

В 90-х годах 20 столетия получают широкое распространение *кластерные вычислительные системы*. В наиболее общей трактовке, *кластерная ВС* или *кластер* – это композиция множества вычислителей, сети связей между ними и программного обеспечения, предназначенная для параллельной обработки информации (реализации параллельных алгоритмов решения сложных задач). При формировании кластерной ВС могут быть использованы как стандартные промышленные компоненты, так и специально созданные средства.

Начало XXI в. ознаменовалось созданием *пространственно-распределенных мультикластерных ВС* как макроколлективов рассредоточенных кластеров, взаимодействующих между собой через локальные и глобальные сети (включая всемирную сеть Internet).

Архитектура современных ВС существенно отличается от изначальных канонов, доминирующее большинство систем являются мультиархитектур-

ными. В зависимости от уровня рассмотрения их функциональных структур, они могут выглядеть и как MISD, и как SIMD, и как MIMD.

1.1.2. Архитектурные особенности ВС с программируемой структурой

Как было отмечено в п. 1.1.1, *вычислительные системы с программируемой структурой* – это распределенные средства обработки информации. Тип архитектуры ВС – MIMD; в системах заложена возможность программной перенастройки архитектуры MIMD в архитектуры MISD или SIMD. Основная функционально-структурная единица вычислительных ресурсов в системах рассматриваемого класса – это *элементарная машина*, которая является композицией из вычислительного модуля и системного устройства. *Вычислительный модуль* (ВМ) служит как для переработки и хранения информации, так и для выполнения функций по управлению системой в целом. *Системное устройство* (СУ) – это та аппаратурная часть ЭМ, которая предназначается только для обеспечения взаимодействия данной ЭМ с ближайшими соседними машинами (точнее, с системными устройствами, с которыми имеется непосредственная связь).

Допускается конфигурирование ВС с произвольным числом ЭМ. Следовательно, ВС с программируемой структурой относятся к масштабируемым средствам обработки информации и допускают формирование конфигураций с массовым параллелизмом (Scalable Massively Parallel Architecture Computing Systems).

Взаимодействие между ЭМ осуществляется через программно настраиваемую сеть связи. Структура ВС описывается графом $G = (C, E)$, множеству вершин C которого сопоставлены элементарные машины (или системные устройства, или локальные коммутаторы), а множеству рёбер E – линии межмашинных связей.

Требования, предъявляемые к структуре ВС. К структурам современных ВС предъявляется ряд требований [95].

1) Простота вложения параллельного алгоритма решения сложной задачи в структуру ВС. Структура ВС должна быть адекватна достаточно широкому классу решаемых задач; настройка проблемно-ориентированных виртуальных конфигураций не должна быть связана со значительными накладными расходами.

2) Удобство адресации элементарных машин и “переноса” подсистем в пределах вычислительной системы. Вычислительная система должна предоставлять возможность пользователям создавать параллельные программы с виртуальными адресами ЭМ. Следовательно, структура ВС должна позволять реализовать простейший “механизм” преобразования виртуальных адресов ЭМ в реальные (физические) адреса машин системы. Необходимость организации одновременного решения нескольких задач на ВС (т.е необходимость разделения пространства элементарных машин между задачами) обосновывает требование простоты перемещения подсистем в пределах системы (при сохранении их топологических свойств).

3) Осуществимость принципа близкодействия и минимума задержек при межмашинных передачах информации в ВС. Принцип близкодействия предопределяет реализацию обменов информацией между “удалёнными” друг от друга ЭМ через промежуточные машины системы. Следовательно, в условиях ограниченности числа связей у каждой ЭМ структура должна обеспечивать минимум задержек при “транзитных” передачах информации.

4) Масштабируемость и большемасштабность структуры ВС. Для формирования конфигураций ВС с заданной эффективностью требуется, чтобы структура обладала способностью к наращиванию и сокращению числа вершин (машин). Изменение числа ЭМ в ВС не должно приводить к коренным перекоммутациям между машинами и (или) к необходимости изменения числа связей для любых ЭМ.

Для достижения высокой производительности ВС при существующих возможностях микропроцессорной техники требуется число ЭМ порядка $10-10^6$. Для поддержки большемасштабности (массового параллелизма) необходимо, чтобы структура ВС обладала способностью эффективно осуществлять межмашинные обмены информацией в условиях невозможности реализации связей по полному графу (например, из-за ограниченности числа выводов с корпусов БИС).

5) Коммутируемость структуры ВС. Вычислительная система должна быть приспособлена к реализации групповых межмашинных обменов информацией. Следовательно, структура ВС должна обладать способностью осуществлять заданное число одновременных непересекающихся взаимодействий между элементарными машинами.

6) Живучесть структуры ВС. Важным требованием к ВС в целом является обеспечение работоспособности при отказе её компонентов или даже подсистем. Основой функциональной целостности ВС как коллектива элементарных машин является живучесть структуры. Под последним понимается способность структуры ВС обеспечить связность требуемого числа работоспособных ЭМ в системе при ненадёжных линиях межмашинных связей.

7) Технологичность структур ВС. Структура сети межмашинных связей ВС не должна предъявлять особых требований к элементной базе, к технологии изготовления микропроцессорных БИС. Системы должны быть восприимчивы к массовой технологии, их “вычислительное ядро” должно формироваться из массовых микропроцессорных БИС. Последнее позволит достичь приемлемых значений технико-экономических показателей ВС.

Структурные характеристики ВС. Структурные задержки при передачах информации между машинами ВС определяются расстоянием (в смысле теории графов) между вершинами структуры, сопоставленными взаимодействующим машинам [27, 95]. Для оценки структурных задержек в вычислительных системах используются диаметр d и средний диаметр $\bar{d}$ структуры.

Диаметр есть максимальное расстояние, определённое на множестве кратчайших путей между парами вершин структуры ВС:

$$d = \max_{i,j}\{d_{ij}\}, \tag{1.1}$$

а средний диаметр –

$$\overline{d} = (N-1)^{-1}\sum_{l=1}^{d} l \cdot n_l , \tag{1.2}$$

где d_{ij} – расстояние, т.е. минимальное число рёбер, образующих путь из вершины i в вершину j; $i, j \in \{0,1,..., N-1\}$; n_l – число вершин, находящихся на расстоянии l от любой выделенной вершины (однородного) графа G.

Показателем оценивающим структурную коммутируемость ВС, является вектор-функция

$$\mathcal{K}(G, s, s') = \{\mathcal{K}_h(G, s, s')\}, \quad h \in \{1,2,..., [N/2]\}, \tag{1.3}$$

в которой координата $\mathcal{K}_h(G, s, s')$ есть вероятность реализации в системе при заданных структуре G и коэффициентах готовности s и s', соответственно, одной ЭМ и линии связи h одновременных непересекающихся межмашинных взаимодействий (обменов информацией между ЭМ); $[x]$ – целая часть числа x.

Структурная живучесть ВС оценивается вектор-функцией

$$\mathcal{L}(G, s, s') = \{\mathcal{L}_r(G, s, s')\}, \quad r \in E_2^N = \{2,3,..., N\}, \tag{1.4}$$

$\mathcal{L}_r(G, s, s')$ является вероятностью существования подсистемы ранга r (т.е. подмножества из r работоспособных ЭМ, связность которых устанавливается через работоспособные линии связи) при заданных структуре G, коэффициентах готовности s и s' элементарной машины и линии связи, соответственно.

Для оценки производительности каналов связи между ЭМ системы используют [17, 83, 91, 100] показатели: пропускная способность и латентность. *Пропускная способность канала связи* (*bandwidth*, *throughput*, *channel capacity*) – наибольшее количество информации, передаваемой по каналу связи в единицу времени. *Латентность* (*latency*) – это задержка при передаче информации между ЭМ системы, вызванная программными и аппаратурными накладными расходами.

Введённые показатели позволяют осуществить с достаточной полнотой анализ структурных возможностей ВС.

Перспективные структуры ВС. Наиболее полно перечисленным выше требованиям удовлетворяют однородные структуры (т.е. описываемые однородными графами) [27, 69, 70, 95]. Такие структуры являются перспективными для формирования масштабируемых и большемасштабных вычислительных систем (в частности, ВС с программируемой структурой).

В компьютерной индустрии получили распространение *n*-мерные структуры вычислительных систем, известные сейчас как циркулянтные (Circulant Sructures). Впервые они были определены и исследованы в Отделе вычислительных систем Института математики СО АН СССР в начале 70-х годов и первоначально назывались D_n-графами [27]. По определению *D_n-граф* или *циркулянтная структура* есть граф G вида: $\{N; \omega_1, \omega_2, ..., \omega_n\}$, в котором:

- N – число вершин или порядок графа;
- вершины помечены целыми числами i по модулю N, следовательно, $i \in \{0,1,..., N-1\}$;
- вершина i соединена ребром (или является смежной) с вершинами $i \pm \omega_1, i \pm \omega_2, ..., i \pm \omega_n \pmod N$;
- $\{\omega_1, \omega_2, ..., \omega_n\}$ – множество целых чисел, называемых образующими, таких, что $0 < \omega_1 < \omega_2 < ... < \omega_n < (N+1)/2$, а для чисел $N; \omega_1, \omega_2, ..., \omega_n$ наибольшим общим делителем является 1;

– n – размерность графа;

– $2n$ – степень вершины в графе.

В качестве примера рассмотрим D_2-граф (рис. 1.1) или двумерный циркулянт вида: $\{12;3,4\}$.

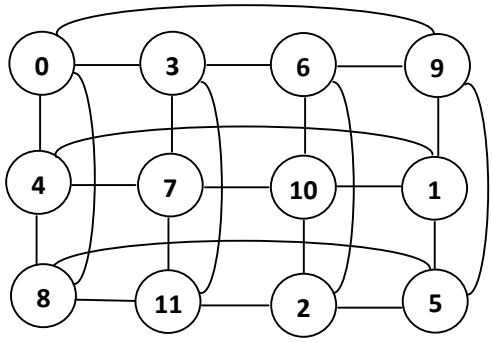


Рис. 1.1. D_2-граф: $\{12;3,4\}$

Графы G вида $\{N;1,\omega_2,...,\omega_n\}$, т.е. D_n-графы или циркулянты с единичной образующей (Loop Networks – петлевые структуры) интенсивно изучаются в последнее время. Циркулянтные структуры $\{N;1,\omega_2\}$ широко внедрены в практику вычислительных систем.

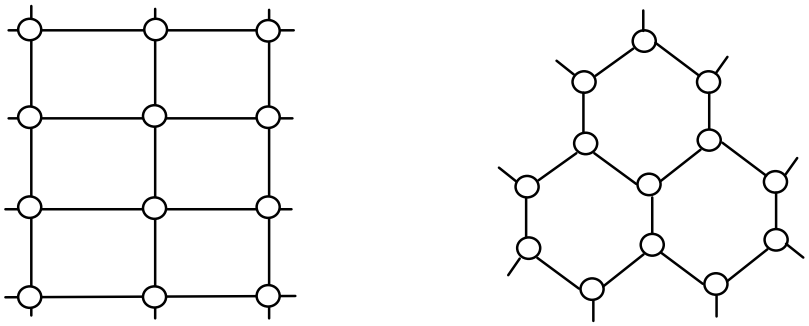
Целые числа $i \in \{0,1,2,...,N-1\}$, отмечающие вершины D_n-графа, называют адресами. Адресация вершин в таких структурах называется диофантовой (в честь древнегреческого математика из Александрии Диофанта, Diophantos, 3 век). В циркулянтных структурах при полном переносе какой-либо подструктуры (всех вершин подструктуры на одно и то же расстояние в одном из направлений) сохраняются все её свойства и адресация вершин. Следовательно, при диофантовой адресации элементарных машин ВС можно простыми средствами реконфигурации осуществить виртуальную адресацию вершин-машин и, следовательно, создавать отказоустойчивые параллельные программы, реализовывать мультипрограммные режимы обработки информации, исключать отказавшие вершины-машины из подсистем, а значит обеспечить живучесть ВС.

При этом алгоритм работы реконфигуратора структуры ВС сводится к изменению адресов α у всех машин подсистемы по формуле:

$$\alpha := [\alpha + (j - i)] \bmod N, \quad \alpha \in \{0,1,..., N-1\}, \tag{1.5}$$

где i – номер ЭМ, исключаемой из подсистемы, а j – номер машины, включаемый в подсистему, $i, j \in \{0,1,..., N-1\}$.

В качестве структур ВС, допускающих масштабирование (изменение числа машин) без коренной перекоммутации уже имеющихся межмашинных связей, используются $Л(N, v, g)$-графы [69-70, 95] (введённые также в Отделе вычислительных систем ИМ СО АН СССР). В такие графы вкладываются D_n-графы; *$Л(N, v, g)$-граф* – это неориентированный однородный граф с числом и степенями вершин, соответственно, N и v и значением обхвата g (рис. 1.2).



а *б*

Рис. 1.2. Фрагменты $Л(N, v, g)$-графов:

$a - v = 4,\ g = 4;\ б - v = 3,\ g = 6$

В $Л(N, v, g)$-графах каждая вершина при $v \geq 3$ входит в не менее v кратчайших простых циклов длиной g (длина кратчайшего цикл в графе называется обхватом). При $v = 2$ $Л(N, v, g)$-граф является простым циклом с N вершинами.

1.1.3. Параллельные алгоритмы и программы

Понятие параллельного алгоритма (Parallel Algorithm) относится к фундаментальным в теории вычислительных систем [12, 13, 15-17, 23, 26-27, 28-30, 34, 65, 67-68, 99, 101-102]. *Параллельный алгоритм* – это описание процесса обработки информации, ориентированное на реализацию в коллективе вычислителей [27, 95]. Такой алгоритм, в отличие от последовательного, предусматривает одновременное выполнение множества операций в пределах одного шага вычислений и как последовательный алгоритм сохраняет зависимость последующих этапов от результатов предыдущих.

Параллельный алгоритм решения задачи составляет основу параллельной программы, которая, в свою очередь, влияет на алгоритм функционирования коллектива вычислителей. Запись параллельного алгоритма на языке программирования, доступном коллективу вычислителей, и называют *параллельной программой*, а сам язык – *параллельным*. Параллельные алгоритмы и программы следует разрабатывать для тех задач, которые недоступны для решения на средствах, основанных на модели вычислителя. Эти задачи принято называть сложными или трудоемкими.

Методы и алгоритмы обработки информации, решения задач, как правило, – последовательные. Процесс “приспособления” методов к реализации на коллективе вычислителей или процесс “расщепления” последовательных алгоритмов решения сложных задач называется *распараллеливанием* (*Paralleling*).

Теоретическая и практическая деятельность по созданию параллельных алгоритмов и программ обработки информации называется *параллельным программированием* (*Parallel Programming*).

Качество параллельного алгоритма (или его эффективность) определяется методикой распараллеливания сложных задач. Выделяют два основных подхода к распараллеливанию задач [27, 34, 68]: локальное и глобальное (крупноблочное) распараллеливание. Первый подход ориентирован на рас-

щепление алгоритма решения сложной задачи на предельно простые блоки (операции или операторы) и требует выделения для каждого этапа вычислений максимально возможного количества одновременно выполняемых блоков. Он не приводит к параллельным алгоритмам, эффективно реализуемым коллективом вычислителей. В самом деле, процесс такого распараллеливания весьма трудоемок, а получаемые параллельные алгоритмы характеризуются не только структурной неоднородностью, но и существенно разными объемами операций на различных этапах вычислений. Последнее является серьезным препятствием на пути (автоматизации) распараллеливания и обеспечения эффективной эксплуатации ресурсов коллектива вычислителей. Локальное распараллеливание позволяет оценить предельные возможности коллектива вычислителей при решении сложных задач, получить предельные оценки по распараллеливанию сложных задач.

Второй подход ориентирован на разбиение сложной задачи на крупные блоки–подзадачи, между которыми существует слабая связность. Тогда в алгоритмах, построенных на основе крупноблочного распараллеливания, операции обмена между подзадачами будут составлять незначительную часть по сравнению с общим числом операций в каждой подзадаче. Такие подзадачи называют *ветвями параллельного алгоритма*, а соответствующие им программы – *ветвями параллельной программы*.

Пусть V – количество операций, которые необходимо выполнить при решении задачи на ВС; n – число параллельных ветвей или число вычислителей, на которых решается задача, $n \geq 2$; Тогда задачу, для которой выполняется условие

$$V \geq n \cdot 10^l, \tag{1.6}$$

будем называть *сложной*, или *системной*, или *трудоемкой*, или с большим объемом вычислений (95).

В соотношении (1.6) l – эмпирический коэффициент, $l \geq 1$. Очевидно, что имеет место зависимость l от быстродействия v каналов связей между вычислителями: $l \to 1$ при $v \to v^*$, где $1/v^*$ – время обращения к локальной памяти в вычислителе.

Задачу, которая имеет небольшой объем вычислений и, следовательно, не допускает эффективного распараллеливания, будем называть *простой*. Простая задача требует для своего решения одного вычислителя.

Структура обменов информацией между ветвями параллельной программы характеризуется информационным графом [15, 87, 72-73]. *Информационный граф параллельной программы* (*Task Graph*) – это конечный граф, вершинам которого соответствуют параллельные ветви, а ребрам – обмены информацией между ветвями.

1.2. Кластерные, и мультикластерные ВС и GRID-системы

1.2.1. Принципы построения кластерных ВС

При построении кластерных ВС, как правило, используются массовые аппаратурно-программные средства. Последнее, по существу, является принципом конструирования кластерных ВС, обеспечивающим их высокую технико-экономическую эффективность [95].

Для создания вычислительных кластеров используются и MISD-, и SIMD-, и MIMD-архитектуры, различные функциональные структуры и конструктивные решения.

Основная функционально-структурная единица вычислительных ресурсов в кластерных ВС – это элементарная машина. Конфигурация ЭМ допускает варьирование в широких пределах – от процессорного ядра до ЭВМ, оснащенной средствами коммуникаций и внешними устройствами.

Современные кластерные ВС [61, 83, 88, 98, 184] преимущественно конфигурируются из многопроцессорных узлов и многоядерных процессоров. Коммуникационные среды кластерных ВС строятся на базе коммутато-

ров и имеют иерархическую организацию. На рис 1.3 приведен пример вычислительного кластера, укомплектованного *N* вычислительными узлами, которые объединены сетью связи стандарта Gigabit Ethernet. Каждый узел укомплектован парой двухъядерных процессоров AMD Opteron 275, взаимодействующих по шине HyperTransport. Каждый процессор имеет интегрированный контроллер памяти, через который его ядра имею доступ к их общей памяти. Видно, что коммуникационная среда кластера имеет иерархическую организацию, в которой первый уровень – это сеть связи стандарта Gigabit Ethernet; второй – шина HyperTransport; третий – общая память ядер. Скорости передачи информации на различных уровнях коммуникационной среды существенно различны.

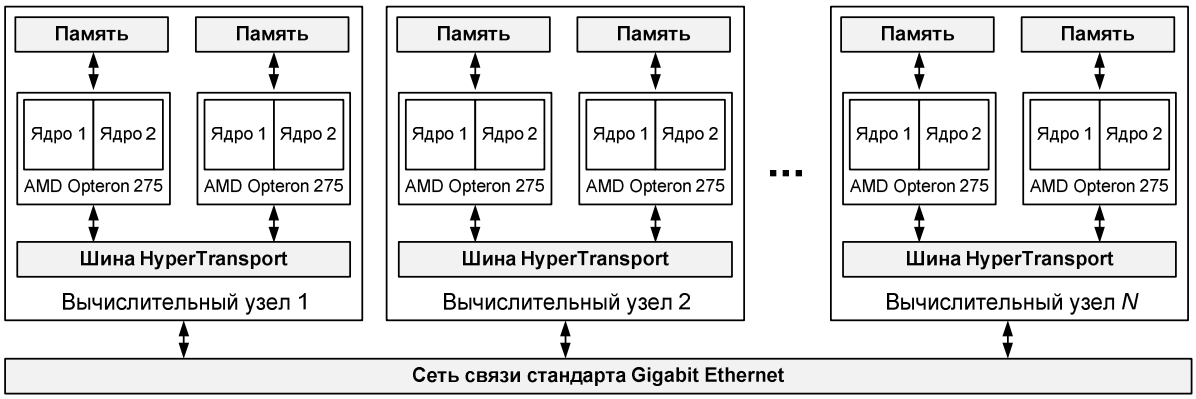


Рис. 1.3. Пример иерархической организации
коммуникационной среды кластерной ВС

Коммуникационные среды современных кластерных ВС преимущественно строятся на базе технологий Gigabit Ethernet, InfiniBand, Myrinet. Для объединения вычислительных узлов в систему используются коммутаторы (switch) с фиксированным количеством входных портов для подключения оконечных устройств (вычислительных узлов, коммутаторов, систем хранения данных и т. п.). В этих условиях для построения большемасштабных кластерных ВС используются различные схемы формирования составных коммутаторов. Например, Fat tree [157, 33] и Clos [33]. Основанная цель – ком-

плексирование заданного числа вычислительных узлов и обеспечение высокой производительности коммуникационной среды.

1.2.2. Мультикластерные ВС и GRID-системы

С развитием вычислительной техники и телекоммуникационных технологий появилась тенденция к построению распределенных ВС путем объединения с помощью локальных и глобальных сетей связи рассредоточенных вычислительных ресурсов начиная от простых ПК и заканчивая ВС с массовым параллелизмом. Так в конце XX столетия в индустрии обработки информации получили развитие GRID-технологии (Global Resource Information Distribution) [95, 126-130]. На основе этих технологий создаются большемасштабные пространственно-распределённые системы обработки информации, способные реализовать параллельные алгоритмы решения суперсложных задач на своих рассредоточенных ресурсах. Такая *GRID-система* представляет собой композицию множества ЭВМ и ВС, пространственно-распределённой коммуникационной сети и программных компонентов для осуществления параллельных вычислений. В GRID-системе могут использоваться гетерогенные и несовместимые вычислительные средства, однако для их совместной работы над параллельными алгоритмами должны быть применены специальные механизмы (например, стандартизованные протоколы) согласованного взаимодействия. GRID-система – это ни что иное как распределённая ВС в смысле введенного выше определения.

Что касается мультикластерных ВС, то это не отдельный класс ВС, а пространственно-распределенные системы, формируемые путем объединения ресурсов рассредоточенных кластеров.

1.2.3. Разработка параллельных программ для кластерных ВС

Вычислительные кластеры относятся к классу систем с распределенной памятью. Параллельные программы для таких ВС приемущественно разрабатываются в модели передачи сообщений (message passing). Широкое распро-

странение получили стандарты, библиотеки и среды для создания параллельных программ в данной модели, в частности стандарт Message Passing Interface (MPI), система Parallel Virtual Machine (PVM), IBM Message Passing Library (MPL) , P4 и др.

В рамках данной модели параллельные ветви программы синхронизируют свою работу путем обмена информацией по каналам межмашинных связей.

1.3. Основные режимы функционирования ВС

В зависимости от сложности задач и характера их поступления в теории вычислительных систем выделяют два основные режима функционирования ВС с программируемой структурой [27, 95]: моно- и мультипрограммный режимы.

1.3.1. Монопрограммный режим

В монопрограммном режиме для решения задачи используются все ресурсы ВС. Задача представляется в виде параллельной программы, число ветвей в которой либо фиксировано, либо допускает варьирование в заданном диапазоне. В качестве единицы ресурса выступает элементарная машина ВС. Все машины используются для решения задачи. Если максимальное число ветвей в параллельной программе менее общего числа ЭМ в системе, то “избыточные” машины используются для повышения надёжности функционирования ВС.

1.3.2. Мультипрограммные режимы

К мультипрограммным относят режимы [27, 95]: 1) обработки наборов задач; 2) обслуживания потоков задач.

При работе ВС в этих режимах одновременно решается несколько задач, следовательно, ресурсы системы делятся между ним.

При организации функционирования ВС в случае обработки набора задач учитывается не только количество задач в наборе, но их параметры: число ветвей в программе (точнее, число машин, на которых она будет выполняться), время решения или вероятностный закон распределения времени решения и др. Алгоритмы организации функционирования ВС [6-7, 27, 41, 43] задают распределение задач по машинам и последовательность выполнения задач на них. В результате становится известным, в каком промежутке времени и на каких машинах (или на какой подсистеме) будет решаться любая задача набора.

Обслуживание потока задач на ВС – принципиально отличается от обработки наборов: задачи поступают в случайные моменты времени, их параметры случайны, следовательно, детерминированный выбор подсистем для решения тех или иных задач исключён. Для режима потока задач применяются методы и алгоритмы [27, 39, 68-73], обеспечивающие стохастически оптимальное функционирование вычислительных систем.

При работе ВС в любом из мультипрограммных режимов система представляется в виде композиции подсистем различных рангов. По мере решения задач эта композиция “автоматически” (с помощью операционной системы) реконфигурируется так, чтобы обеспечить её адекватность текущей мультипрограммной ситуации. Любая подсистема обладает всеми архитектурными свойствами системы, поэтому её организация при решении выделенной ей задачи может осуществляться теми же методами, что и организация работы всей ВС в первом режиме.

1.4. Вложение параллельных программ в распределенные ВС

Время выполнения параллельных программ на распределенных ВС существенно зависит от того насколько они эффективно вложены в систему [15-17, 44, 49, 65, 68-70, 73, 84, 87, 93, 96-99, 103-105, 109-110, 151, 156, 164, 178, 182].

Под эффективным *вложением* (*Task Map*, *Task Allocation*, *Task Assignment*) понимается такое распределение ветвей параллельной программы между ЭМ системы, при котором достигаются минимумы накладных расходов на межмашинные обмены информацией и дисбаланса загрузки ЭМ.

В зависимости от того, чем характеризуется поступившая в систему параллельная программ, возникают различные постановки задачи вложения. Если для параллельной программы задан информационный граф, то говорят о задаче вложения информационного графа (структуры) программы в ВС. В случае, если о параллельной программе известно лишь количество ЭМ, требуемых для её реализации, то рассматривается задача формирования в пределах ВС подсистемы ЭМ, удовлетворяющей заданным критериям качества.

Задача оптимального вложения параллельной программы возникает при организации функционировании ВС в любом из основных режимов [97]: решения сложной задачи, обработки наборов задач, обслуживания потоков задач.

В режиме обслуживания потоков задач в пределах ВС формируются подсистемы ЭМ для выполнения на них поступающих параллельных программ. Элементарные машины, входящие в подсистему, могут принадлежать различным вычислительным системам, размещаться в различных коммуникационных шкафах и т. п. Следствием этого является то, что скорости передачи информации между ЭМ могут быть существенно различными. Поэтому требуется не только формировать подсистемы с требуемым числом ЭМ, но и эффективно вкладывать параллельные программы в них.

В режиме обработки наборов задач строится расписание выполнения параллельных программ. В данном режиме задача вложения возникает или на этапе запуска программы на выделенной подсистеме, или же на этапе синтеза расписания, в случае, если алгоритмы вложения интегрированы в подсистему поддержки мультипрограммных режимов функционирования ВС.

1.4.1. Задача оптимального вложения параллельных программ

Задача оптимального вложения параллельной программы в ВС относится к дискретной оптимизации и является трудноразрешимой [93, 96, 105, 109]. Рассмотрим ее формальную постановку.

Пусть распределенная ВС укомплектована N элементарными машинами, множество которых обозначим через $C = \{1, 2, \ldots, N\}$. На множестве C задана весовая функция $\omega_c = \omega(c)$, ставящая в соответствие ЭМ их производительность ($c \in C$, $[\omega(c)] = \text{FLOPS}$). Каналы связи между ЭМ системы характеризуются показателями производительности $b_{pq} = b(p,q,m)$ и $l_{pq} = l(p,q,m)$ – пропускная способность и латентность канала связи между ЭМ $p,q \in C$ при передачи сообщений размером m байт ($[b(p,q,m)] = $ байт/с, $[l(p,q,m)] = $ с).

Считается, что поступившая на выполнение параллельная программа представлена информационным графом $G = (V,E)$. Где $V = \{1, 2, \ldots, M\}$ – множество ветвей параллельной программы, а $E \subseteq V \times V$ – множество информационно-логических связей между её ветвями (обмены).

На множествах V и E заданы весовые функции: $w_i = w(i)$ – функция, ставящая в соответствие ветвям количество арифметических и логических операций, выполняемых ими ($i \in V$); $d_{ij} = d(i,j)$ – объем данных, передаваемых между ветвями $i, j \in V$ за время выполнения параллельной программы ($[d(i,j)] = $ байт).

Задача оптимального вложения параллельной программы в ВС заключается в отыскании инъективной функции $f : V \to C$, ставящей в соответствие ветвям параллельной программы ЭМ системы. Требуется найти x_{ij}:

$$X = \{x_{ij} : i \in V, j \in C\},$$

$$x_{ij} = \begin{cases} 1, & \text{если } f(i) = j; \\ 0 & \text{иначе.} \end{cases}$$

Качество вложения параллельной программы в ВС оценивается ожидаемым временем T ее выполнения, которое определяется максимальным из времен выполнения ее ветвей [170]. Время t_i выполнения ветви $i \in V$ параллельной программы складывается из времени t_i' выполнения арифметических и логических операций ЭМ и времени t_i'' взаимодействия со смежными ветвями [69-70, 110, 116, 133].

$$t_i' = \sum_{p=1}^{N} x_{ip} \cdot t'(i, p),$$

$$t_i'' = \sum_{j=1}^{M} \sum_{p=1}^{N} \sum_{q=1}^{N} x_{ip} \cdot x_{jq} \cdot t''(i, j, p, q),$$

где $t'(i, p)$ – время выполнения ветви i на ЭМ p; $t''(i, j, p, q)$ – время взаимодействия между ветвями i, j, распределенными на ЭМ p и q, соответственно.

Вид функций $t'(i, p)$ и $t''(i, j, p, q)$ зависит от конфигурации ВС и принятых моделей для оценки времени выполнения арифметических и логических операций элементарной машиной и времени передачи данных по каналам межмашинных связей.

Тогда

$$T(X) = \max_{i \in V} \{t_i' + t_i''\} =$$

$$= \max_{i \in V} \left\{ \sum_{p=1}^{N} x_{ip} \cdot t'(i, p) + \sum_{j=1}^{M} \sum_{p=1}^{N} \sum_{q=1}^{N} x_{ip} \cdot x_{jq} \cdot t''(i, j, p, q) \right\}, \qquad (1.7)$$

Учитывая требование инъективности функции f, сформулируем задачу оптимального вложения в ВС параллельной программы с целью минимизации времени ее выполнения:

$$T(X) = \max_{i \in V} \{t_i' + t_i''\} \to \min_{(x_{ij})} \tag{1.8}$$

при ограничениях:

$$\sum_{j=1}^{N} x_{ij} = 1, \quad i = 1,2,\ldots,M, \tag{1.9}$$

$$\sum_{i=1}^{M} x_{ij} \le 1, \quad j = 1,2,\ldots,N, \tag{1.10}$$

$$x_{ij} \in \{0,1\}, \quad i \in V, \quad j \in C. \tag{1.11}$$

Ограничения (1.9), (1.11) гарантируют назначение каждой ветви параллельной программы на единственную ЭМ, ограничения (1.10) обеспечивают назначение на ЭМ не более одной ветви.

Задача (1.8) – (1.11) оптимального вложения параллельной программы в ВС является трудноразрешимой [93, 96, 105, 109]. Мощность множества D допустимых решений задачи оценивается величиной $|D| = A_N^M = N!/(N-M)!$. Для большемасштабных распределенных ВС отыскание точного решения связано со значительной вычислительной сложностью. Целесообразно применение приближенных методов и алгоритмов решения задачи.

1.4.2. Алгоритмы вложения параллельных программ

Существующие методы и алгоритмы вложения параллельных программ в ВС различаются по типу систем, на которые они ориентированы, по используемым моделям параллельных программ, показателям оптимальности вложения, точности, централизованности или децентрализованности алгоритмов, по статической или динамической схеме формирования вложений.

В работах [84, 103, 109, 155, 176-177, 181, 183] предложены алгоритмы вложения параллельных программ, в которых ВС представлена в виде графа межмашинных связей. Предполагается, что информация между ЭМ переда-

ётся по кратчайшему пути и все каналы связи однородны. В качестве показателя производительности канала связи между ЭМ используется длина пути (в смысле теории графов). В работах [69-70, 114, 122, 140, 141, 144] описаны алгоритмы вложения в системы фиксированной структуры (например, трехмерные торы, гиперкубы и т. д.). Применение подобных алгоритмов осложнено тем, что современные ВС мультиархитектурны, их коммуникационные среды имеют иерархическую организацию и, как следствие, неоднородные каналы связи между ЭМ. Кроме того, большинство систем строится на базе коммутаторов, в которых применяются алгоритмы динамической маршрутизации. Поэтому целесообразно использование моделей вычислительных систем, в которых производительность каналов связи характеризуется такими показателями как пропускная способности и латентность.

Эффективное вложение в ВС параллельной программы требует учета структуры информационных обменов и объемов данных, передаваемых между ее ветвями. В работах [84, 109, 155, 176-177] предложены алгоритмы, опирающиеся лишь на структуру информационного графа программы и игнорирующие объемы и интенсивность межмашинных обменов. Однако учет таких параметров практически необходим. Анализ популярных свободно распространяемых параллельных программ, библиотек и тестов производительности High-Performance Linpack, NAS Parallel Benchmarks и SPEC MPI2007 показывает, что большинство из них характеризуется неоднородными информационными графами. Неоднородность выражается в объемах данных, передаваемых между ветвями, размерах используемых сообщений, а также в схемах реализуемых обменов.

В качестве показателей оптимальности вложения используют время выполнения параллельной программы или функцию, минимизация которой доставляет минимум времени выполнения программы.

Алгоритмы вложения, предложенные в [103-104, 110, 112-113, 116, 164], опираются на предположение о том, что время выполнения параллельной

программы – это сумма времен выполнения ее ветвей. Однако практика решения промышленных задач и анализа их производительности показывает, что время выполнения программы определяется максимальным из времен выполнения ее ветвей [170]. Кроме того, в ряде работ [69-70, 110, 135, 168] используются составные показатели в виде линейной комбинации двух функций, например, средней загруженности элементарных машин в системе и времени передачи данных по каналам связи. Использование подобных показателей осложнено выбором коэффициентов пропорциональности, которыми связанны функции и зачастую их несовместимыми размерностями. Например, размерность первой функции – количество операций, а второй – единицы времени. Вложение параллельных программ в пространственно-распределенные ВС требует учета в показателе времени доставки параллельной программы до элементарных машин системы.

Из сказанного следует, что существующие алгоритмы не позволяют строить эффективные вложения параллельных программ в ВС с иерархической организацией коммуникационных сред. Актуальной является разработка нетрудоемких методов и алгоритмов вложения параллельных программ в ВС с иерархической организацией коммуникационных сред.

1.4.3. Алгоритмы формирования подсистем в ВС

Вложения в ВС параллельных программ, для которых не заданы информационные графы, осуществляется путем формирования подсистемы элементарных машин и распределения по ней параллельных ветвей программы. Формируемая подсистема должна обеспечивать эффективную реализацию параллельных программ и доставлять минимум времени их выполнения. Доминирующее большинство алгоритмов [106, 108, 115, 159] формирования в пределах ВС подсистем ориентированы на системы с определенными структурами сетей межмашинных связей (например, на 2D-решетки, гиперкубические и тороидальные структуры) и стремятся сохранять топологическое по-

добие подсистемы структуре ВС. Возможности применение таких алгоритмов для ВС с иерархической организацией, каналы связи в которых имеет различные производительности, существенно ограничены. Поэтому актуальной является разработка алгоритмов формирования подсистем, допускающих применение как в системах со статической структурой сети межмашинных связей, так и в системах с динамической структурой.

1.4.4. Обзор средств вложения параллельных программ и формирования подсистем

Современные ВС функционируют под управлением систем пакетной обработки заданий (например, TORQUE/OpenPBS, IBM LoadLeveler, Sun Grid Engine, Altair PBS Pro, SLURM, Cleo и пр.), которые реализуют поддержку мультипрограммных режимов функционирования. Системы пакетной обработки заданий (СПОЗ) поддерживают очереди задач и осуществляют планирование вычислений. Для каждой параллельной программы с помощью поддерживаемых алгоритмов в пределах системы формируется подсистема ЭМ. Распространение получили элементарные алгоритмы формирования подсистем: PAL (Processor Allocation Linear) – осуществляет выбор первых M свободных ЭМ, и PAR (Processor Allocation Random) – генерирует случайное подмножества из M свободных ЭМ.

Начальное вложение параллельной программы в выделенную подсистему также осуществляется СПОЗ. Данная задача чаще всего решается циклическим (Task Map Round Robin, TMRR) или линейным (Task Map Linear) алгоритмами формирования вложений. Алгоритм TMRR распределяет параллельные ветви по ЭМ системы из N вычислительны узлов в следующем порядке: первая ветвь на первый вычислительный узел, вторая на второй, …, ветвь N на узел N, ветвь $N + 1$ на узел 1 и т. д. Алгоритм TML распределяет ветви на первые M свободных ЭМ.

Средства оптимизации вложения параллельных программ присутствуют и в коммуникационных библиотеках стандарта MPI (MPICH2, OpenMPI, MVAPICH2, Intel MPI, HP-MPI). Вложение реализуется утилитой mpiexec, поддерживающей в большинстве случаев лишь алгоритмы TMRR и TML.

1.5. Выводы

1. Современные вычислительные системы являются мультиархитектурными, их коммуникационные среды имеют иерархическую организацию.

2. Эффективность эксплуатации ресурсов распределенных ВС существенно зависит от того, как организованно их функционирование. В любом из режимов функционирования возникает задача эффективного вложения параллельной программы в ВС. Под эффективным вложением понимается такое распределение ветвей параллельной программы, при котором достигаются минимумы накладных расходов на межмашинные обмены и дисбаланса загрузки элементарных машин.

3. Задача оптимального вложения является трудноразрешимой. Актуальной является разработка приближенных методов и алгоритмов вложения параллельных программ в распределенные ВС.

ГЛАВА 2. АЛГОРИТМЫ ВЛОЖЕНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ В ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

В этой главе рассматривается математическая модель коммуникационных сред вычислительных систем с иерархической организацией. Ставится задача оптимального вложения информационных графов параллельных программ в такие системы. Предлагается нетрудоемкий метод поиска субоптимальных решений задачи.

Для вложения в ВС параллельных программ, для которых не заданы информационные графы, предложены эвристические алгоритмы формирования подсистем, допускающих эффективную реализацию основных схем межмашинных обменов информацией.

2.1. Оптимизация вложения параллельных программ в ВС с иерархической организацией коммуникационных сред

Рассмотрим задачу оптимального вложения параллельных программ в ВС с иерархической организаций. К таким системам относятся современные мультиархитектурные ВС, например, большинство систем из списка Top500 [80].

2.1.1. Модель ВС с иерархической организацией коммуникационной среды

Пусть ВС с иерархической организацией укомплектована N однородными ЭМ. Коммуникационная среда системы может быть представлена в виде дерева, содержащего L уровней. Каждый уровень системы образован отдельным видом структурных элементов системы (например, телекоммуникационные шкафы, вычислительные узлы и т. п.), которые объединены каналами связи своего уровня. На уровне l размещено n_l элементов, $l \in \{1, 2, \ldots, L\}$. Для каждого элемента на уровне l задано количество n_{lk},

$k \in \{1, 2, \ldots, n_l\}$, его прямых дочерних узлов. Дополнительно определим функцию $g(l, k_1, k_2)$ – номер уровня, на котором находится элемент, являющийся ближайшим общим предком для элементов k_1, $k_2 \in \{1, 2, \ldots, n_l\}$ уровня l. Например, на рис. 2.1 приведен фрагмент ВС с иерархической организацией – вычислительный кластер из 3 двухпроцессорных узлов на базе двухъядерных процессоров AMD Opteron 275. В приведенном примере в качестве элементарных машин выступают процессорные ядра. Первый уровень коммуникационной среды образован сетью связи стандарта InfiniBand, через которую взаимодействуют узлы, второй уровень представлен шиной HyperTransport, через которую взаимодействую процессоры узла, третий уровень – средства доступа процессорных ядер к их общей памяти.

На рис. 2.1 ближайшим общим предком для элементов 3, 5 уровня 3 является элемент 1 уровня 1, т.е. $g(3,3,5) = 1$.

Обозначим C_{lk} – множество элементарных машин, принадлежащих потомкам элемента с номером k на уровне l. Очевидно, что $C_{11} = C$. Положим $c_{lk} = |C_{lk}|$. На структуру дерева наложены следующие ограничения
$\forall l \in \{1,2,\ldots,L\}$, $\forall k_1, k_2 \in \{1,2,\ldots,n_l\}$:

$$n_{lk_1} = n_{lk_2}, \tag{2.1}$$

$$c_{lk_1} = c_{lk_2}. \tag{2.2}$$

Первое ограничение (2.1) гарантирует, что элементы одного уровня имеют одинаковое количество дочерних узлов. Второе (2.2) обеспечивает равенство числа ЭМ принадлежащих потомкам элементов одного уровня.

Для каждого уровня коммуникационной среды известны значения показателей производительности каналов связи на нем. Пусть b_l – значение пропускной способности каналов связи на уровне l ($[b_l]$ = байт/с). Например, на

рис. 2.1 уровень 1 представлен технологией InfiniBand, для версии DDR InfiniBand характерное значение b_l = 2 Гбайт/с.

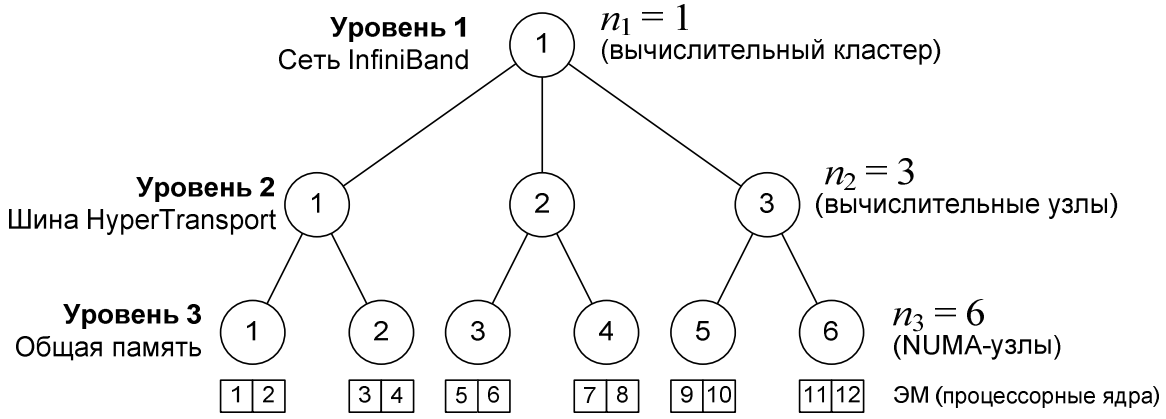


Рис. 2.1. Пример иерархической организции
коммуникационной среды вычислительного кластера:
три вычислительных узла на базе 2 x AMD Opteron 275;
$N = 12$; $L = 3$; $n_{23} = 2$; $C_{23} = \{9, 10, 11, 12\}$; $c_{23} = 4$; $g(3, 3, 4) = 2$; $z(1, 7) = 1$

2.1.2. Оценка ожидаемого времени выполнения параллельных программ на вычислительных системах

Выразим ожидаемое время выполнения параллельной программы на ВС с иерархической организацией. Считаем, что параллельная программа представлена информационным графом $G = (V, E)$, где $V = \{1, 2, \ldots, M\}$ – множество параллельных ветвей программы; $E \subseteq V \times V$ – множество информационно-логических связей между ее параллельными ветвями (обмены информацией). Обозначим за d_{ij} вес ребра $(i, j) \in E$, отражающий объем данных, передаваемый по нему за время выполнения программы ($[d_{ij}]$ = байт). Время выполнения параллельной программы существенно зависит от эффективности её вложения в систему.

Вложение параллельной программы в ВС характеризуется инъективной функцией $f: V \to C$, ставящей в соответствие ветвям параллельной программы элементарные машины системы. Функция f задается значениями x_{ij}:

$$X = \{ x_{ij} : i \in V, j \in C \},$$

$$x_{ij} = \begin{cases} 1, & \text{если } f(i) = j; \\ 0 & \text{иначе.} \end{cases}$$

Время T выполнения параллельной программы с заданным вложением X в систему определяется максимальным из времен выполнения ее ветвей. По условию, ЭМ системы имеют одинаковую производительность, поэтому в функции $T(X)$ будем учитывать только накладные расходы на межмашинные обмены. Следовательно

$$T(X) = \max_{i \in V} \{ t_i'' \} = \max_{i \in V} \left\{ \sum_{j=1}^{M} \sum_{p=1}^{N} \sum_{q=1}^{N} x_{ip} \cdot x_{jq} \cdot t''(i, j, p, q) \right\}. \qquad (2.3)$$

Для оценки времени передачи сообщений между параллельными ветвями будем использовать модель Хокни (R. Hockney) [91-92, 136-137]. Тогда функция $t''(i, j, p, q)$ может быть записана в виде $t''(i, j, p, q) = d_{ij} / b_{z(p,q)}$. Функция $z(p, q)$ ставит в соответствие ЭМ p и q номер уровня элемента, являющегося ближайшим общим предком для них, уровень, через который они взаимодействуют. На рис. 2.1 ЭМ 1 и 7 принадлежат различным узлам, взаимодействие между которыми осуществляется через сеть InfiniBand, следовательно, $z(1, 7) = 1$.

2.1.3. Задача оптимального вложения параллельных программ в ВС

Учитывая инъективность функции f, сформулируем задачу оптимального вложения параллельной программы в ВС с иерархической организацией. В

качестве показателя оптимальности вложения будем использовать ожидаемое время (2.3) выполнения параллельной программы.

$$T(X) = \max_{i \in V} \left\{ \sum_{j=1}^{M} \sum_{p=1}^{N} \sum_{q=1}^{N} x_{ip} \cdot x_{jq} \cdot d_{ij} / b_{z(p,q)} \right\} \to \min_{(x_{ij})} \tag{2.4}$$

при ограничениях:

$$\sum_{j=1}^{N} x_{ij} = 1, \quad i = 1,2,\ldots,M, \tag{2.5}$$

$$\sum_{i=1}^{M} x_{ij} \le 1, \quad j = 1,2,\ldots,N, \tag{2.6}$$

$$x_{ij} \in \{0,1\}, \quad i \in V, \quad j \in C. \tag{2.7}$$

Ограничения (2.5), (2.7) гарантируют назначение каждой ветви параллельной программы на единственную ЭМ, ограничения (2.6) обеспечивают назначение на машину не более одной ветви.

Задача (2.4) – (2.7) является трудноразрешимой. Рассмотрим приближенные методы ее решения.

2.2. Иерархический метод вложения параллельных программ в ВС

Для решения задачи вложения параллельных программ в ВС с иерархической организацией коммуникационных сред предложен метод [44, 54], основанный на решении задачи разбиении взвешенного графа на k непересекающихся подмножеств. Рассмотрим последнюю задачу.

2.2.1. Задача оптимального разбиения графа на k непересекающихся подмножеств

Пусть дан взвешенный граф $G' = (V', E')$; требуется отыскать разбиение вершин $V' = \{1, 2, \ldots, n\}$ графа на k непересекающихся подмножеств

V_1', V_2', …, V_k' с целью минимизации максимальной суммы весов ребер, инцидентных любому подмножеству разбиения. Количество элементов в каждом из подмножеств не должно превышать заданного числа s. Обозначим $E'(i, j)$ – множество ребер соединяющих вершины из подмножеств V_i' и V_j', $i, j \in \{1, 2,\ldots, k\}$

$$E'(i, j) = \{(u, v) \in E' : u \in V_i', v \in V_j', i \neq j\};$$

$c(u, v, i, j)$ – вес ребра, инцидентного вершинам из разных подмножеств $u \in V_i'$ и $v \in V_j'$

$$c(u, v, i, j) = w(u, v) \cdot W(i, j),$$

где $w(u, v)$ – вес ребра $(u, v) \in E'$, $W(i, j)$ – весовой коэффициент для ребер, соединяющих вершины из подмножеств i и j.

Учитывая ограничения, накладываемые на подмножества V_i', получаем задачу оптимального разбиения графа на k непересекающихся подмножеств.

$$F(V_1', V_2', \ldots, V_k') = \max_{i=1,k} \left\{ \sum_{j=1}^{k} \sum_{(u,v) \in E'(i,j)} c(u, v, i, j) \right\} \to \min_{(V_1', V_2', \ldots, V_k')} \tag{2.8}$$

при ограничениях:

$$V_1' \cap V_2' \cap \ldots \cap V_k' = \varnothing, \tag{2.9}$$

$$V_1' \cup V_2' \cup \ldots \cup V_k' = V', \tag{2.10}$$

$$|V_i'| > 0, \quad i = 1,2,\ldots,k, \tag{2.11}$$

$$|V_i'| \leq s, \quad i = 1,2,\ldots,k. \tag{2.12}$$

На рис. 2.2 приведен пример взвешенного графа. На рис. 2.3 представлен возможный вариант разбиения графа на три подмножества.

Суммарный вес ребер инцидентных первому подмножеству V_1' равен $w(1, 5)W(1, 2) + w(6, 8)W(1, 3) + w(2, 3)W(1, 3) = 22$; для второго подмножества V_2' – 40; для третьего подмножества V_3' – 38. Значение целевой функции $F(V_1', V_2', V_3') = 40$.

Известно [131], что задача (2.8) – (2.12) разбиения взвешенного графа на k непересекающихся подмножеств является трудноразрешимой.

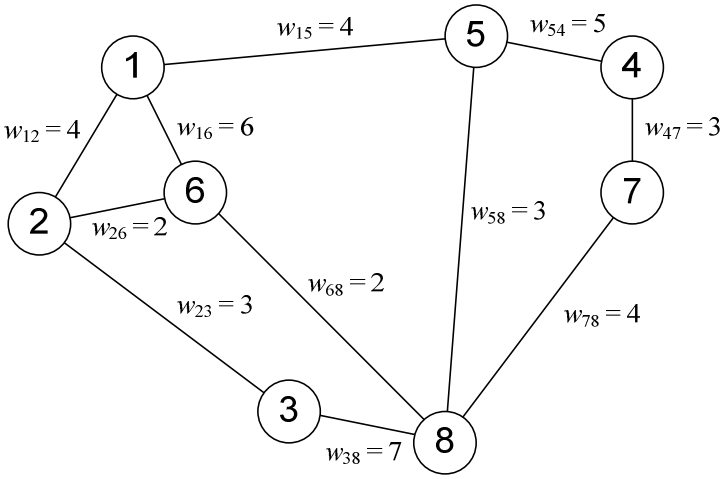


Рис. 2.2. Пример взвешенного графа

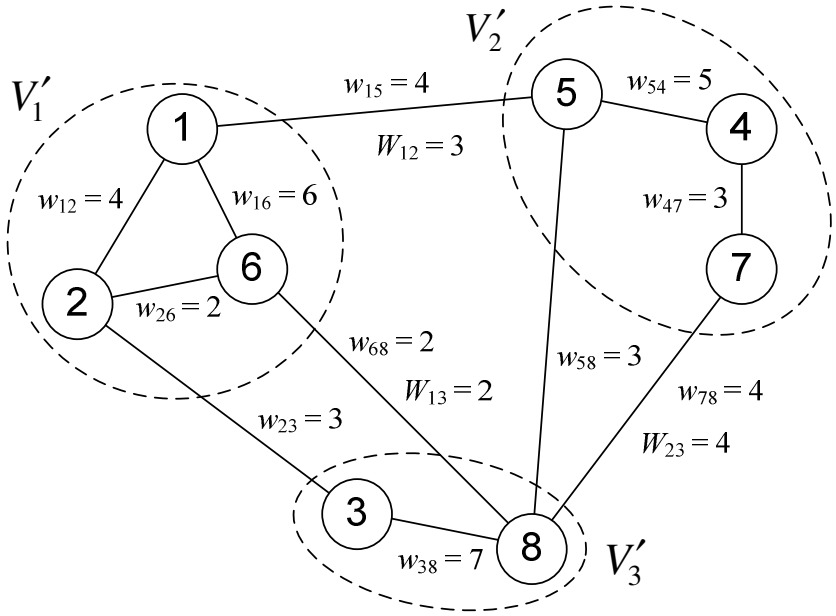


Рис. 2.3. Разбиение взвешенного графа на три подмножества:
$k = 3$; $s = 3$; $F(V_1', V_2', V_3') = 40$

2.2.2. Метод вложения параллельных программ в ВС

Предложим метод решения задачи вложения параллельных программ в ВС с иерархической организацией. В основе метода лежит идея разбиения информационного графа параллельной программы на подмножества с параллельными ветвями, обменивающимися большими объёмами данных, и их вложения в ЭМ, связанными быстрыми каналами связи.

Суть метода. Для исходных данных задачи (2.4) – (2.7) одним из известных методов [111, 134, 146-149, 169] строится решение задачи (2.8) – (2.12) разбиения взвешенного графа на k непересекающихся подмножеств. В задаче (2.8) – (2.12) полагаем, что $V' = V$, $k = [(M-1)/c_{L1}] + 1$, $s = c_{L1}$, $c(u, v, i, j) = d_{uv} / b_{g(L,i,j)}$. Решение – k подмножеств V_1', V_2', …, V_k'.

По решению задачи (2.8) – (2.12) строится решение задачи (2.4) – (2.7). Вложение $f : V \to C$ формируется следующим образом. Ветви с номерами из подмножества V_i' назначаются на ЭМ из множества C_{Li}, $i = 1,2,...,k$. Значение целевой функции T задачи (2.4) – (2.7) не зависит от выбора номера ЭМ во множестве C_{Li}. Это объясняется тем, что каналы связи между элементарными машинами в подмножестве C_{Li} имеют одинаковую производительность.

Обозначим описанный метод TMGP (Task Map Graph Partitioning). В листинге 2.1 приведен его псевдокод.

В псевдокоде метода используется структура данных очередь (*queue*) [1, 31, 63], для которой определены операции *Enqueue*(*Q*, *i*) – добавление элемента *i* в конец очереди *Q*; *Enqueue*(*Q*, *Q'*) – добавление элементов очереди *Q'* в конец очереди *Q*; *Dequeue*(*Q*) – извлечения первого элемента из очереди *Q*; *QueueSize*(*Q*) – определение количества элементов в очереди *Q*. Трудоемкость перечисленных операций составляет $O(1)$.

Здесь и далее при описании методов и алгоритмов используются обозначения принятые в [18, 20, 31, 63].

Листинг 2.1. Псевдокод метода TMGP

Входные данные: $G = (V, E)$ – информационный граф параллельной программы; N, L, n_l, n_{lk}, C, C_{lk}, c_{lk} – описание ВС с иерархической организацией ($l \in \{1, 2, \ldots, L\}$, $k \in \{1, 2, \ldots, n_l\}$).

Выходные данные: $x[i, j]$ – вложение; $x[i, j] = 1$, если ветвь i назначена на ЭМ j, иначе $x[i, j] = 0$.

```
1   k ← [(M − 1)/c_L1] + 1
2   s ← c_L1
3   (p_1, p_2, …, p_M) ← PartGraph(G, k, s)
4   for i ← 1 to M do
5       c ← Dequeue(C_{p_i})
6       x[i, c] ← 1
7   end for
```

Поясним смысл основных этапов работы метода. Функция *PartGraph* реализует решение задачи (2.8) – (2.12) одним из известных методов [111, 134, 146-149, 169] и возвращает список $p_1, p_2, \ldots, p_M$ номеров подмножеств разбиения, к которым принадлежат вершины графа G; $p_i \in \{1,2,...,k\}$ – это номер подмножества, которому принадлежит вершина i информационного графа G. C_j – это очередь номеров ЭМ из подмножества C_{Lj}, $j \in \{1,2,...,n_L\}$.

На рис. 2.4 приведен пример вложения информационного графа задачи в вычислительный кластер с иерархической организацией коммуникационной среды. Разбиение графа получено приближенным решением задачи (2.8) – (2.12) с параметрами: $M = 12$, $c_{L1} = 4$, $k = [(M - 1)/c_{L1}] + 1 = 3$, $s = 4$.

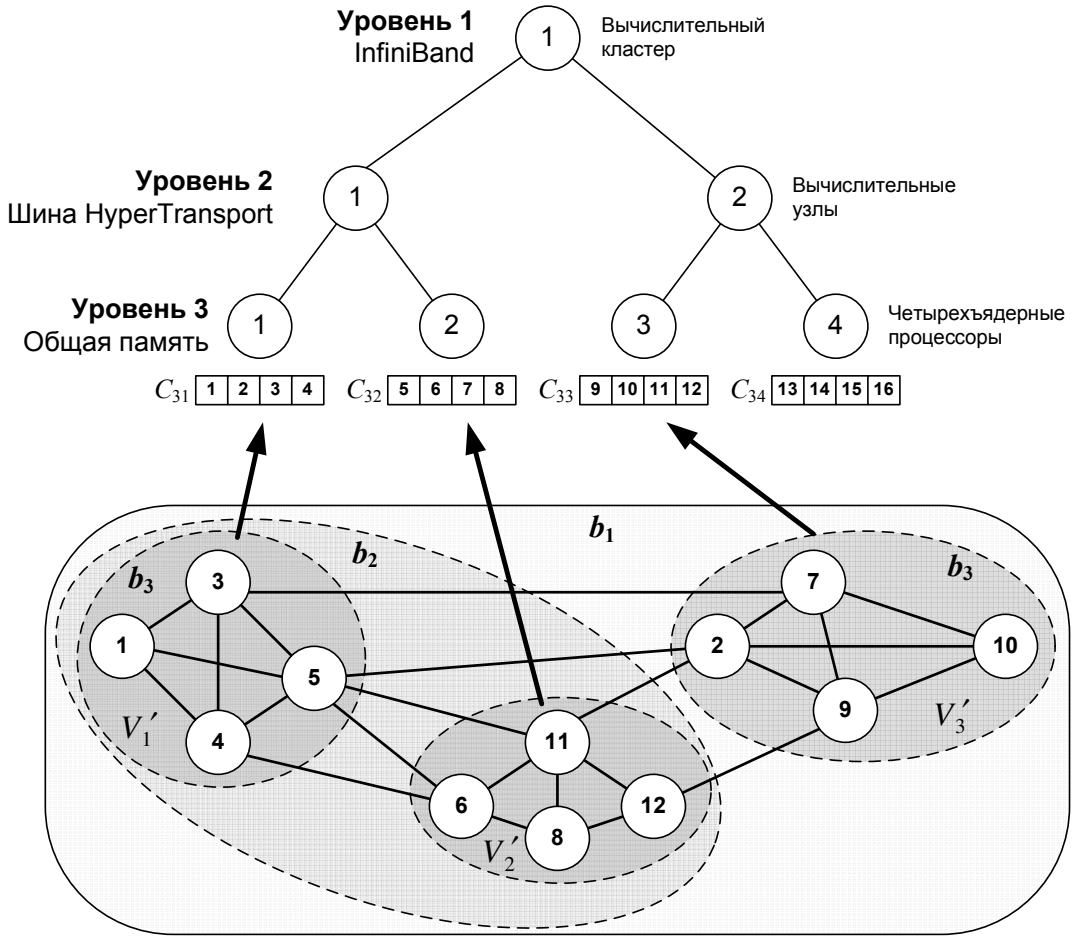


Рис. 2.4. Пример вложения информационного графа в ВС
с иерархической организацией:
$N = 16$; $L = 3$; $M = 12$; $b_1 = 2$ ГБ/с; $b_2 = 6$ ГБ/с; $b_1 = 8$ ГБ/с

Важно отметить, что производительность каналов связи между элементарными машинами, входящими в подмножества C_{31}, C_{32}, C_{33}, C_{34}, различна. Для рассматриваемого примера функция $c(u,v,i,j) = w(u,v) \cdot W(i,j) = d_{uv} / b_{g(L,i,j)}$ строилась следующим образом:

$$w(u,v) = d_{uv},\ W(i,j) = 1/b_{g(L,i,j)},\ u,v \in \{1,2,...,12\},\ i,j = \{1,2,3,4\},$$
$$W(1,2) = 1/(6 \cdot 2^{30}),\ W(1,3) = 1/(2 \cdot 2^{30}),\ W(1,4) = 1/(2 \cdot 2^{30}),$$
$$W(2,3) = 1/(2 \cdot 2^{30}),\ W(2,4) = 1/(2 \cdot 2^{30}),\ W(3,4) = 1/(6 \cdot 2^{30}).$$

Обозначим количество операций требуемых для выполнения функции *PartGraph* через T_{GP}. Справедливо утверждение.

Утверждение 2.1. *Вычислительная сложность метода TMGP равна*

$$T_{TMGP} = O(T_{GP} + M).$$

Доказательство. Трудоемкость метода определяется вычислительной сложностью двух последовательно выполняемых шагов – формирование подмножеств V_1', V_2', …, V_k' функцией *PartGraph* и построение вложения *X*.

Трудоемкость формирования вложения равна $O(M)$, так как требует назначения каждой ветви параллельной программы на элементарную машину. Следовательно, суммарная трудоемкость составляет

$$T_{TMGP} = O(T_{GP} + M).$$

Утверждение доказано.

Задача оптимального разбиения взвешенного графа на *k* непересекающихся подмножеств трудноразрешима. Эффективность метода TMGP определяется алгоритмами решения задачи разбиения графа на *k* непересекающихся подмножеств.

Для решения задачи разбиения графа на *k* непересекающихся подмножеств существуют [111, 134, 146-149, 169] точные и приближенные методы и алгоритмы, характеризующиеся различными временными и пространственными сложностями. Большинство известных алгоритмов имеют практические ограничения на размер обрабатываемых графов. Интерес представляют многоуровневые алгоритмы разбиения графов [107, 134, 146-148], позволяющие получать субоптимальные решения задачи (2.8) – (2.12) и характеризующиеся невысокой вычислительной сложностью.

2.2.3. Многоуровневые методы разбиения графов

Рассмотрим основные этапы работы многоуровневых методов разбиения графов (рис. 2.5). Имеется исходный граф $G_0 = (V_0, E_0)$, который требуется разбить на *k* непересекающихся подмножеств, так, чтобы целевая функция (2.8) приняла субминимальное значение.

1) Этап сжатия графа (coarsening). Исходный граф G_0 преобразуется в последовательность графов $G_1, G_2, \ldots, G_m$ меньших размерностей, таких, что $|V_0| > |V_1| > ... > |V_m|$. Графы меньших размерностей формируются построением наибольших паросочетаний (maximal matching) и стягиванием согласованных вершин в мультивершины. На рис. 2.6 приведен пример стягивания вершин графа G_{i-1} и формирования графа G_i.

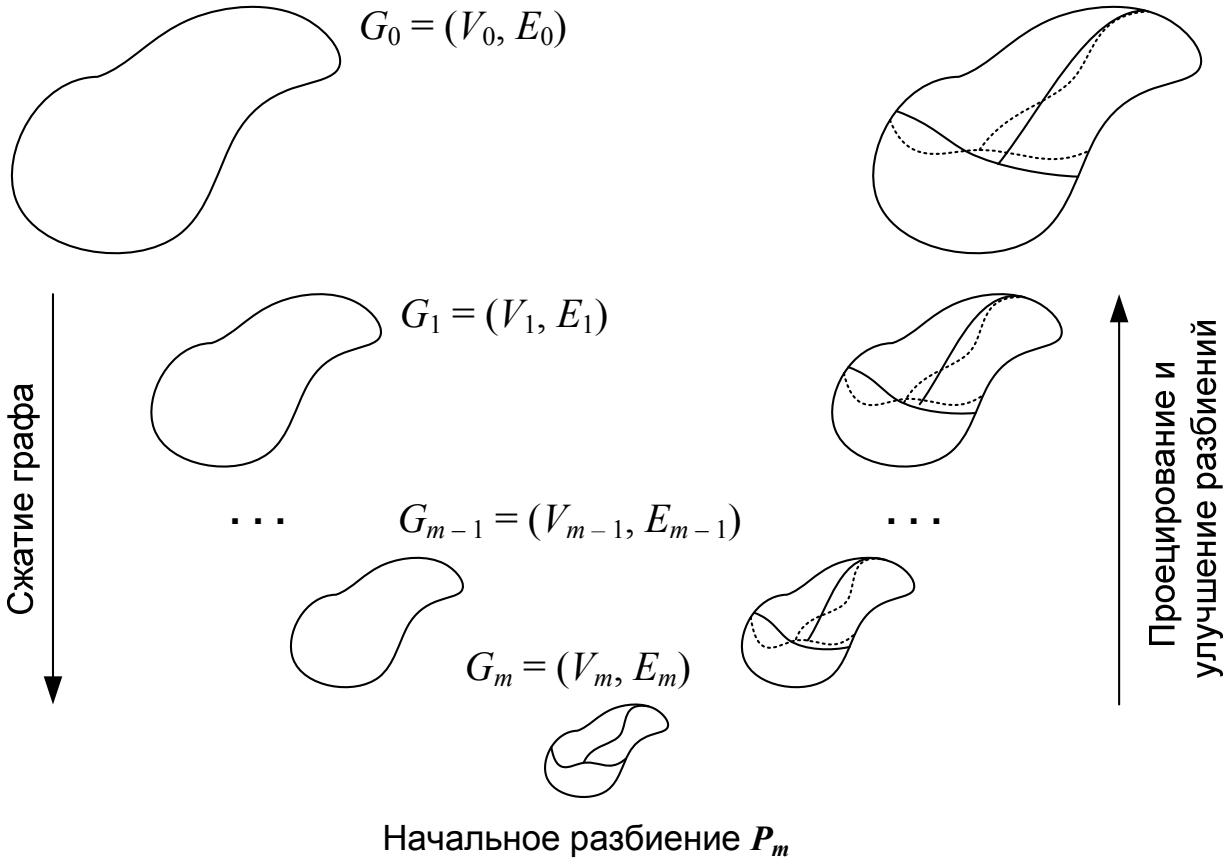
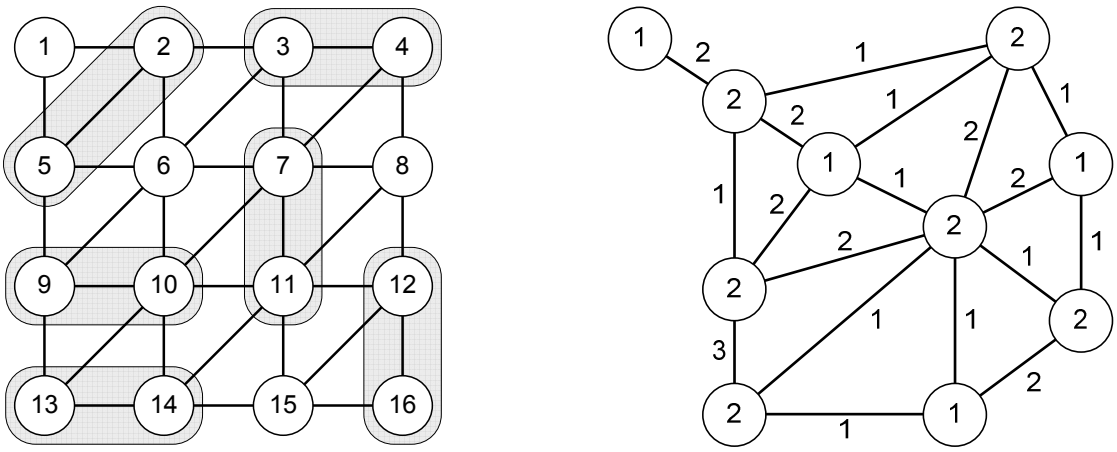


Рис. 2.5. Схема работы многоуровневых алгоритмов разбиения графов:
——— – исходное разбиение P_i; ..... – улучшенное разбиение P_{i-1}

2) Этап начального разбиения (partitioning). Граф $G_m = (V_m, E_m)$ малой размерности разбивается одним из известных алгоритмов на k непересекающихся подмножеств. Результат – это разбиение P_m. На данном этапе применяются алгоритмы рекурсивной бисекции графа, а для формирования разделения графа на две части используются комбинаторные [169] и спектральные

методы [107, 169]. Бисекция графа – это разделение его вершин на два непересекающихся подмножества.

3) Этап улучшения разбиений (refinement). Разбиение P_m проецируется на граф $G_{m-1} = (V_{m-1}, E_{m-1})$. Получается разбиение P_{m-1}, которое улучшается (в смысле показателя (2.8)) известными алгоритмами локальной оптимизации. Процедура проецирования разбиения и его улучшения продолжается, пока не будет достигнут исходный граф G_0. В основе большинства алгоритмов улучшения разбиений лежит модификация нетрудоемкой эвристики Кернигана-Лина (Kernighan-Lin) [123].



а *б*

Рис. 2.6. Пример сжатия графа:
а – исходный граф с выделенными ребрами паросочетания;
б – сжатый граф с мультивершинами

Основное достоинство многоуровневых алгоритмов – это возможность обрабатывать графы больших размерностей за приемлемое время.

2.2.4. Алгоритм вложения параллельных программ в ВС

На основе метода TMGP разработан алгоритм [44, 54] вложения параллельных программ в ВС с иерархической организацией коммуникационных

сред. Разбиение информационных графов программ осуществляется алгоритмом, предложенным в работе [148].

Обозначим алгоритм TMMGP (Task Map Multilevel Graph Partitioning). Опишем его основные шаги.

На первом этапе осуществляется сжатие информационного графа $G = (V, E)$ задачи и построение последовательности G_1, G_2, …, G_m. Сжатие осуществляется до тех пор, пока размеры графа G_m не позволят эффективно применить к нему алгоритм рекурсивной бисекции (recursive bisection). Эмпирически установлено [146-148], что сжатие целесообразно осуществлять, пока в графе G_m не останется M' мультивершин.

$$M' = \max\left\{\frac{M}{40 \cdot \log_2 k}, 20 \cdot k\right\}.$$

Сжатие графа $G_i = (V_i, E_i)$ осуществляется стохастическим алгоритмом HEM (Heavy Edge Matching) [146]. Вершины графа посещаются в случайном порядке. Если вершина $u \in V_i$ не посещалась, то среди её смежных необработанных вершин отыскивается вершина $v \in V_i$ с максимальным значением веса ребра $(u, v) \in E_i$. Ребро $(u, v) \in E_i$ включается в паросочетание, вершины u, v отмечаются как посещенные. Трудоемкость алгоритма HEM равна $O(|E_i|)$. Оценим количество итераций сжатия графов алгоритмом HEM для достижения графа G_m: $|V_m| \le M'$.

При сжатии графов и переходе от графа G_i к графу G_{i+1} количество вершин в последнем уменьшается вдвое. Справедливо неравенство

$$|V_i| > |V_{i+1}| \ge |V_i|/2.$$

Нетрудно заметить, что количество итераций m сжатия графов есть m-й член геометрической прогрессии со знаменателем $1/2$ и первым членом равным 1. Отсюда по определению m-го члена геометрической прогрессии

$$M' = 1 \cdot (1/2)^{m-1},\ m = \log_2 \frac{M}{M'} + 1.$$

На втором этапе рекурсивной бисекцией графа G_m на k подмножеств формируется начальное разбиение P_m. Бисекция реализуется алгоритм GGP (Graph Growing Partitioning) [148, 169]. В основе алгоритма GGP лежит идея обхода графа G_m в ширину начиная с периферической вершины [169]. Чтобы разбить граф на k подмножеств, требуется выполнить порядка $\log_2 k$ его бисекций. Так как граф G_m имеет небольшую размерность, то время реализации рекурсивной бисекции с использованием алгоритма GGP незначительно и требует выполнения порядка $O(|E| \cdot \log_2 k)$ операций [148].

На третьем этапе разбиение P_i, $i = m, m-1,...,1$ проецируется на граф G_{i-1} и улучшается нетрудоемким алгоритмом BKL (Boundary Kernighan-Lin), основанным на эвристике Кернигана-Лина (Kernighan-Lin, KL) [123, 148, 169], а также ее менее трудоемкой модификации – эвристике FM (Fiduccia-Mattheyses) [123]. Трудоемкость алгоритма BKL равна $O(|E_i|)$ [148].

В листинге 2.2 приведен псевдокод алгоритма TMMGP.

Листинг 2.2. Псевдокод алгоритма TMMGP

Входные данные:	$G = (V, E)$ – информационный граф параллельной программы; N, L, n_l, n_{lk}, C, C_{lk}, c_{lk} – описание ВС ($l \in \{1, 2, \ldots, L\}$, $k \in \{1, 2, \ldots, n_l\}$).
Выходные данные:	$x[i, j]$ – вложение; $x[i, j] = 1$, если ветвь i назначена на ЭМ j, иначе $x[i, j] = 0$.

```
1   k ← [(M − 1)/c_L1] + 1
2   s ← c_L1
```

3 $M' \leftarrow \max\left\{\frac{M}{40 \cdot \log_2 k}, 20 \cdot k\right\}$

4 $m \leftarrow \log_2 \frac{M}{M'} + 1$

5 **for** $i \leftarrow 1$ **to** m **do**

6 $G_i \leftarrow CoarseGraphHEM(G_{i-1})$

7 **end for**

8 $P_m \leftarrow RecursiveBisectionGGP(G_m, k, s)$

9 **for** $i \leftarrow m$ **to** 1 **do**

10 $P_{i-1} \leftarrow ProjectPartition(G_{i-1}, P_i)$

11 $P_{i-1} \leftarrow RefinePartitionBKL(G_{i-1}, P_{i-1})$

12 **end for**

13 **for** $i \leftarrow 1$ **to** M **do**

14 $c \leftarrow Dequeue(C_{P_{0,i}})$

15 $x[i, c] \leftarrow 1$

16 **end for**

Поясним смысл основных функций, используемых в листинге 2.2. Функция $CoarseGraphHEM(G_{i-1})$ реализует сжатие графа G_{i-1} алгоритмом HEM; функция $RecursiveBisectionGGP(G_m, k, s)$ осуществляет рекурсивную бисекцию графа G_m на k подмножеств; функция $ProjectPartition(G_{i-1}, P_i)$ проецирует разбиение P_i на граф G_{i-1}; функция $RefinePartitionBKL(G_{i-1}, P_{i-1})$ улучшает разбиение P_{i-1} графа G_{i-1} алгоритмом BKL.

Рассмотрим работу алгоритма TMMGP на примере вложения параллельной MPI-программы NAS Parallel Benchmark Conjugate Gradient (NPB CG), реализующей решение системы линейных алгебраических уравнений

(СЛАУ) методом сопряженных градиентов, в вычислительный кластер. На рис. 2.7 приведен информационный граф MPI-программы NPB CG.

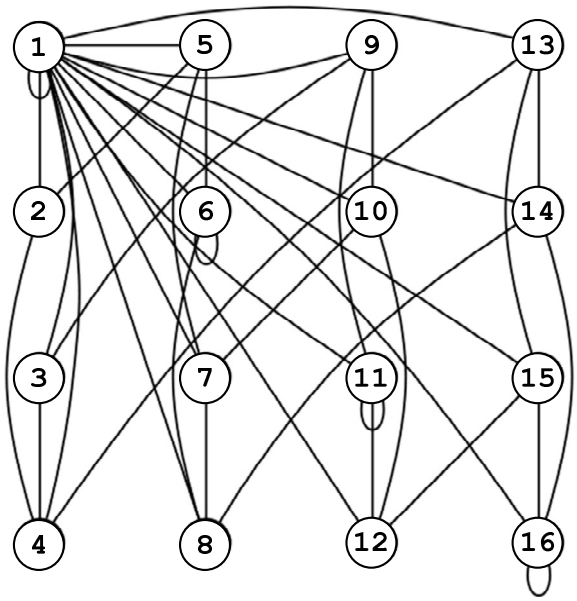


Рис. 2.7. Информационный граф параллельной программы NPB CG: $M = 16$

На рис. 2.8 приведен пример субоптимального вложения программы NPB CG в вычислительный кластер на базе двухъядерных процессоров Intel Xeon 5150. Коммуникационная среда кластера имеет иерархическую организацию. Первый уровень коммуникационной среды образован сетью связи стандарта Gigabit Ethernet, второй уровень – средствами доступа многоядерных процессоров к общей памяти. Каждый узел укомплектован парой двухъядерных процессоров.

Для графа на рис. 2.8 задача разбиения (2.8) – (2.12) решалась со следующими параметрами: $k = [(M-1)/c_{L1}] + 1 = (15/4) + 1 = 4$, $s = 4$. Сжатие и улучшение разбиений графа не осуществлялось, так как $M' = 80$ и $m < 0$. На рисунке изображен конечный результат рекурсивной бисекции графа алгоритмом GGP.

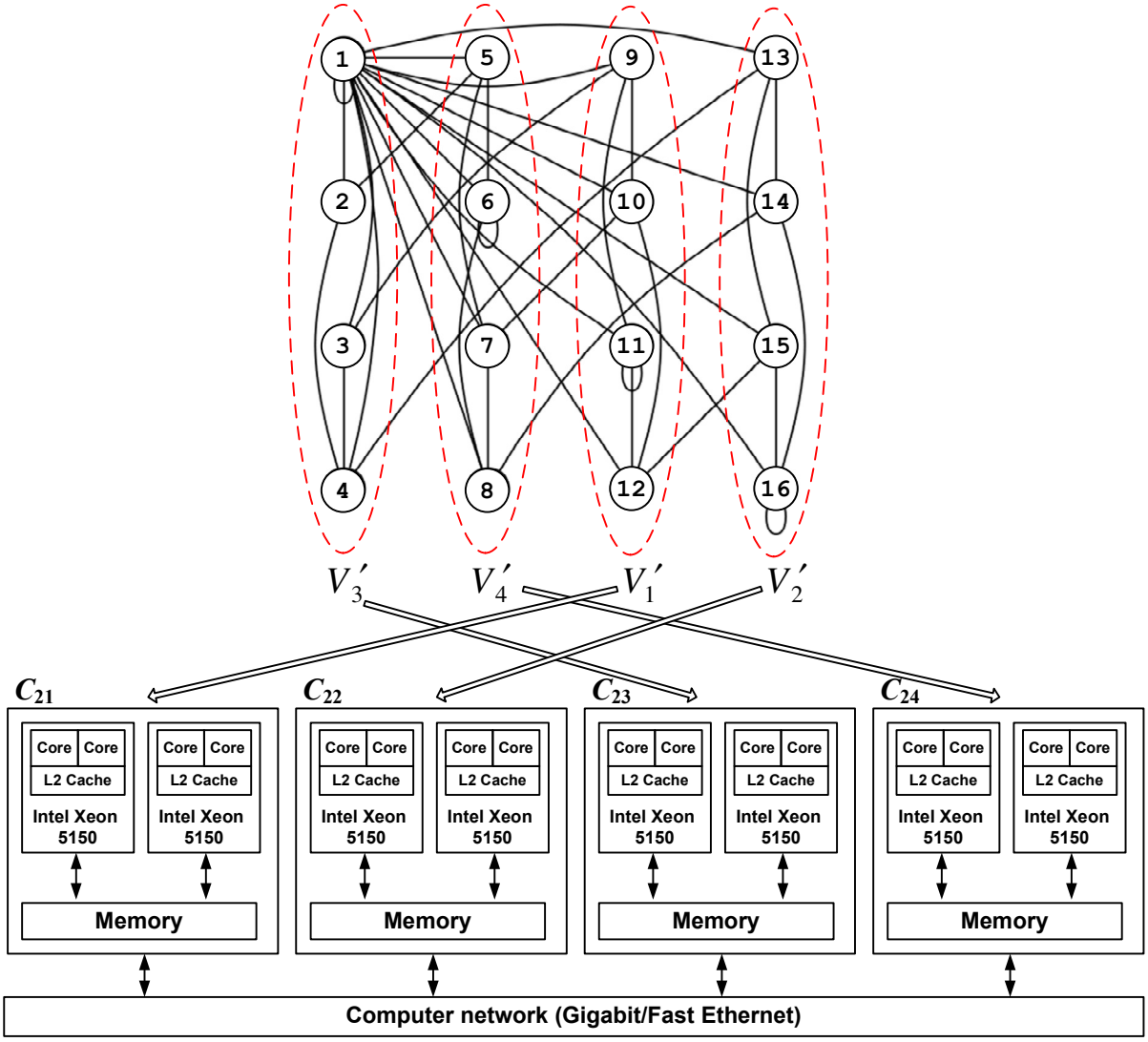


Рис. 2.8. Вложение параллельной программы в вычислительный кластер:

$M = 16; N = 16; L = 2; c_{21} = c_{22} = c_{23} = c_{24} = 4$

Результирующее вложение

$$X = \{x_{1,9} = 1, x_{2,10} = 1, x_{3,11} = 1, x_{4,12} = 1, x_{5,13} = 1, x_{6,14} = 1, x_{7,15} = 1, x_{8,16} = 1,$$
$$x_{9,1} = 1, x_{10,2} = 1, x_{11,3} = 1, x_{12,4} = 1, x_{13,5} = 1, x_{14,6} = 1, x_{15,7} = 1, x_{16,8} = 1\}.$$

Справедливо утверждение.

Утверждение 2.2. *Вычислительная сложность алгоритма TMMGP равна*

$$T_{TMMGP} = O(|E| \log_2 k).$$

Доказательство. Трудоемкость метода определяется вычислительной сложностью трех последовательно выполняемых этапов: сжатие графа, начальное разбиение и улучшение разбиений. Обозначим через T_1 трудоемкость этапа сжатия графа. Оценим ее. Для сжатия графов G_0, G_1, …, G_m требуется выполнить m + 1 раз алгоритм HEM. С переходом от графа к графу количество ребер сокращается в два раза. Для графа G_i справедливо $|E_i| = |E| / 2^i$, $i = 1,2,...,m$. Тогда трудоемкость сжатия графов (строки 5–7) равна

$$T_1 = \frac{|E|}{2^0} + \frac{|E|}{2^1} + ... + \frac{|E|}{2^m} = |E| \cdot \sum_{i=0}^{m} \frac{1}{2^i} = |E| \cdot (2 - \frac{2}{2^m}) = |E| \cdot (\frac{2M - M'}{M}),$$

$$\frac{2M - M'}{M} \le 2.$$

Следовательно,

$$T_1 = O(|E|).$$

Трудоемкость формирования начального разбиения равна $T_2 = O(|E| \log_2 k)$.

Трудоемкость проецирования и улучшения разбиений алгоритмом BKL равна $T_3 = O(|E|)$. Доказательство аналогично рассуждениям о вычислительной сложности этапа сжатия графов G_0, G_1, …, G_m.

Суммарная трудоемкость алгоритма TMMGP равна

$$T_{TMMGP} = T_1 + T_2 + T_3 = O(|E| + |E| \log_2 k + |E|) = O(|E| \log_2 k).$$

Утверждение доказано.

Результаты численного моделирования работы алгоритма TMMGP приведены в п. 4.3.

2.3. Эвристический алгоритм вложения параллельных программ в подсистемы ВС

В данном параграфе предложены эвристические алгоритмы вложения параллельных программ в подсистемы ВС с иерархической организацией коммуникационных сред. Предполагается, что все ЭМ имеют одинаковую производительность.

2.3.1. Модель подсистемы ВС с иерархической организацией коммуникационной среды

Рассмотрим модификацию математической модели коммуникационной серды ВС с иерархической организацией, описанной в п. 2.1.1. Допустим существование конфигураций подсистем, в которых элементы одного уровня могут содержать различное количество элементарных машин. Другими словами, отменим выполнение условия (2.2). Следовательно $\exists k_1, k_2 \in \{1,2,\ldots,n_L\}$:

$$c_{Lk_1} \neq c_{Lk_2}. \tag{2.13}$$

На рис. 2.9 приведен пример подсистемы с иерархической организацией. Серым цветом обозначены ЭМ недоступные для использования в данный момент, например, вследствие их выхода из работоспособного состояния или использования для решения других параллельных задач.

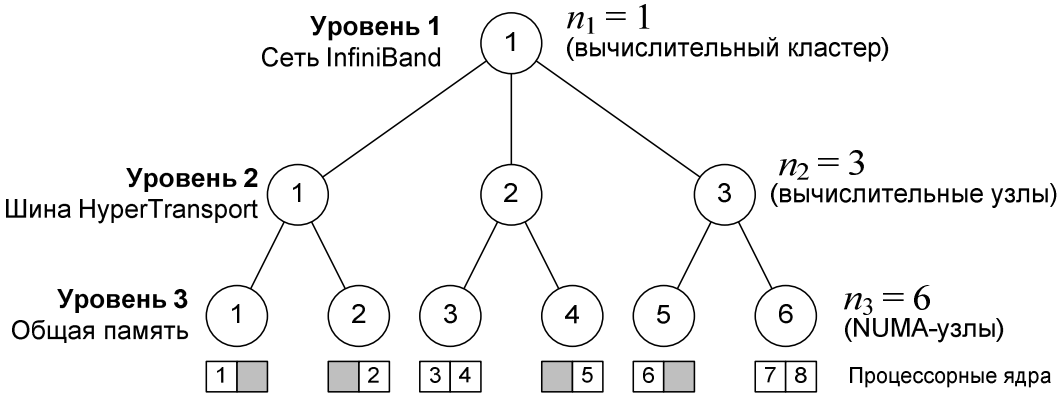


Рис. 2.9. Пример подсистемы с иерархической организацией:

$N = 8$; $L = 3$; $c_{31} = 1$; $c_{33} = 2$

Такие подсистемы могут возникать и в результате работы алгоритмов выделения ЭМ для параллельных программ. В приведенном примере видно, что $c_{31} \neq c_{33}$, следовательно, конфигурация системы удовлетворяет условию (2.13).

Предложим алгоритм поиска приближенных решений задачи (2.4) – (2.7) вложения параллельных программ в подсистемы, удовлетворяющие условию (2.13).

2.3.2. Эвристический алгоритм вложения параллельных программ в подсистему ВС

Алгоритм основан на распределении смежных ветвей параллельной программы, обменивающихся большими объемами данных, по ЭМ, которые объединены быстрыми каналами связи [48]. Входными данными алгоритма являются информационный граф $G = (V, E)$ параллельной программы и описание подсистемы.

Средняя скорость передачи данных от элементарной машины $p \in \{1, 2, \ldots, N\}$ до остальных машин подсистемы характеризуется средним геометрическим значением B_p пропускных способностей каналов связи между ней и остальными машинами. Значение B_p может быть рассчитано по определению

$$B_p = \left(\prod_{\substack{q \in C \\ q \neq p}} b_{z(p,q)} \right)^{\frac{1}{N-1}} = \exp\left(\frac{1}{N-1} \cdot \sum_{\substack{q \in C \\ q \neq p}} b_{z(p,q)} \right).$$

Оценка объемов данных, передаваемых между ветвью $i \in V$ и смежными ветвями, аналогично осуществляется через среднее геометрическое значение D_i объемов данных d_{ij}, $i, j \in \{1, 2, \ldots, M\}$, передаваемых между параллельными ветвями.

$$D_i = \left(\prod_{j \in Adj(i)} d_{ij} \right)^{\frac{1}{|Adj(i)|}} = \exp\left(\frac{1}{|Adj(i)|} \cdot \sum_{j \in Adj(i)} d_{ij} \right),$$

где $Adj(i)$ – множество ветвей смежных с ветвью *i*.

Будем считать, что информационный граф $G = (V, E)$ представлен списками смежности. Через Q_i обозначим список смежности вершины $i \in \{1, 2, \ldots, M\}$.

Опишем основные этапы работы алгоритма.

Для каждой элементарной машины $p \in \{1, 2, \ldots, N\}$ рассчитывается значение B_p. Номера элементарных машин сортируются по невозрастанию значений B_p и в таком порядке помещаются в очередь *C*.

Для упорядочивания последовательности из *n* целых чисел, лежащих в интервале [0, *k*], используется алгоритм сортировки подсчетом (Counting Sort), вычислительная сложность которого оценивается величиной $\Theta(k + n)$ [31].

Для каждой вершины $i \in \{1, 2, \ldots, M\}$ информационного графа параллельной программы рассчитывается значение D_i. Аналогично, номера ветвей сортируются по невозрастанию значений D_i и в таком порядке помещаются в очередь *Q*. По значениям D_i сортируются и вершины в списках смежности Q_i, $i = 1, 2, \ldots, M$.

Осуществляется обход вершин информационного графа $G = (V, E)$. Из очереди *Q* извлекается вершина с максимальным значением D_i, она и ее смежные ветви, взятые в порядке не возрастания значений D_i, распределяются на свободные ЭМ с максимальными значениями B_p (первые в очереди *C*). После чего из очереди *Q* извлекается очередная необработанная вершина и процедура вложения текущей и смежных с ней ветвей повторяется.

Обозначим алгоритм TMGT (Task Map Graph Traversal). В листинге 2.3 приведен его псевдокод.

Листинг 2.3. Псевдокод алгоритма TMGT

Входные данные: $G = (V, E)$ – информационный граф программы; N, L, n_l, n_{lk}, C, C_{lk}, c_{lk}, b_{pq} – описание подсистемы ($l \in \{1, 2, \ldots, L\}$, $k \in \{1, 2, \ldots, n_l\}$, $p, q \in \{1, 2, \ldots, N\}$).

Выходные данные: $x[i, j]$ – вложение; $x[i, j] = 1$, если ветвь i назначена на ЭМ j, иначе $x[i, j] = 0$.

```
1   for p ← 1 to N do                    /* Расчет компонент вектора B */
2     Enqueue(C, p)
3     B[p] ← 0
4     for q ← 1 to N do
5       if p ≠ q then
6         B[p] ← B[p] + ln(b_pq)
7     end for
8     B[p] ← exp(1 / (N – 1) * B[p])
9   end for
10  CountingSort(C)          /* Сортировка номеров ЭМ по значениям B[p] */
11  for i ← 1 to M do                    /* Расчет компонент вектора D */
12    Enqueue(Q, i)
13    mapped[i] ← 0
14    D[i] ← 0
15    for (для) каждого j ∈ Adj(i) do
16      D[i] ← D[i] + ln(d_ij)
17    end for
18    D[i] ← exp(1 / |Adj(i)| * D[i])
19  end for
```

```
20    CountingSort(Q)        /* Сортировка номеров ветвей по значениям D[i] */
21    for  i ← 1 to M do                    /* Сортировка списков смежности */
22       CountingSort(Q_i)
23    end for
24    while QueueSize(Q) > 0 do             /* Обход информационного графа */
25       i ← Dequeue(Q)
26       if mapped[i] = 0 then                      /* Если ветвь i невложена */
27          Enqueue(Q', i)
28          Enqueue(Q', Q_i)
29          while QueueSize(Q') > 0 do        /* Вложение смежных ветвей */
30             i ← Dequeue(Q')
31             if mapped[i] = 0 then
32                c ← Dequeue(C)
33                x[i, c] ← 1
34                mapped[i] ← 1
35             end if
36          end while
37       end if
38    end while
```

Рассмотрим работу алгоритма на примере вложения информационного графа (рис. 2.10) в подсистему с иерархической организацией (рис. 2.11).

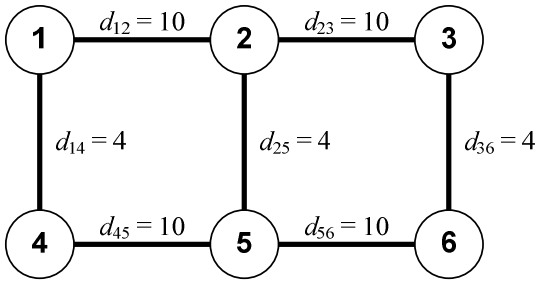


Рис. 2.10. Пример информационного графа параллельной программы:
$M = 6$; $Q_1 = (2, 4)$; $Q_2 = (1, 3, 5)$; $Q_3 = (2, 6)$; $Q_4 = (5, 1)$; $Q_5 = (4, 6, 2)$; $Q_6 = (5, 3)$

Выпишем значения $b_{pq} = b_{z(p,q)}$, $p, q \in \{1, 2, \ldots, 7\}$ для системы, приведенной на рис. 2.11.

$$\left\| b_{pq} \right\| = \begin{pmatrix} 0 & 6 & 2 & 2 & 2 & 2 & 2 \\ 6 & 0 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 0 & 8 & 2 & 2 & 2 \\ 2 & 2 & 8 & 0 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 0 & 8 & 6 \\ 2 & 2 & 2 & 2 & 8 & 0 & 6 \\ 2 & 2 & 2 & 2 & 6 & 6 & 0 \end{pmatrix} \qquad (2.14)$$

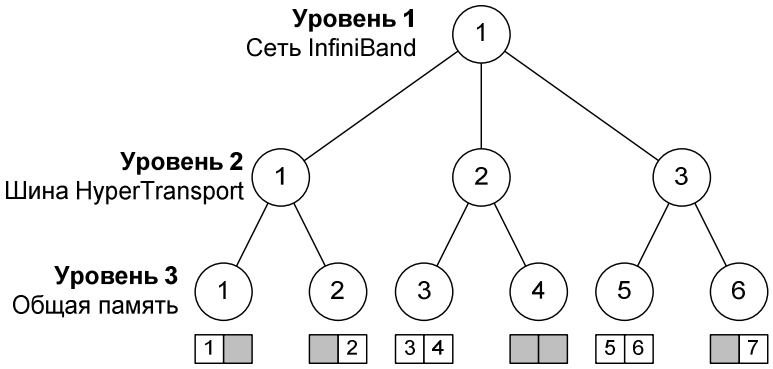


Рис. 2.11. Пример подсистемы с иерархической организацией:
$N = 7$; $L = 3$; $b_1 = 2$ ГБ/с; $b_2 = 6$ ГБ/с; $b_3 = 8$ ГБ/с

Рассчитаем, по (2.14) вектор средних геометрических значений пропускных способностей каналов связи между ЭМ подсистемы:

$$B = (2{,}40;\ 2{,}40;\ 2{,}52;\ 2{,}52;\ 3{,}03;\ 3{,}03;\ 2{,}88).$$

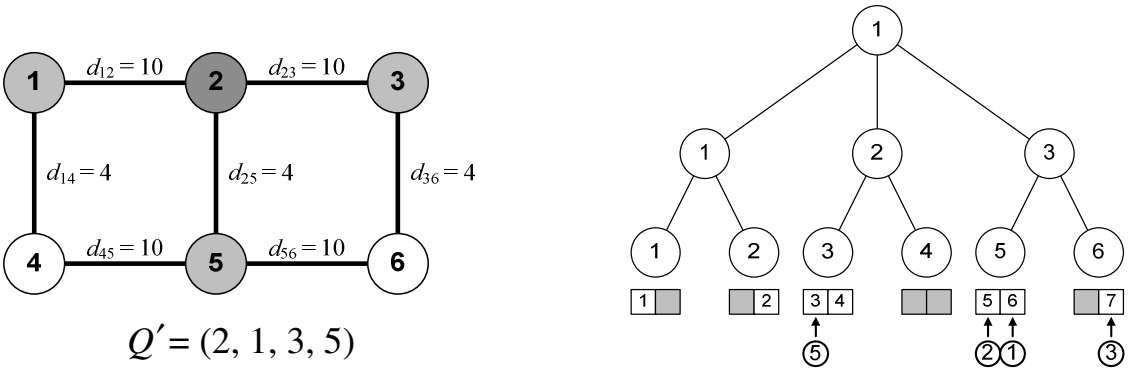
Аналогично запишем вектор D для ветвей параллельной программы:

$$D = (6{,}32;\ 7{,}37;\ 6{,}32;\ 6{,}32;\ 7{,}37;\ 6{,}32).$$

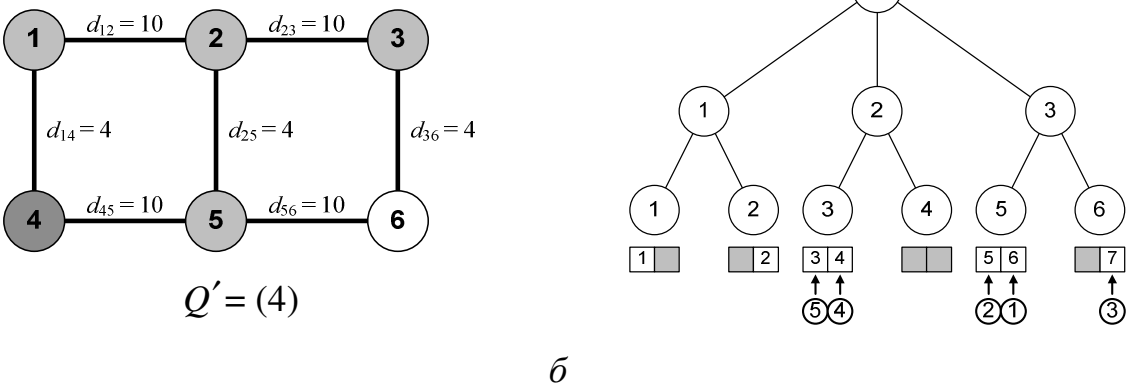
Упорядоченные списки номеров параллельных ветвей Q и элементарных машин C примут следующий вид:

$$Q = (2, 5, 1, 3, 4, 6),\ C = (5, 6, 7, 3, 4, 1, 2).$$

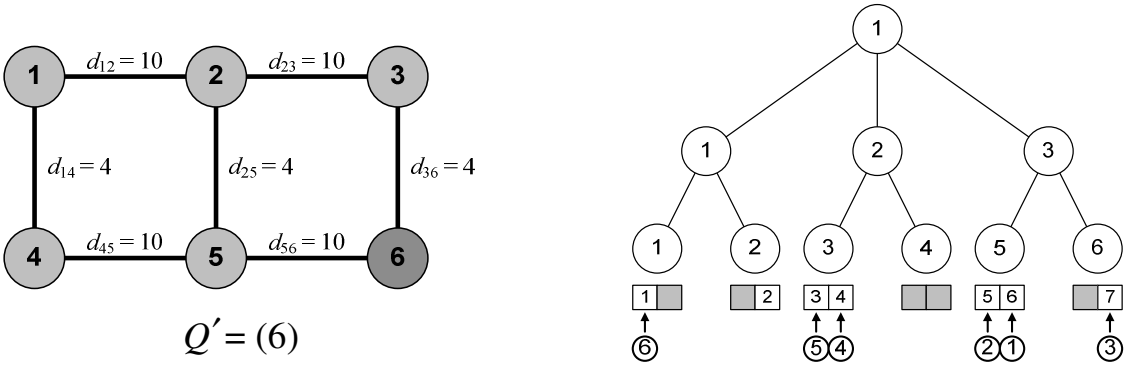
На рис. 2.12 представлены промежуточные результаты работы алгоритма между итерациями цикла 24–38. Для каждой итерации на графе серым цветом отмечены обработанные вершины. Начальная вершина итерации отмечена темно-серым цветом.



а



б



в

Рис. 2.12. Пример работы алгоритма TMGT. Обход графа (строки 24–38):
а – первая итерации; *б* – вторая итерация;
в – третья итерация

Конечный результат работы алгоритма – вложение X:

$$X = \{x_{16} = 1, x_{25} = 1, x_{37} = 1, x_{44} = 1, x_{53} = 1, x_{61} = 1\}.$$

Вычислим значение целевой функции (2.4) для построенного вложения.

$$T(X) = \max_{i \in V} \left\{ \sum_{j=1}^{M} \sum_{p=1}^{N} \sum_{q=1}^{N} x_{ip} \cdot x_{jq} \cdot d_{ij} \,/\, b_{z(p,q)} \right\},$$

$$t_1'' = \frac{13}{4}, \quad t_2'' = \frac{59}{12}, \quad t_3'' = \frac{11}{3},$$

$$t_4'' = \frac{13}{4}, \quad t_5'' = \frac{33}{4}, \quad t_6'' = 7,$$

([t_i''] = байт·с/ГБ).

$$T(X) = \max_{i \in V} \{t_i''\} = \frac{33}{4}.$$

Относительно вычислительной сложности алгоритма TMGT справедливо утверждение.

Утверждение 2.3. *Вычислительная сложность алгоритма TMGT равна*

$$T_{TMGT} = O(N^2 + M \cdot |E|).$$

Доказательство. Введем обозначения: T_1 – трудоемкость вычисления компонент вектора B (строки 1–9); T_2 – трудоемкость сортировки номеров ЭМ по значениям соответствующих компонент вектора B (строка 10); T_3 – трудоемкость расчета компонент вектора D (строки 11–19); T_4 – трудоемкость сортировки номеров параллельных ветвей (строка 20); T_5 – трудоемкость сортировки списков смежности (строки 21–23); T_6 – трудоемкость обхода информационного графа (строки 24–38).

Трудоемкость алгоритма составляет $O(T_1 + T_2 + T_3 + T_4 + T_5 + T_6)$.

Трудоемкость расчета векторов B и D определяется трудоемкостью расчета среднего геометрического значения n чисел, которая оценивается вели-

чиной $O(n)$. Поэтому $T_1 = O(N^2)$, $T_3 = O(M \cdot |E|)$. Сортировка подсчетом n неотрицательных целых чисел требует выполнения порядка $O(n)$ операций, следовательно, $T_2 = O(N)$ и $T_4 = O(M)$, $T_5 = O(M \cdot |E|)$.

В строках 24–38 реализуется обход графа, трудоемкость которого эквивалентна трудоемкости алгоритма обхода графа в ширину, представленного списками смежности [1, 31], и составляет $T_6 = O(M + |E|)$.

По условию задачи $N \geq M$, следовательно, трудоемкость алгоритма

$$T_{TMGT} = O(N^2 + N + M \cdot |E| + M + M \cdot |E| + M + |E|) = O(N^2 + M \cdot |E|)$$

Утверждение доказано.

Предложенный алгоритм TMGT имеет полиномиальную трудоемкость и позволяет строить приближенные решения задачи оптимального вложения параллельных программ в подсистемы ВС с иерархической организацией.

Как и TMMGP, алгоритм TMGT может быть рекомендован для использования в системах управления ресурсами распределенных ВС. Результаты моделирования работы алгоритма приведены в п. 4.3.

2.4. Оценка производительности ВС при реализации основных схем межмашинных обменов

Вложение в ВС параллельных программ, для которых не заданы информационные графы, осуществляется путем формирования подсистем элементарных машин и распределения по ним параллельных ветвей.

Формируемая для параллельной программы подсистема должна обеспечивать эффективную реализацию основных схем межмашинных обменов. Из теории вычислительных систем известно [27, 95], что на практике среди основных схем межмашинных обменов доминируют “коллективные” схемы: трансляционный (One-to-all Broadcast), трансляционно-циклический (All-to-all Broadcast) и коллекторный (All-to-one Broadcast) обмены. Поэтому, в условиях отсутствия данных об информационном графе параллельной про-

граммы, практически обоснованным является предположение о том, что граф программы является полным. Исходя из этого предположения, очевидно, что на время выполнения параллельной программы будут оказывать влияния производительности всех каналов межмашинных связей подсистемы.

Для оценки структурных задержек используются [95] показатели: диаметр, средний диаметр, вектор-функция структурной коммутируемости. Перечисленные показатели имеют ряд особенностей, которые затрудняют их использование для оценки производительности формируемой подсистемы по реализации основных схем межмашинных обменов.

1) Перечисленные показатели преимущественно используются для ВС со статической однородной макроструктурой и каналами межмашинных связей одинаковой производительности. Это ограничивает их применение для мультиархитектурных систем, в частности, построенных на базе коммутаторов.

2) В показателях учитываются либо предельные характеристик макроструктуры (например, в диаметре), либо среднее арифметическое значение, в котором один член может компенсировать значения остальных.

Для ВС с иерархической организацией коммуникационных сред может быть адаптирован показатель “средний диаметр”. Производительность подсистемы при реализации основных схем межмашинных обменов, можно характеризовать средним геометрическим значением кратчайших расстояний (или пропускных способностей каналов связи) между ЭМ подсистемы. Рассмотрим формальную постановку задачи формирования в пределах ВС подсистемы элементарных машин.

Пусть ВС укомплектована N однородными ЭМ, из которых n доступны для реализации ветвей параллельных программ. Обозначим через $G' = (V', E')$ макроструктуру системы – граф, отражающий связи между её ЭМ; $V' = \{1, 2, \ldots, n\}$ – множество элементарных машин; $E' \subset V' \times V'$ – множество каналов межмашинных связей; l_{pq} – кратчайшее

расстояние (в смысле теории графов) между элементарными машинами p и q ($p, q \in V'$).

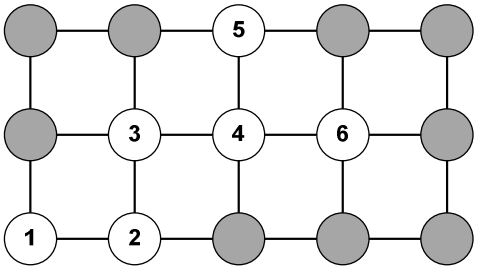
Для поступившей в систему параллельной программы требуется сформировать подсистему ранга M. Ранг подсистемы – это количество элементарных машин в ней. Через $X = (x_1, x_2, \ldots, x_n)$ обозначим вектор, задающий номера ЭМ, входящих в формируемую подсистему. Пусть $x_p = 1$, если p – ая ЭМ включена в состав подсистемы, и $x_p = 0$ в противном случае ($p \in V'$). Тогда среднее геометрическое значение $L(X)$ кратчайших расстояний между ЭМ, входящими в подсистему X, будет

$$L(X) = \left(\prod_{p=1}^{n-1} \prod_{q=p+1}^{n} (x_p x_q (l_{pq} - 1) + 1) \right)^{\frac{1}{k}}, \qquad (2.15)$$

где $k = n(n-1)/2$.

На рис. 2.13 приведен пример системы из 15 ЭМ с макроструктурой в виде двумерной решетки. Для подсистемы X = (0, 0, 1, 1, 1, 1) среднее геометрическое значение кратчайших расстояний между ЭМ будет равно

$$L(X) = \sqrt[6]{l_{34} \cdot l_{35} \cdot l_{36} \cdot l_{45} \cdot l_{46} \cdot l_{56}} = 1{,}41.$$



$$\left\| l_{pq} \right\| = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 4 \\ 1 & 0 & 1 & 2 & 3 & 3 \\ 2 & 1 & 0 & 1 & 2 & 2 \\ 3 & 2 & 1 & 0 & 1 & 1 \\ 5 & 3 & 2 & 1 & 0 & 2 \\ 4 & 3 & 2 & 1 & 2 & 0 \end{pmatrix}$$

Рис. 2.13. Пример макроструктуры ВС:

$N = 15$; $n = 6$

Сформулируем задачу формирования оптимальной подсистемы, в смысле показателя (2.15).

$$L(X) = \left(\prod_{p=1}^{n-1} \prod_{q=p+1}^{n} (x_p x_q (l_{pq} - 1) + 1) \right)^{\frac{1}{k}} \rightarrow \min_{(x_i)} \tag{2.16}$$

при ограничениях:

$$\sum_{i=1}^{n} x_i = M\,, \tag{2.17}$$

$$x_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n. \tag{2.18}$$

Задача (2.16) – (2.18) относится к дискретной оптимизации. Рассмотрим алгоритмы ее приближенного решения.

2.5. Алгоритмы формирования подсистем ВС

Для решения задачи (2.16) – (2.18) предложено два алгоритма [59]. Опишем первый из них.

Алгоритм начинает работу с поиска начальной элементарной машины, обладающей максимальным средним геометрическим значением кратчайших расстояний от неё до остальных машин системы. Заметим, что в системе может быть несколько элементарных машин p, обладающих равными максимальными значениями L_p

$$L_p = \left(\prod_{\substack{q=1, \\ q \neq p}}^{n} l_{pq} \right)^{\frac{1}{n-1}}.$$

Начальная машина включается в состав формируемой подсистемы. После чего в подсистему последовательно включаются ЭМ, ближайшие к подсистеме по значению показателя (2.15).

Обозначим описанный алгоритм PAGS (Processor Allocation Growing Subsystem). В листинге 2.4 приведен его псевдокод.

Листинг 2.4. Псевдокод алгоритма PAGS

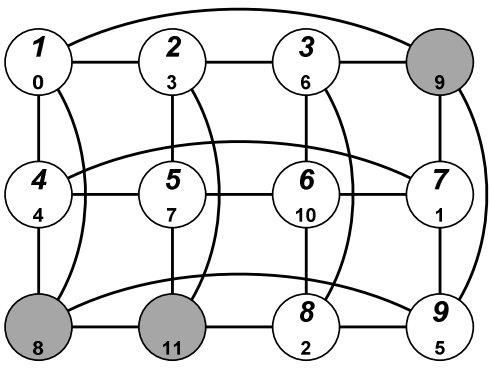
Входные данные: n – количество ЭМ в системе; M – ранг формируемой подсистемы; l_{pq} – матрица кратчайших расстояний между элементарными машинами ($p, q \in \{1, 2, \ldots, n\}$).

Выходные данные: $x[p]$ – сформированная подсистема; $x[p] = 1$, если элементарная машина p включена в состав подсистемы, иначе $x[p] = 0$.

```
1    L_min ← ∞
2    for p ← 1 to n do                          /* Поиск начальной ЭМ */
3       L ← 0
4       for q ← 1 to n do
5          if p ≠ q then
6             L ← L + ln(l_pq)
7          end if
8       end for
9       L ← exp(1 / (n – 1) * L)
10      if L < L_min then
11         v ← p
12         L_min ← L
13      end if
14   end for
15   k ← 0
16   for p ← 1 to n do
17      l[p] ← 1
18   end for
19   repeat                                     /* Формирование подсистемы */
```

```
20      x[v] ← 1
21      k ← k + 1
22      for p ← 1 to n d        /* Корректировка расстояний до подсистемы */
23        if x[p] = 0 then
24          l[p] ← l[p] * l_pv
25      end for
26      L_min ← ∞                       /* Поиск ЭМ ближайшей к подсистеме */
27      for  p ← 1 to n do
28        if l[p] < L_min and x[p] = 0 then
29          v ← p
30          L_min ← l[p]
31        end if
32      end for
33    until k ≠ M
```

Рассмотрим пример работы алгоритма PAGS на примере формирования подсистемы ранга 4 в ВС с макроструктурой D_2-графа. Система изображена на рис 2.14. Серым цветом обозначены ЭМ недоступные для использования. Курсивом указаны номера ЭМ в рамках подсистемы, оригинальные адреса ЭМ приведены ниже.



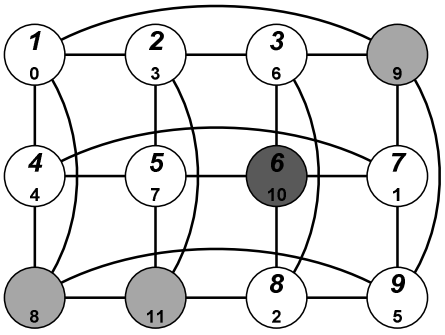
$$l_{pq} = \begin{pmatrix} 0 & 1 & 2 & 1 & 2 & 3 & 2 & 3 & 3 \\ 1 & 0 & 1 & 2 & 1 & 2 & 3 & 2 & 3 \\ 2 & 1 & 0 & 3 & 2 & 1 & 2 & 1 & 2 \\ 1 & 2 & 3 & 0 & 1 & 2 & 1 & 3 & 2 \\ 2 & 1 & 2 & 1 & 0 & 1 & 2 & 2 & 3 \\ 3 & 2 & 1 & 2 & 1 & 0 & 1 & 1 & 2 \\ 2 & 3 & 2 & 1 & 2 & 1 & 0 & 2 & 1 \\ 3 & 2 & 1 & 3 & 2 & 1 & 2 & 0 & 1 \\ 3 & 3 & 2 & 2 & 3 & 2 & 1 & 1 & 0 \end{pmatrix}$$

Рис. 2.14. Макроструктура системы – D_2-граф: {12; 3, 4}:
$N = 12$; $n = 9$

В приведенном примере в качестве начальной будет выбрана машина № 6, так как ей соответствует минимальное значение $L_6 = 1{,}49$.

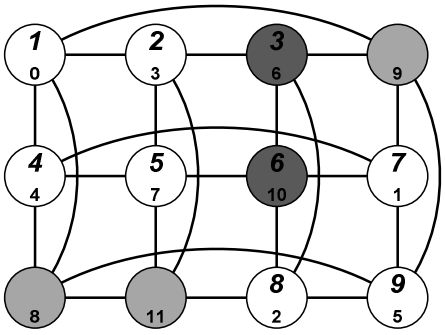
$$L_1 = 1{,}96;\ \ L_2 = 1{,}71;\ \ L_3 = 1{,}62;\ \ L_4 = 1{,}71;\ \ L_5 = 1{,}62;$$
$$L_6 = 1{,}49;\ \ L_7 = 1{,}62;\ \ L_8 = 1{,}71;\ \ L_9 = 1{,}96.$$

На рис. 2.15 изображены этапы формирования подсистемы.



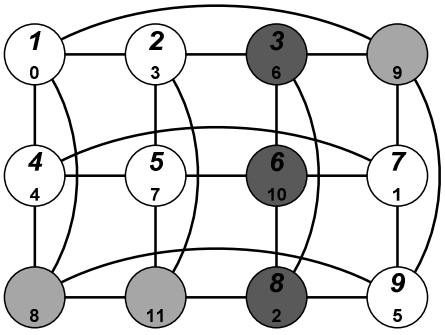
$l_1 = 3;\ \ l_2 = 2;\ \ l_3 = 1;\ \ l_4 = 2;$
$l_5 = 1;\ \ l_7 = 1;\ \ l_8 = 1;\ \ l_9 = 2$

а



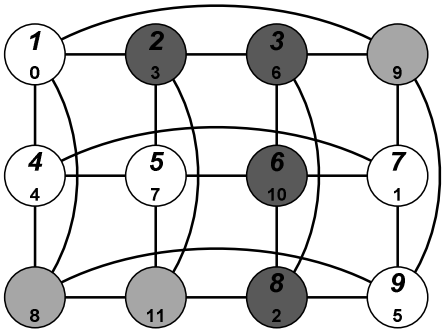
$l_1 = 6;\ \ l_2 = 2;\ \ l_4 = 6;$
$l_5 = 2;\ \ l_7 = 2;\ \ l_8 = 1;\ \ l_9 = 4$

б



$l_1 = 18;\ \ l_2 = 4;\ \ l_4 = 18;$
$l_5 = 4;\ \ l_7 = 4;\ \ l_9 = 4$

в



$l_1 = 18;\ \ l_4 = 36;$
$l_5 = 4;\ \ l_7 = 12;\ \ l_9 = 12$

г

Рис. 2.15. Формирование подсистемы алгоритмом PAGS:

а – включение начальной ЭМ в подсистему;

б – включение второй ЭМ; *в* – включение третьей ЭМ;

г – включение четвертой ЭМ

В результате работы алгоритма сформирована подсистема

$$X = (0, 1, 1, 0, 0, 1, 0, 1, 0),\ \ L(X) = \left(l_{23} \cdot l_{26} \cdot l_{28} \cdot l_{36} \cdot l_{38} \cdot l_{68}\right)^{\frac{1}{6}} = 1{,}26.$$

Относительно трудоемкости алгоритма PAGS справедливо утверждение.

Утверждение 2.4. *Вычислительная сложность алгоритма PAGS равна*

$$T_{PAGS} = O(n^2).$$

Доказательство. Трудоемкость алгоритма определяется трудоемкостью поиска начальной ЭМ (строки 1–14), которая равна $O(n^2)$. Инициализация массива l требует выполнения порядка $O(n)$ операций. Операции цикла в строках 19–33 выполняются M раз – по разу для каждой ЭМ. Трудоемкость выполнения одной итерации цикла равна $O(n)$. Суммарная трудоемкость алгоритма равна $T_{PAGS} = O(n^2 + n + M \cdot n)$. По условию задачи $M \le n$, следовательно, $T_{PAGS} = O(n^2)$.

Утверждение доказано.

При формировании подсистемы элементарных машин может возникнуть ситуация, когда добавление очередной машины приведет к увеличению компонент связности в макростурктуре ВС. Например, на рис. 2.16 приведена макроструктура в виде двумерной решетки. Темно-серым цветом обозначены машины недоступные для использования (инцидентные им каналы связи, также недоступны для коммуникаций), серым цветом обозначены ЭМ, включенные в формируемую подсистему.

Машины с номерами 6, 7, 8, 9 являются точками сочленения макроструктуры. Включение любой из них в подсистему приведет к потере связности графа свободных ЭМ. Это, в свою очередь, приведет к “фрагментации макроструктуры” и снизит вероятность успешного формирования подсистем заданного ранга.

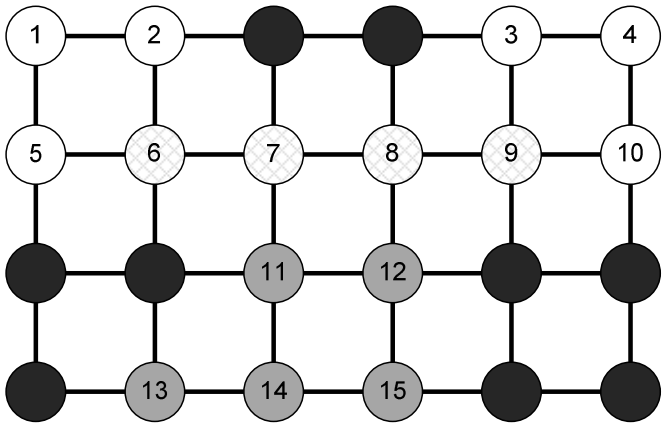


Рис. 2.16. Пример макроструктуры ВС

Для предотвращения фрагментации макроструктуры предложен модифицированный алгоритм PAGCS (Processor Allocation Growing Connected Subsystem). В алгоритм добавлена процедура поиска в графе точек сочленения и их исключения из списка кандидатов на включение в подсистему. В листинге 2.5 приведен псевдокод алгоритма PAGCS.

Листинг 2.5. Псевдокод алгоритма PAGCS

Входные данные: n – количество ЭМ в системе; M – ранг формируемой подсистемы; l_{pq} – матрица кратчайших расстояний между элементарными машинами ($p, q \in \{1, 2, \ldots, n\}$).

Выходные данные: $x[p]$ – сформированная подсистема; $x[p] = 1$, если элементарная машина p включена в состав подсистемы, иначе $x[p] = 0$.

```
1    a ← SearchArticulationPoints(G′, x)        /* Поиск точек сочленения */
2    L_min ← ∞
3    for p ← 1 to n do                          /* Поиск начальной ЭМ */
4       L ← 0
5       for q ← 1 to n do
6          if p ≠ q then
```

```
7              L ← L + ln(l_pq)
8            end if
9          end for
10         L ← exp(1 / (n – 1) * L)
11         if L < L_min then
12           v ← p
13           L_min ← L
14         end if
15       end for
16       k ← 0
17       for p ← 1 to n do
18         l[p] ← 1
19       end for
20       repeat
21         x[v] ← 1
22         k ← k + 1
23         for p ← 1 to n do                    /* Корректировка расстояний */
24           if x[p] = 0 then
25             l[p] ← l[p] * l_pv
26         end for
27         a ← SearchArticulationPoints(G′, x)      /* Поиск точек сочленения */
28         L_min ← ∞                          /* Поиск ЭМ ближайшей к подсистеме */
29         for  p ← 1 to n do
30           if l[p] < L_min and x[p] = 0 and a[p] = 0 then
31             v ← p
32             L_min ← l[p]
33           end if
34         end for
35         if  L_min = ∞ then
```

36 **error** *“Невозможно сформировать подсистему”*

37 **until** $k \neq M$

Функция *SearchArticulationPoints*(G', x) осуществляет поиск точек сочленения среди свободных ЭМ в графе $G' = (V', E')$. Функция возвращает массив a с информацией о том, какие вершины графа являются точками сочленения: $a[p] = 1$, если, элементарная машина p является точкой сочленения, иначе $a[p] = 0$.

Точки сочленения отыскиваются модифицированным алгоритмом обхода графа в глубину [1]. Трудоемкость функции *SearchArticulationPoints*(G', x) равна $O(\max\{n, |E'|\})$ (если граф G' представлен списками смежности) [31].

Относительно вычислительной сложности алгоритма PAGCS справедливо утверждение.

Утверждение 2.5. *Вычислительная сложность алгоритма PAGCS равна*

$$T_{PAGCS} = O(n^2 + n \cdot \max\{n, |E'|\}).$$

Доказательство. Доказательство аналогично доказательству утверждения 2.4. Отличие заключается в трудоемкости цикла в строках 20–37, которая составляет $O(n \cdot \max\{n, |E'|\})$ и определяется трудоемкостью поиска точек сочленения в графе G'.

Утверждение доказано.

Стоить заметить, что предложенный алгоритм PAGS может быть успешно применен для формирования подсистем в мультиархитектурных ВС, например, в вычислительных кластерах. Для этого в качестве значений l_{pq} следует брать значения пропускных способностей каналов связи между элементарными машинами и решать задачу (2.16) – (2.18) на максимум. Результаты моделирования алгоритмов представлены в п. 4.4.

2.6. Выводы

1. Описана математическая модель коммуникационной среды ВС с иерархической организацией. Сформулирована задача оптимального вложения параллельных программ в такие системы.

2. Предложен метод TMGP и алгоритм TMMGP формирования субоптимальных вложений параллельных программ в ВС с иерархической организацией коммуникационных сред. Вложение строится путем разбиения информационного графа программы на подмножества с параллельными ветвями, обменивающиеся большими объемами данных, и их распределения на элементарные машины, между которыми имеются быстрые каналы связи.

3. Для вложения параллельных программ в подсистемы ВС с иерархической организацией разработан эвристический алгоритм TMGT, который обладает полиномиальной трудоемкостью.

4. Вложение в ВС параллельных программ, для которых не заданы информационные графы, осуществляется эвристическими алгоритмами формирования подсистем, допускающими эффективную реализацию основных схем межмашинных обменов.

Разработанные алгоритмы ориентированы на использование с средствах управления ресурсами распределенных ВС и поддержки мультипрограммных режимов их функционирования. Алгоритмы TMMGP и TMGT рекомендованы для оптимизации вложения параллельных программ в библиотеках MPI и формирования виртуальных топологий (функции MPI_Graph_create, MPI_Cart_create).

Алгоритмы PAGS и PAGCS могут быть использованы в составе систем пакетной обработки заданий (например, TORQUE, OpenPBS и др.) для формирования подсистем ЭМ.

ГЛАВА 3. АЛГОРИТМЫ ВЛОЖЕНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ В ПРОСТРАНСТВЕННО-РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

В этой главе рассматриваются математическая модель коммуникационных сред пространственно-распределенных ВС и задача оптимального вложения параллельных программ в такие системы. Предложены стохастические последовательный и параллельный алгоритмы поиска субоптимальных решений задачи.

Описаны алгоритмы формирования в пределах распределенных ВС однородных подсистем и вложения в них параллельных программ.

3.1. Оптимизация вложения параллельных программ в пространственно-распределенные ВС

Рассмотрим задачу оптимального вложения параллельных программ в пространственно-распределенные ВС. Такие системы – это макроколлективы пространственно рассредоточенных ВС, взаимодействующих через локальные и глобальные сети связи (включая сеть Интернет).

3.1.1. Модель коммуникационной среды пространственно-распределенной ВС

Пусть пространственно-распределенная ВС укомплектована N элементарными машинами. Коммуникационная среда системы может быть представлена в виде дерева, содержащего L уровней (рис. 3.1). Высоту дерева обозначим через h. Каждый уровень системы образован отдельным видом структурных элементов системы (например, пространственно-рассредоточенные вычислительные системы, телекоммуникационные шкафы, вычислительные узлы и т. п.), которые объединены каналами связи своего уровня. На уровне l размещено n_l элементов, $l \in \{1, 2, \ldots, L\}$. Для каж-

дого элемента на уровне l задано множество N_{lk} его прямых дочерних узлов, $k \in \{1, 2, \ldots, n_l\}$. Количество дочерних узлов обозначим через $n_{lk} = |N_{lk}|$.

Пусть $C = \{1, 2, \ldots, N\}$ – это множество ЭМ входящих в систему; C_{lk} – множество ЭМ, принадлежащих потомкам элемента k на уровне l. Очевидно, что $C_{11} = C$. Положим $c_{lk} = |C_{lk}|$. Подмножества $C_{21}, C_{22}, \ldots, C_{2n_2}$ – это пространственно-распределенные вычислительные подсистемы. Обозначим через $\omega_i = \omega_{2i}$ производительность ЭМ в подсистеме C_{2i} ($i \in \{1, 2, \ldots, n_2\}$, $[\omega_{2i}]$ = FLOPS). Аналогично, ω_{lk} – это производительность ЭМ в подмножестве C_{lk}. Далее считаем $H = n_2$.

На структуру дерева наложены следующие ограничения $\forall l \in \{1,2,\ldots,L\}$, $\forall k \in \{1,2,\ldots,n_l\}$, $\forall k_1, k_2 \in N_{lk}$:

$$n_{lk_1} = n_{lk_2}, \tag{3.1}$$

$$c_{lk_1} = c_{lk_2}. \tag{3.2}$$

Первое ограничение (3.1) гарантирует, что элементы одного уровня (определенной подсистемы) имеют одинаковое количество дочерних узлов. Второе (3.2) обеспечивает равенство числа ЭМ принадлежащих потомкам элементов одного уровня.

Каждый уровень $l \in \{1, 2, \ldots, L\}$ коммуникационной среды характеризуется своими значениями показателей производительности: $b(l, k_1, k_2, m)$ – зависимость пропускной способности канала связи между элементами k_1 и k_2 на уровне l от размера m передаваемого сообщения ($[b(l, k_1, k_2, m)]$ = байт/с); $l(l, k_1, k_2, m)$ – зависимость латентности канала связи между элементами k_1 и k_2 на уровне l от размера m передаваемого сообщения ($[l(l, k_1, k_2, m)]$ = с). Аналогично, $b(l, k_1, k_2)$ – максимальное значение пропускной способности канала связи между элементами k_1 и k_2 на уровне l. Определим функции

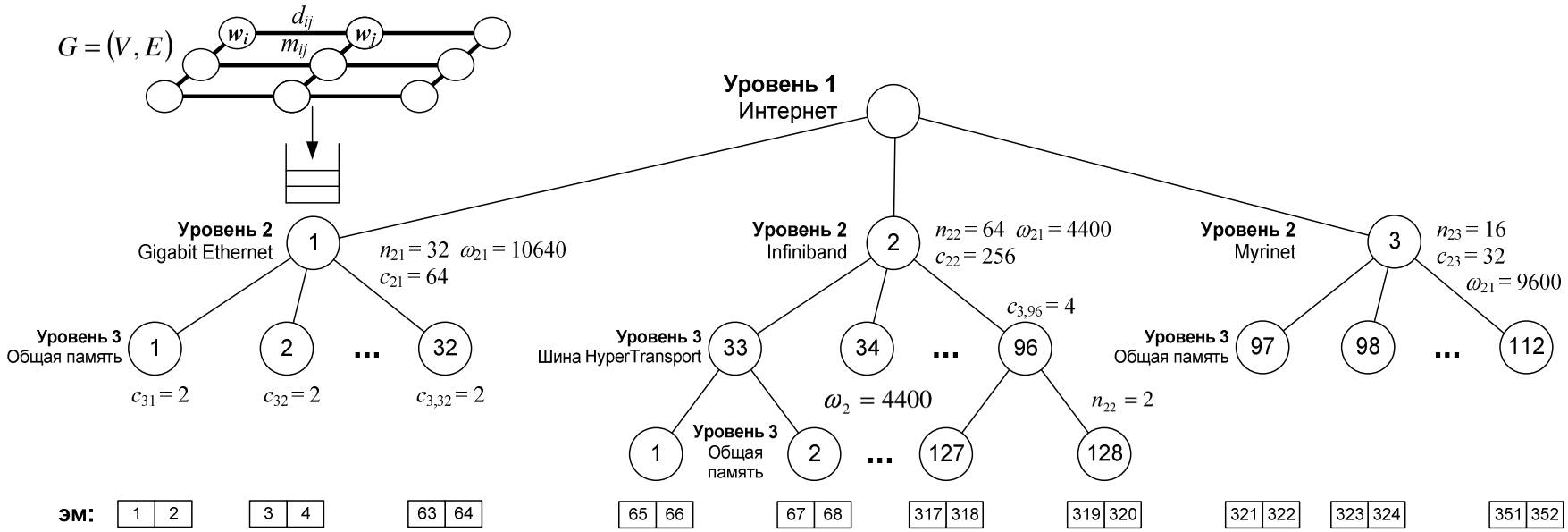


Рис. 3.1. Пространственно-распределенная ВС с иерархической организацией:

$N = 352$; $H = 3$; $L = 3$; $N_{22} = \{33, 34, \ldots, 96\}$

$b(l_1, k_1, l_2, k_2)$ пропускной способности канала связи между элементом k_1 на уровне l_1 и элементом k_2 на уровне l_2. Предполагается, что в системе $\forall k_1, k_2$

$$b(1, k_1, k_2) \le b(2, k_1, k_2) \le \ldots \le b(L, k_1, k_2).$$

3.1.2. Организации выполнения параллельных программ на пространственно-распределенной ВС

Положим, что процедура вложения параллельной программы осуществляется с подсистемы $s \in \{1,2,...,H\}$ и поступившая на выполнение параллельная программа представлена информационным графом $G = (V, E)$, где $V = \{1, 2, \ldots, M\}$ – множество параллельных ветвей программы; $E \subseteq V \times V$ – множество информационно-логических связей между её параллельными ветвями. Будем полагать: w_i – количество элементарных операций (арифметических и логических) выполняемых ветвью $i \in V$; d_{ij} – объем данных передаваемый по ребру $(i, j) \in E$ за время выполнения параллельной программы ($[d_{ij}]$= байт, $i, j \in V$); m_{ij} – средний размер сообщения передаваемого по ребру $(i, j) \in E$ за одну операцию приема/передачи ($[m_{ij}]$= байт, $i, j \in V$); z – размер файла и входных данных параллельной программы ($[z]$ = байт).

Требуется построить инъективную функцию $f : V \to C$, ставящую в соответствие ветвям параллельной программы элементарные машины распределенной ВС. Требуется найти x_{ij}:

$$X = \{x_{ij} : i \in V, j \in C\},\ x_{ij} = \begin{cases} 1, & \text{если } f(i) = j; \\ 0 & \text{иначе.} \end{cases}$$

Качество вложения оценивается ожидаемым временем T обслуживания параллельной программы, которое складывается из времени доставки программы на ЭМ и времени выполнения ветвей на них.

Время τ доставки программы на ЭМ распределенной ВС определяется максимальным из времен передачи файла программы до подсистем, элементарные машины которых назначены для выполнения её ветвей:

$$\tau = \max_{i \in V}\{\tau_i\} = \max_{i \in V}\left\{\sum_{p=1}^{N} x_{ip} \cdot t(s, g(p), z)\right\},$$

где τ_i – время доставки программы с подсистемы $s \in \{1,2,...,H\}$ на подсистему, элементарная машина которой выделена для выполнения параллельной ветви i; $g(p) \in \{1,2,...,H\}$ – номер подсистемы, которой принадлежит элементарная машина $p \in C$.

Для оценки времени передачи данных по каналам связи между подсистемами используется модель Хокни (R. Hockney), в которой время передачи сообщения размером z байт между двумя подсистемами выражается как

$$t(k_1, k_2, z) = l(1, k_1, k_2, z) + z/b(1, k_1, k_2, z).$$

Считаем, что после доставки файла и данных программы на элементарные машины осуществляется одновременный запуск всех параллельных ветвей на выполнение.

Время t_i выполнения ветви $i \in V$ параллельной программы складывается из времени выполнения арифметических и логических операций элементарной машиной и времени взаимодействия со смежными ветвями. Тогда ожидаемое время выполнения параллельной программы есть

$$\max_{i \in V}\{t_i\} = \max_{i \in V}\{t_i' + t_i''\},$$

где $t_i' = \sum_{p=1}^{N} x_{ip} \cdot \frac{w_i}{\omega_{g(p)}}$ – время выполнения арифметических и логических операций элементарной машиной, на которую назначена ветвь, $t_i'' = \sum_{j=1}^{M}\sum_{p=1}^{N}\sum_{q=1}^{N} x_{ip} \cdot x_{jq} \cdot t(i, j, p, q)$ – время взаимодействия со смежными ветвями,

$t(i, j, p, q)$ – время взаимодействия между ветвями $i, j \in V$, назначенными на элементарные машины p и q, соответственно ($p, q \in C$).

$$t(i, j, p, q) = l(l, k_1, k_2, m_{ij}) \cdot d_{ij} / m_{ij} + d_{ij} / b(l, k_1, k_2, m_{ij}),$$

где $l = u(p, q)$, $k_1 = p(l, p)$, $k_2 = p(l, q)$. Функция $u(p, q)$ ставит в соответствие номерам ЭМ $p, q \in C$ номер уровня коммуникационной среды, являющегося ближайшим общим предком для них. Функция $p(l, c)$ ставит в соответствие элементарной машине $c \in C$ номер родительского подмножества C_{lk} на уровне *l*.

3.1.3. Задача оптимального вложения параллельных программ в распределенную ВС

Учитывая инъективность функции $f : V \to C$ сформулируем задачу оптимального вложения в пространственно-распределенную ВС параллельной программы с целью минимизации времени ее выполнения.

$$T(X) = \max_{i \in V}\{\tau_i\} + \max_{i \in V}\{t_i\} =$$

$$= \max_{i \in V}\left\{\sum_{p=1}^{N} x_{ip} \cdot t(s, g(p), z)\right\} +$$

$$+ \max_{i \in V}\left\{\sum_{p=1}^{N} x_{ip} \cdot \frac{w_i}{\omega_{g(p)}} + \sum_{j=1}^{M}\sum_{p=1}^{N}\sum_{q=1}^{N} x_{ip} \cdot x_{jq} \cdot t(i, j, p, q)\right\} \to \min_{(x_{ij})} \qquad (3.3)$$

при ограничениях:

$$\sum_{j=1}^{N} x_{ij} = 1, \quad i = 1, 2, \ldots, M, \qquad (3.4)$$

$$\sum_{i=1}^{M} x_{ij} \le 1, \quad j = 1, 2, \ldots, N, \qquad (3.5)$$

$$x_{ij} \in \{0,1\}, \quad i \in V, j \in C. \tag{3.6}$$

Ограничения (3.4), (3.6) гарантируют назначение каждой ветви параллельной программы на единственную ЭМ, ограничения (3.5) обеспечивают назначение на ЭМ не более одной ветви. Обозначим через D множество вложений, удовлетворяющих условиям (3.4) – (3.6). Предложим алгоритмы приближенного решения задачи (3.3) – (3.6).

3.2. Стохастический алгоритм вложения параллельных программ в пространственно-распределенные ВС

На основе метаэвристики имитации отжига (Simulated Annealing) [9, 64, 142, 152, 155] разработан [45, 49, 96, 151] стохастический алгоритм, позволяющий отыскивать приближенные решения задачи вложения параллельных программ в пространственно-распределенные ВС. Для краткости будем называть его TMSA (Task Map Simulated Annealing). Общая схема алгоритма выглядит следующим образом. На первом шаге алгоритма строится начальное допустимое решение $x^{(0)} \in D$. На k-ом шаге имеется текущее решение $x^{(k)} \in D$ и наилучшее на данный момент $x^* \in D$. Шаг начинается с выбора случайным образом соседнего решения $y \in D$ из окрестности $N(x^{(k)})$ текущего. Если значение целевой функции (3.1) от соседнего решения меньше значения целевой функции от текущего, то соседнее решение принимается за текущее. Кроме того, даже если соседнее решение по целевой функции превосходит текущее, то с вероятностью

$$P(x^{(k)}, y, k) = \begin{cases} 1, & \text{если } T(y) \le T(x); \\ \exp\left((T(x) - T(y))/c_k\right) & \text{иначе.} \end{cases}$$

соседнее решение принимается за текущее. Данный шаг позволяет алгоритму имитации отжига не “застревать” в локальных экстремумах и исследовать большее количество допустимых решений.

3.2.1. Последовательный алгоритм вложения

Опишем основные операции алгоритма TMSA. Представим решение задачи в виде вектора $x = (x_1, x_2, ..., x_M)$, где $x_i \in \{1,2,..., N\}$ – номер ЭМ, на которую назначена ветвь $i \in V$. Начальное решение $x^{(0)} \in D$ строится по следующему правилу.

Правило 3.1. Подсистемы C_{2i} распределенной ВС сортируются в порядке невозрастания количества c_{2i} ЭМ в них, после чего ветви параллельной программы последовательно назначаются на ЭМ, начиная с самой большой подсистемы.

Решение из окрестности текущего $y \in N(x)$ выбирается по следующему правилу.

Правило 3.2. Формируется равномерно распределенное псевдослучайное целое число $\xi_1 \in [0, N-1]$. Компоненты вектора решения $x = (x_1, x_2, ..., x_M)$ циклически сдвигаются на значение ξ_1 так, что

$$x_i := \begin{cases} x_i + \xi_1, & \text{если } x_i + \xi_1 \le N; \\ (x_i + \xi_1) \bmod N & \text{иначе.} \end{cases}, \quad i = 1,2,...M .$$

После чего вектор решения в случайной позиции $\xi_2 \in [1, M-1]$ делится на две части $[1, \xi_2]$ и $[\xi_2 + 1, M]$, которые переставляются местами.

Последовательность $\{c_k\}$ выбирается так, чтобы $c_k \to 0$ при $k \to \infty$. Это обеспечивает на начальных этапах работы алгоритма принятие с большей вероятностью решений со значениями целевой функции большими, чем текущее. В алгоритме TMSA последовательность $\{c_k\}$ строится следующим образом:

$$c_k = \alpha / (k+1) \cdot \beta ,$$

$$\alpha = ((c_0 - c_R) \cdot (R+1))/R,$$

$$\beta = c_0 - \alpha,$$

где R – номер последнего элемента последовательности. В листинге 3.1 приведен псевдокод алгоритма TMSA.

Листинг 3.1. Псевдокод алгоритма TMSA

Входные данные: $G = (V, E)$ – информационный граф параллельной программы; N, H, L, n_l, n_{lk}, N_{lk}, $N_{l,k,u}$, C, C_{lk}, c_{lk}, $b(l_1, k_1, l_2, k_2)$ – описание пространственно-распределенной ВС ($l \in \{1, 2, \ldots, L\}$, $k \in \{1, 2, \ldots, n_l\}$); $c_{\max}$, $c_{\min}$, R, K – параметры алгоритма.

Выходные данные: $x[i, j]$ – вложение; $x[i, j] = 1$, если ветвь i назначена на ЭМ j, иначе $x[i, j] = 0$.

```
1   α ← (c_max − c_min) · (R+1)/R
2   β ← c_max − α
3   x ← InitialSolution()              /* Формирование начального решения */
4   x* ← x
5   c ← c_max
6   k ← 0
7   while c > c_min do
8     d ← 0
9     while d < K do
10      y ← N(x)                       /* Выбор решения из окрестности */
11      if U(0,1) < P(x, y, k) then
12          x′ ← y
13      if F(y) < F(x*) then
14          x* ← y
```

15 $d \leftarrow d + 1$

16 **end while**

17 $x \leftarrow x'$

18 $k \leftarrow k + 1$

19 $c \leftarrow \alpha/(k+1) \cdot \beta$

20 **end while**

Поясним смысл некоторых этапов алгоритма. В строке 11 используется функция $U(n, m)$, которая возвращает равномерно распределенное псевдослучайное число из интервала $[n, m]$. Функция *InitialSolution* формирует начальное решение по правилу 3.1. Выбор решения из окрестности текущего (функция N в строке 10) осуществляется по правилу 3.2.

Относительно вычислительной сложности алгоритма TMSA справедливо утверждение.

Утверждение 3.1. *Вычислительная сложность алгоритма TMSA равна*

$$T_{TMSA} = O(\max\{M + H, R \cdot K \cdot M^2\}).$$

Доказательство. Трудоемкость формирования начального решения по правилу 3.1 равна $O(M + H)$. Строки 10 – 15 внутри цикла выполняются $R \cdot K$ раз. Вычислительная сложность выбора решения из окрестности текущего равна $O(M)$. Трудоемкость вычисления целевой функции равна $O(M^2)$. Суммарная трудоемкость алгоритма составляет

$$T_{TMSA} = O(M + H + R \cdot K \cdot (M + M^2)) = O(\max\{M + H, R \cdot K \cdot M^2\}).$$

Утверждение доказано.

3.2.2. Параллельный алгоритм вложения

Для вложения параллельных программ с большим количеством M ветвей предложен параллельный алгоритм TMPSA (Task Map Parallel Simulated Annealing). Для построения алгоритма использована методика крупноблоч-

ного распараллеливания [27, 34, 68, 95]. Итерации цикла (строки 9 – 16) алгоритма TMSA реализуются одновременно на разных ЭМ. Обозначим за n количество параллельных ветвей алгоритма. Выделим корневую ветвь (ветвь с номером 0). Рабочие ветви получают решение от корневой ветви и выполняют $M/(n-1)$ итераций цикла (строки 10 – 15), после чего отправляют корневой ветви значение целевой функции лучшего найденного решения. Корневая ветвь определяет наилучшее найденное решение, делает его текущим и итерации цикла по значениям последовательности $\{c_k\}$ продолжаются. За *rank* обозначим номер текущей ветви.

В листинге 3.2 приведен псевдокод корневой ветви алгоритма TMPSA, а в листинге 3.3 – псевдокод рабочих ветвей.

Листинг 3.2. Псевдокод корневой ветви алгоритма TMPSA

Входные данные: $G = (V, E)$ – информационный граф параллельной программы; N, H, L, n_l, n_{lk}, N_{lk}, $N_{l,k,u}$, C, C_{lk}, c_{lk}, $b(l_1, k_1, l_2, k_2)$ – описание пространственно-распределенной ВС ($l \in \{1, 2, \ldots, L\}$, $k \in \{1, 2, \ldots, n_l\}$); $c_{\max}$, $c_{\min}$, R, K – параметры алгоритма; n – количество параллельных ветвей алгоритма.

Выходные данные: $x[i, j]$ – вложение; $x[i, j] = 1$, если ветвь i назначена на ЭМ j, иначе $x[i, j] = 0$.

```
1   α ← (c_max − c_min) · (R + 1)/R
2   β ← c_max − α
3   x ← InitialSolution()              /* Формирование начального решения */
4   x* ← x
5   c ← c_max
6   k ← 0
```

7 **while** $c > c_{min}$ **do**

8 $Send(x; 1, 2, \ldots, n-1)$

9 $(F_1, F_2, \ldots, F_{n-1}) \leftarrow Receive(1, 2, \ldots, n-1)$

10 $r \leftarrow \arg\min_{i=\overline{1,n-1}}\{F_i\}$

11 $x' \leftarrow Receive(r)$

12 $x^* \leftarrow x'$

13 $k \leftarrow k + 1$

14 $x \leftarrow x'$

15 $k \leftarrow k + 1$

16 $c \leftarrow \alpha/(k+1) \cdot \beta$

17 **end while**

Поясним смысл функций *Send*, *Receive*. Функция $Send(x; p_1, p_2, \ldots, p_n)$ осуществляет трансляционную передачу вектора x элементарным машинам с номерами $p_1, p_2, \ldots, p_n$. Функция $Receive(p_1, p_2, \ldots, p_n)$ осуществляет коллекторный прием и возвращает вектор данных, принятых от ЭМ $p_1, p_2, \ldots, p_n$.

Листинг 3.3. Псевдокод параллельных ветвей алгоритма TMPSA

Входные данные: $G = (V, E)$ – информационный граф параллельной программы; N, H, L, n_l, n_{lk}, N_{lk}, $N_{l,k,u}$, C, C_{lk}, c_{lk}, $b(l_1, k_1, l_2, k_2)$ – описание пространственно-распределенной ВС ($l \in \{1, 2, \ldots, L\}$, $k \in \{1, 2, \ldots, n_l\}$); $c_{\max}$, $c_{\min}$, R, K – параметры алгоритма; n – количество параллельных ветвей алгоритма; *rank* – номер ветви.

Выходные данные: $x[i, j]$ – вложение; $x[i, j] = 1$, если ветвь i назначена на ЭМ j, иначе $x[i, j] = 0$.

1 $\alpha \leftarrow (c_{max} - c_{min}) \cdot (R+1)/R$

2 $\beta \leftarrow c_{max} - \alpha$

3 $c \leftarrow c_{max}$

4 $k \leftarrow 0$

5 **while** $c > c_{min}$ **do**

6 $x \leftarrow Receive(0)$

7 $x^* \leftarrow x$

8 $d \leftarrow 0$

9 **while** $d \le \frac{M}{n-1}$ **do**

10 $y \leftarrow N(x)$ ** Выбор решения из окрестности **

11 **if** $U(0,1) < P(x, y, k)$ **then**

12 $x' \leftarrow y$

13 **if** $F(y) < F(x^*)$ **then**

14 $x^* \leftarrow y$

15 $d \leftarrow d + 1$

16 **end while**

17 $Send(F(x^*); 0, 1, \ldots, n-1)$

18 $(F_1, F_2, \ldots, F_{n-1}) \leftarrow Receive(1, 2, \ldots, n-1)$

19 $r \leftarrow \arg\min_{i=\overline{1,n-1}}\{F_i\}$

20 **if** $r = rank$ **then**

21 $Send(x^*; 0)$

22 $k \leftarrow k + 1$

23 $c \leftarrow \alpha/(k+1) \cdot \beta$

24 **end while**

Алгоритмы TMSA и TMPSA предназначены для использования в составе систем управления ресурсами большемасштабных распределенных ВС

(GRID-диспетчерах), особенно на гетерогенных конфигурациях (например, мультикластерных ВС и GRID-системах), а также при решении сложных параллельных задач. Результаты моделирования алгоритмов приведены в п. 4.3.

3.3. Алгоритм вложения параллельных программ в подсистемы пространственно-распределенных ВС

Рассмотрим алгоритм вложения параллельных программ в подсистемы пространственно-распределенных ВС. Пусть имеется подсистема пространственно-распределенной ВС, в которой элементы одного уровня могут содержать различное количество элементарных машин, т. е. допустимо существование подсистем, таких, что

$$\exists k_1, k_2 \in \{1,2,\ldots,n_L\} \;\; c_{Lk_1} \neq c_{Lk_2}. \tag{3.7}$$

Введем обозначения (рис. 3.2): h_{lk} – высота поддерева с корнем в узле k на уровне l иерархии системы; $N_{l,k,u}$ – множество дочерних узлов на уровне u элемента k с уровня l ($u < l$) (рис. 3.2); B_{lk} и $\overline{B}_{lk}$ – показатели, характеризующие производительности коммуникационных сред между элементарными машинами в подмножествах C_{lk}.

$$B_{lk} = (b_1, b_2, ..., b_{h_{lk}+1}), \quad \overline{B}_{lk} = \left(\prod_{i=1}^{h_{lk}+1} b_i \right)^{\frac{1}{h_{lk}+1}},$$

$b_i = \min\limits_{p,q \in N_{l,k,l+i}} \{b(l+i, p, l+i, q)\}$ – минимальное значение пропускной способности каналов связи на уровне $l + i$ между дочерними элементами узла k с уровня l; $b(l_1, k_1, l_2, k_2)$ – пропускная способность канала связи между элементом k_1 на уровне l_1 и элементом k_2 с уровня l_2.

На рис. 3.2 в качестве минимальных пропускных способностей каналов связи взяты следующие значения: Fast Ethernet – 25 МБ/с;

Gigabit Ethernet – 125 МБ/с; InfiniBand – 2000 МБ/с; Myrinet – 1000 МБ/с; Shared Memory – 8000 МБ/с. Компоненты векторов B_{lk} приведены в МБ/с.

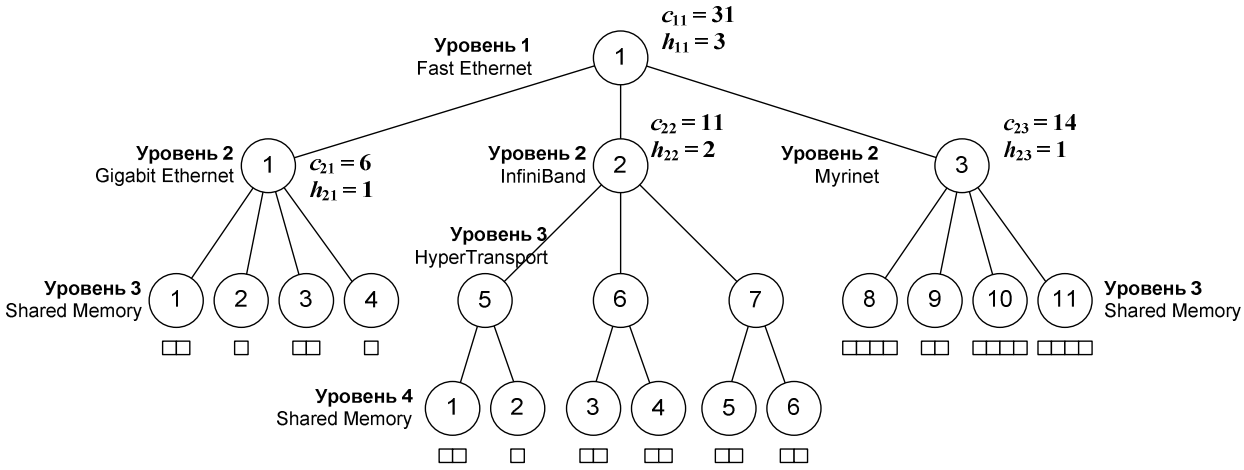


Рис. 3.2. Подсистема пространственно-распределенной ВС:

$B_{21} = (125; 8000)$; $B_{22} = (2000; 6000; 8000)$; $B_{23} = (1000; 8000)$;

$\overline{B}_{21} = 1000$; $\overline{B}_{22} = 4579$; $\overline{B}_{23} = 2828$; $b(2,1,4,3) = 25$; $b(4,5,3,6) = 6000$;

$N_{2,1,3} = \{1, 2, 3, 4\}$; $N_{3,6,4} = \{3, 4\}$

Требуется субоптимально вложить параллельную программу, представленную информационным графом $G = (V, E)$, в подсистему пространственно-распределенной ВС.

В качестве показателя эффективности вложения X используется ожидаемое время $T(X)$ выполнения параллельной программы (п. 3.1.3).

$$T(X) = \max_{i \in V}\{t_i\} = \\ = \max_{i \in V}\left\{\sum_{p=1}^{N} x_{ip} \cdot \frac{w_i}{\omega_{g(p)}} + \sum_{j=1}^{M}\sum_{p=1}^{N}\sum_{q=1}^{N} x_{ip} \cdot x_{jq} \cdot t(i, j, p, q)\right\}, \tag{3.8}$$

где t_i – это время выполнения i-ой параллельной ветви.

Требуется найти x_{ij}, доставляющие минимум функции (3.8) и удовлетворяющие ограничениям:

$$\sum_{j=1}^{N} x_{ij} = 1, \quad i = 1, 2, \ldots, M, \tag{3.9}$$

$$\sum_{i=1}^{M} x_{ij} \le 1, \quad j = 1, 2, \ldots, N, \tag{3.10}$$

$$x_{ij} \in \{0,1\}, \quad i \in V, j \in C. \tag{3.11}$$

Предложим алгоритм приближенного решения задачи (3.8) – (3.11).

Алгоритм. Обозначим алгоритм TMDS (Task Map Distributed Subsystem). Работа алгоритма состоит из двух основных этапов: 1) формирование подсистемы ранга M из элементарных машин ВС; 2) обход информационного графа задачи и распределение параллельных ветвей по ЭМ, сформированной подсистем.

Формирование подсистемы ранга M осуществляется последовательным включением подмножеств C_{lk} в подсистему. На практике для параллельной программы может быть задано или определено путем анализа её информационного графа, какая конфигурация подсистемы предпочтительна для задачи в смысле показателя (3.8). Например, для программ с большим значением коэффициента накладных расходов эффективными являются подсистемы с быстрыми каналами межмашинных связей, а для программ с незначительными значениями коэффициента накладных расходов в первую очередь следует учитывать производительность ЭМ, включаемых в подсистему.

В алгоритм TMDS введен параметр $\delta \in \{0,1\}$, который позволяет пользователю приоритезировать параметры выбора подмножеств C_{lk}, включаемых в формируемую подсистему. Если $\delta = 0$, то при выборе очередного подмножества C_{lk} в первую очередь учитывается производительность ЭМ в нем, а уже затем – производительности каналов межмашинных связей. В случае $\delta = 1$, в первую очередь учитываются производительности каналов межмашинных связей в подмножестве C_{lk}.

Обозначим через s количество подмножеств C_{lk}, включенных в состав подсистемы; L_i и K_i – это, соответственно, номер уровня и подмножества C_{lk}, включенного в подсистему под номером $i \in \{1, 2, \ldots, s\}$. Например, $L_3 = 3$, $K_3 = 4$ означает, что третьим подмножеством в подсистему включено C_{34}.

Работа алгоритма начинается с упорядочивания подмножеств $C_{21}, C_{22}, \ldots, C_{2H}$ по невозрастанию значений $U(2,k,\delta)$, $k \in \{1, 2, \ldots, H\}$.

$$U(l,k,\delta) = \begin{cases} \omega_{lk}, & \text{если } \delta = 0, \\ W(l,k) & \text{иначе} \end{cases},$$

$$W(l,k) = \begin{cases} \overline{B}_{lk}, & \text{если } s = 0, \\ \sqrt{\left(\prod_{j=1}^{s} b(L_j, K_j, l, k)\right)^{\frac{1}{s}} \cdot \overline{B}_{lk}} & \text{иначе} \end{cases}.$$

Среди первых подряд идущих подмножеств с совпадающими значениями $U(2,k,\delta)$ отыскивается подмножество с максимальным значением $U(2,k,1-\delta)$. Функция $W(l, k)$ характеризует производительности каналов связи между ЭМ в подмножестве C_{lk} и среднее значение производительности каналов связи от подсистемы до подмножества C_{lk}.

Если сумма количества c_{2k} ЭМ в найденном подмножестве C_{2k} и в подсистеме меньше M, то подмножество включается в подсистему и обработка текущего уровня продолжается. Если после включения очередного подмножества в подсистему, сумма $r + c_{2k} > M$, то осуществляется аналогичный просмотр дочерних подмножеств C_{3i}, $i \in N_{2k}$ (r – количество ЭМ включенных в подсистему). Рекурсивный поиск среди дочерних подмножеств продолжается до тех пор, пока сумма $r + c_{lk}$ не будет меньше или равна M (или не кончатся дочерние элементы).

В результате работы алгоритма в состав подсистемы будет включено s подмножеств $C_{L_1K_1}$, $C_{L_2K_2}$, …, $C_{L_sK_s}$. Первые M элементарных машин, взятые из подмножеств в порядке их включения в подсистему, помещаются в очередь Q.

На втором этапе работы алгоритма осуществляется обход вершин информационного графа параллельной программы. Предполагается, что граф представлен списками смежности. Стартовой выбирается вершина i с максимальным средним геометрическим значением D_i объемов данных, передаваемых между ветвью i и смежными с ней ветвями.

$$D_i = \left(\prod_{j \in Adj(i)} d_{ij} \right)^{\frac{1}{|Adj(i)|}}.$$

Ветвь i назначается на первую в очереди Q элементарную машину. Затем в порядке неубывания значений d_{ij} посещаются смежные с i вершины, которые вкладываются в ЭМ, следующие в очереди. Среди невложенных ветвей отыскивается ветвь с максимальным значением D_i и процедура повторяется. В листинге 3.4 приведен псевдокод алгоритма.

Листинг 3.4. Псевдокод алгоритма TMDS

Входные данные: $G = (V, E)$ – информационный граф параллельной программы; N, H, L, n_l, n_{lk}, N_{lk}, $N_{l,k,u}$, C, C_{lk}, c_{lk}, h_{lk}, B_{lk}, $\overline{B}_{lk}$, $b(l_1,k_1,l_2,k_2)$ – описание пространственно-распределенной ВС ($l \in \{1, 2, \ldots, L\}$, $k \in \{1, 2, \ldots, n_l\}$);

Выходные данные: $x[i, j]$ – вложение; $x[i, j] = 1$, если ветвь i назначена на ЭМ j, иначе $x[i, j] = 0$.

```
1   r ← 0        /* Счетчик количества ЭМ, включенных в подсистему */
2   s ← 0        /* Количество подмножеств, включенных в подсистему */
```

```
3   ProcessNode(1, 1)
4   TraverseTaskGraph()

ProcessNode(l, k)          /* Обрабатывает дочерние узлы элемента (l,k) */
1   ComputeB(l, k)
2   while r < M do
3     i ← FindNode(l, k)
4     if c_{l+1,i} + r > M then
5       if n_{li} > 0 then
6         ProcessNode(l + 1, i)
7       else
8         AddNodeToSubsystem(k, l + 1, i)
9       end if
10    else
11      AddNodeToSubsystem(k, l + 1, i)
12    end if
13  end while

ComputeB(l, k)   /* Расчет расстояний от подмножеств до подсистемы */
1   for (для) каждого i ∈ N_{lk} do
2     B_i ← 1          /* Расстояние от подмножества i до подсистемы */
3     for j ← 1 to s do
4       B_i ← B_i · b(L_j, K_j, l+1, i)
5     end for
6   end for

AddNodeToSubsystem(p, l, k)          /* Включает C_{lk} в подсистему */
1   r ← r + c_{lk}
```

2 $s \leftarrow s + 1$

3 $L[s] \leftarrow l$

4 $K[s] \leftarrow k$

5 $used[l, k] = 1$

6 **for** (для) каждого $i \in N_{l-1,p}$ **do**

7 $B_i \leftarrow B_i \cdot b(L_s, K_s, l, i)$

8 **end for**

FindNode(l, k)

1 **if** $\delta = 0$ **then**

2 $(k_1, k_2, \ldots, k_n) \leftarrow CountingSort(\omega_{l+1,j})$ */* Сортировка подсистем */*

3 $B_{max} \leftarrow 0$

4 $i \leftarrow 1$

5 **while** $\omega_{l+1,i} = \omega_{l+1,j_1}$ **do**

6 **if** $used[l + 1, i] = 0$ **then**

7 $B \leftarrow \overline{B}_{l+1,i}$

8 **if** $s > 0$ **then**

9 $B \leftarrow \sqrt{B \cdot (B_i)^{\frac{1}{s}}}$

10 **if** $B_{max} < B$ **then**

11 $B_{max} \leftarrow B$

12 $i_{max} \leftarrow i$

13 **end if**

14 **end if**

15 $i \leftarrow i + 1$

16 **end while**

17 **else**

18 **if** $s > 0$ **then**

```
19          (k_1, k_2, ..., k_n) ← CountingSort(√(B̄_{l+1,j} · (B_j)^{1/s}))
20      else
21          (k_1, k_2, ..., k_n) ← CountingSort(B̄_{l+1,j})
22      end if
23      ω_max ← 0
24      i ← 1
25      while W(l+1,i) = W(l+1, j_1) do
26          if used[l + 1, i] = 0 then
27              if ω_max < ω_i then
28                  ω_max ← ω_i
29                  i_max ← i
30              end if
31          end if
32          i ← i + 1
33      end while
34  end if
35  return i_max
```

```
TraverseTaskGraph()                    /* Обход информационного графа */
 1  for i ← 1 to s do
 2      Enqueue(C, C_{L_s K_s})
 3  end for
 4  for i ← 1 to M do                  /* Расчет компонент вектора D */
 5      Enqueue(Q, i)
 6      mapped[i] ← 0
 7      D[i] ← 0
 8      for (для) каждого j ∈ Adj(i) do
```

```
 9          D[i] ← D[i] + ln(d_ij)
10        end for
11        D[i] ← exp(1 / |Adj(i)| * D[i])
12    end for
13    CountingSort(Q)        /* Сортировка номеров ветвей по значениям D[i] */
14    for  i ← 1 to M do                    /* Сортировка списков смежности */
15        CountingSort(Q_i)
16    end for
17    while QueueSize(Q) > 0 do
18        i ← Dequeue(Q)
19        if mapped[i] = 0 then                    /* Если ветвь i не вложена */
20            Enqueue(Q', i)
21            Enqueue(Q', Q_i)
22            while QueueSize(Q') > 0 do          /* Вложение смежных ветвей */
23                i ← Dequeue(Q')
24                if mapped[i] = 0 then
25                    c ← Dequeue(C)
26                    x[i, c] ← 1
27                    mapped[i] ← 1
28                end if
29            end while
30        end if
31    end while
```

Рассмотрим пример вложения параллельной программы, представленной информационным графом на рис. 3.3, в подсистему пространственно-распределенной ВС (рис. 3.4) для случая $\delta = 0$.

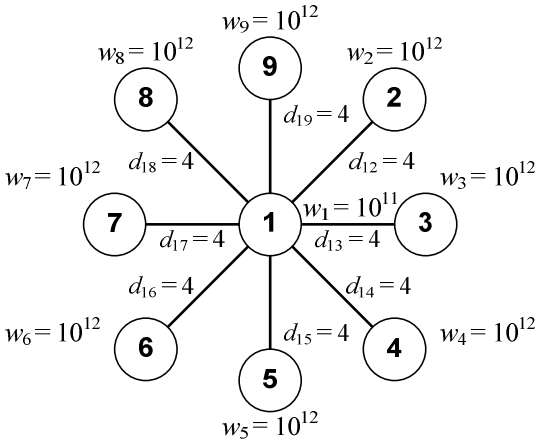
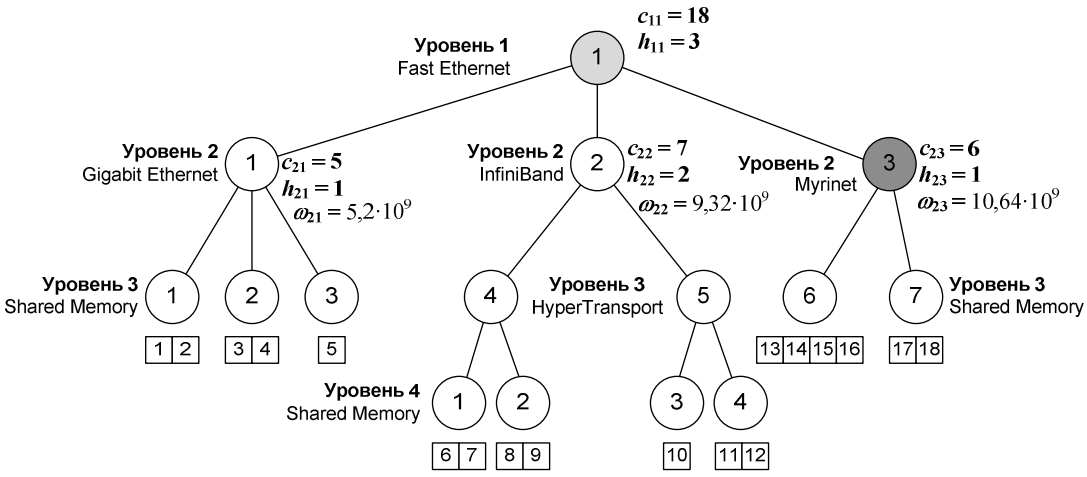
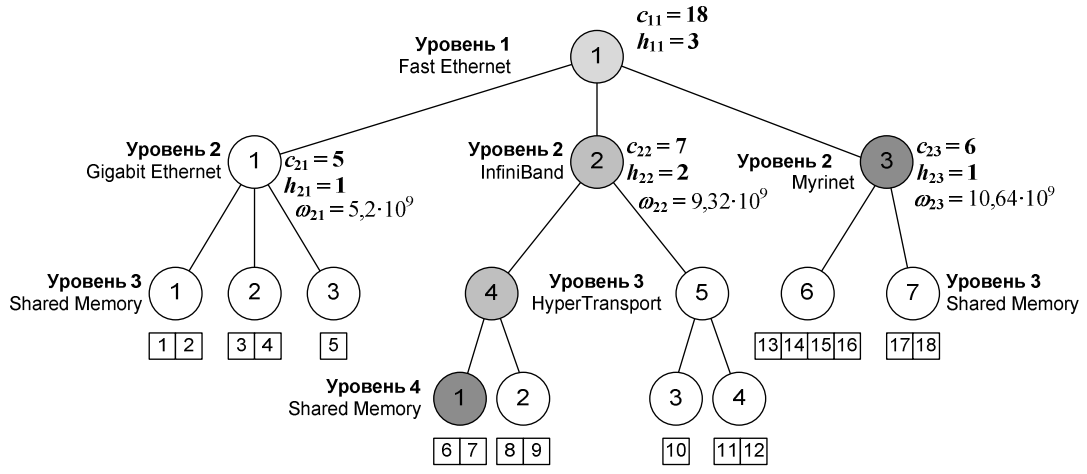


Рис. 3.3. Информационный граф параллельной программы



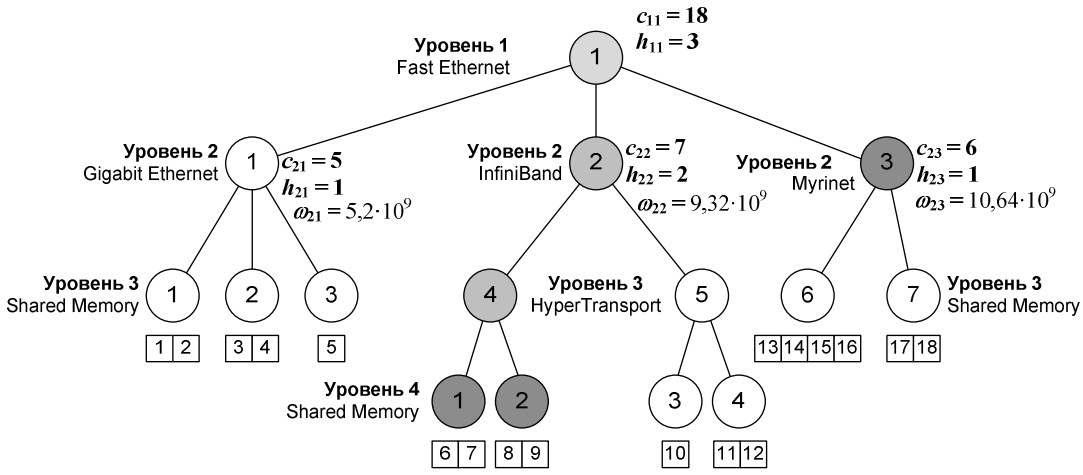
$r = 6$

а



$r = 8$

б



$r = 10$

в

Рис. 3.4. Пример работы алгоритма TMDS:

а – включение первого подмножества;

б – включение второго подмножества;

в – включение третьего подмножества

В соответствии с алгоритмом вершины информационного графа обработаны в следующем порядке: 1, 2, 3, 4, 5, 6, 7, 8, 9. Результирующее вложение:

$$X = \{x_{1,13} = 1;\ x_{2,14} = 1;\ x_{3,15} = 1;\ x_{4,16} = 1;\ x_{5,17} = 1;$$
$$x_{6,18} = 1;\ x_{7,6} = 1;\ x_{8,7} = 1;\ x_{9,8} = 1\}$$

Вычислим значение целевой функции (3.8).

$$T(X) = \max_{i=1,9}\{t_i\} = \max_{i \in V}\left\{\sum_{p=1}^{18} x_{ip} \cdot \frac{w_i}{\omega_{g_p}} + \sum_{j=1}^{9}\sum_{p=1}^{18}\sum_{q=1}^{18} x_{ip} \cdot x_{jq} \cdot t(i, j, p, q)\right\}.$$

$$t_1 = \frac{10^{11}}{10{,}64 \cdot 10^9} + 3 \cdot \frac{4}{8000} + 2 \cdot \frac{4}{1000} + 3 \cdot \frac{4}{25} = 9{,}89;$$

$$t_2 = t_3 = t_4 = \frac{10^{12}}{10{,}64 \cdot 10^9} + \frac{4}{8000} = 94;$$

$$t_5 = t_6 = \frac{10^{12}}{10{,}64 \cdot 10^9} + \frac{4}{1000} = 94;$$

$$t_7 = t_8 = t_9 = \frac{10^{12}}{9{,}32 \cdot 10^9} + \frac{4}{25} = 107{,}46;$$

$$T(X) = \max_{i=1,M}\{t_i\} = 107{,}46.$$

Относительно вычислительной сложности алгоритма TMDS справедливо утверждение.

Утверждение 3.2. *Вычислительная сложность алгоритма TMDS равна*

$$T_{TMDS} = O(\max\{hN^2, M + |E|\}).$$

Доказательство. В ходе работы алгоритма исследуются подмножества C_{lk} на разных уровнях дерева коммуникационной среды системы. В худшем случае возможна следующая конфигурация ВС (рис. 3.5). В состав подсистемы включаются элементарные машины из $H - 1$ подмножеств C_{21}, C_{22}, …, $C_{2,H-1}$. В подмножестве C_{2H} элементарных машин больше, чем требуется. Осуществляется переход на уровень 3 к дочерним подмножествам элемента C_{2H}. Таких подмножеств n_{2H}. Аналогично, элементарные машины из $n_{2H} - 1$ подмножества на уровне 3 включаются в подсистему, и осуществляется переход на следующий уровень к элементам подмножества C_{3,n_3}. Спуск по дереву продолжается до тех пор, пока система не будет сформирована. В худшем случае требуется просмотреть все h уровней в иерархии системы.

Для исследования подмножеств одного уровня требуется выполнить следующие операции: 1) рассчитать элементы B_i (функция *ComputeB*); 2) выполнить n_{lk} раз поиск очередного подмножества для включения его в подсистему (функция *FindNode*); 3) добавить найденное подмножество в подсистему (функция *AddNodeToSubsystem*).

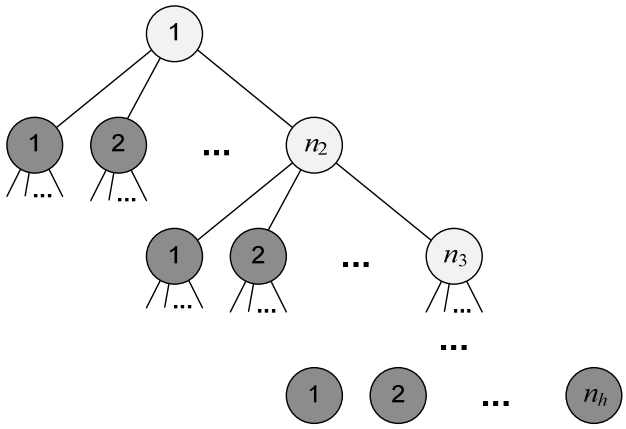


Рис. 3.5. Возможная конфигурация пространственно-распределенной ВС

Трудоемкость функции *ComputeB* равна $O(N^2)$, так как возможна ситуация, когда $n_{lk} = N$ и в состав подсистемы включено N подмножеств (если каждое подмножество содержит по одной ЭМ). Аналогично, трудоемкость функции *FindNode* составляет $O(N)$. Добавление подмножества в подсистему (функция *AddNodeToSubsystem*) требует выполнения $O(N)$ операций. Следовательно, для обработки одного уровня необходимо выполнить порядка $O(N^2 + N + N) = O(N^2)$ операций. Для обработки h уровней требуется выполнить $O(h \cdot N^2)$ операций. Оценим трудоемкость второго этапа алгоритма. Трудоемкость формирования очереди Q равна $O(N)$. Вычислительная сложность расчета компонент вектора D равна $O(M^2)$. Сортировка элементов очереди Q требует выполнения $O(N)$ операций, а списков смежности – $O(M^2)$. Трудоемкость обхода информационного графа равна $O(M + |E|)$ (см. доказательство трудоемкости алгоритма TMGT). Тогда окончательная трудоемкость алгоритма:

$$T_{TMDS} = O(h \cdot N^2 + N + M^2 + N + M^2) + O(M + |E|) = O(\max\{hN^2, M + |E|\}).$$

Утверждение доказано.

Алгоритм TMDS может быть рекомендован для использования в составе диспетчеров распределенных ВС для оптимизации вложения параллельных

программ. Приоритезация параметров выбора подсистем ЭМ позволяет настраивать алгоритм на формирование вложений адекватных требованиям поступающих параллельных программ. Результаты моделирования работы алгоритма TMDS приведены в п. 4.3.

3.4. Алгоритм формирования подсистем в пространственно-распределенных ВС

Рассмотрим задачу формирования субоптимальных подсистем пространственно-распределенных ВС для выполнения параллельных программ без заданных информационных графов. Обозначим через $X = (x_1, x_2, \ldots, x_n)$ вектор, задающий номера ЭМ, входящих в формируемую подсистему. Пусть $x_p = 1$, если p – ая ЭМ включена в состав подсистемы и $x_p = 0$ в противном случае ($p \in \{1,2,...,n\}$). В качестве показателя эффективности подсистемы будем использовать, по аналогии с п. 2.4, среднее геометрическое значение $B(X)$ пропускных способностей каналов связи между элементарными машинами подсистемы.

$$B(X) = \left(\prod_{p=1}^{n-1} \prod_{q=p+1}^{n} (x_p x_q (b_{pq} - 1) + 1) \right)^{\frac{1}{k}}, \tag{3.12}$$

где $k = n(n-1)/2$.

Требуется сформировать в пределах пространственно-распределенной ВС подсистему ЭМ, доставляющую минимум функции (3.12) и удовлетворяющую ограничениям (3.13), (3.14).

$$\sum_{i=1}^{n} x_i = M, \tag{3.13}$$

$$x_i \in \{0, 1\}, \quad i = 1, 2, \ldots, n. \tag{3.14}$$

Рассмотрим алгоритм поиска приближенных решений задачи (3.12) – (3.14).

3.4.1. Показатель однородности подсистемы распределенной ВС

Элементарные машины в пространственно-распределенных ВС могут иметь различные производительности. При формировании подсистем для выполнения параллельных программ практически оправданным является использование ЭМ, незначительно отличающихся производительностью друг от друга. Для оценки однородности подсистемы будем использовать показатель $V(X)$ – коэффициент вариации производительности ЭМ, входящих в подсистему X ($[V(X)] = \%$).

$$V(X) = \frac{\sqrt{D(X)}}{E(X)} \cdot 100\%,$$

$$E(X) = \left(\sum_{p=1}^{N} x_p \right)^{-1} \cdot \sum_{p=1}^{N} x_p \cdot \omega_{g(p)},$$

$$D(X) = \left(\sum_{p=1}^{N} x_p - 1 \right)^{-1} \cdot \sum_{p=1}^{N} (x_p \cdot \omega_{g(p)} - E(X))^2 .$$

Величины $E(X)$ и $D(X)$ – это, соответственно, оценки математического ожидания и дисперсии производительности ЭМ, входящих в подсистему X.

Будем называть подсистему X Δ-однородной, если она удовлетворяет критерию

$$K = K(X, \Delta) = \begin{cases} 1, & \text{если } V(X) \le \Delta \\ 0 & \text{иначе} \end{cases}.$$

Рассмотрим алгоритм формирования Δ-однородных подсистем в пространственно-распределённых ВС.

Алгоритм. Алгоритм последовательно включает элементарные машины из подмножеств C_{2k}, $k \in \{1, 2, \ldots, H\}$ в формируемую подсистему так, чтобы

значение $V(X)$ при включении очередного подмножества было минимальным, а подсистема оставалась Δ-однородной. Работа алгоритма начинается с поиска подмножества C_{2k}, элементарные машины которого имеют максимальную производительность. Алгоритм заканчивает работу, когда в подсистему будет включено не менее M элементарных машин. Результат работы алгоритма – это список номеров k подмножеств C_{2k}, включенных в подсистему.

Обозначим алгоритм PAHS (Processor Allocation Homogeneous Subsystem). Псевдокод алгоритма приведен в листинге 3.5.

Листинг 3.5. Псевдокод алгоритма PAHS

Входные данные: N, L, H, n_l, n_{lk}, C, C_{lk}, c_{lk}, ω_i – описание пространственно-распределенной ВС ($l \in \{1, 2, \ldots, L\}$, $k \in \{1, 2, \ldots, n_l\}$); M – ранг подсистемы.

Выходные данные: $x[p]$ – сформированная подсистема; $x[p] = 1$, если элементарная машина p включена в состав подсистемы, иначе $x[p] = 0$ ($p \in \{1, 2, \ldots, N\}$).

```
1   i_max ← 1
2   for i ← 1 to H do                    /* Поиск начального подмножества C_lk */
3      if ω_i > ω_{i_max} then
4         i_max ← i
5   end for
6   s ← 1
7   S[s] ← i_max
8   used[i_max] ← 1
9   r ← N_{i_max}
10  V′ ← 0
11  V″ ← 0
12  while r < M do                       /* Просмотр подмножеств C_lk */
```

13 $V_{min} \leftarrow \infty$

14 **for** $i \leftarrow 1$ **to** H **do** /* *Поиск очередного подмножества* C_{lk} */

15 **if** $used[i] = 0$ **then**

16 $V_1 \leftarrow N_i \cdot \omega_i + V'$

17 $V_2 \leftarrow N_i \cdot \omega_i^2 + V''$

18 $n \leftarrow r + N_i$

19 $E \leftarrow V_1 / n$

20 $D \leftarrow V_2 / (n-1) - V_1^2 / (n(n-1))$

21 $V \leftarrow \sqrt{D} / E \cdot 100\%$

22 **if** $V_{min} > V$ **then**

23 $V_{min} \leftarrow V$

24 $i_{min} \leftarrow i$

25 **end if**

26 **end if**

27 **end for**

28 **if** $V_{min} \leq \Delta$ **then** /* *Проверка на* Δ*-однородность* */

29 $s \leftarrow s + 1$

30 $S[s] \leftarrow i_{min}$

31 $r \leftarrow r + N_{i_{min}}$

32 $V' \leftarrow V' + N_{i_{min}} \cdot \omega_{i_{min}}$

33 $V'' \leftarrow V'' + N_{i_{min}} \cdot \omega_{i_{min}}^2$

34 $used[i_{max}] \leftarrow 1$

35 **else**

36 **error** "*Невозможно сформировать* Δ*-однородную подсистему*"

37 **end if**

38 **end while**

Рассмотрим пример формирования Δ-однородной подсистемы ранга 100 с параметром $\Delta = 10\%$. Рассматриваемая система приведена на рис. 3.6.

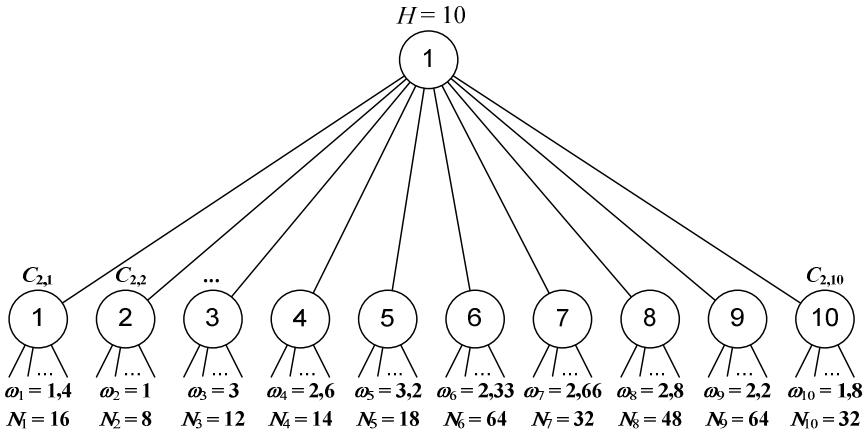
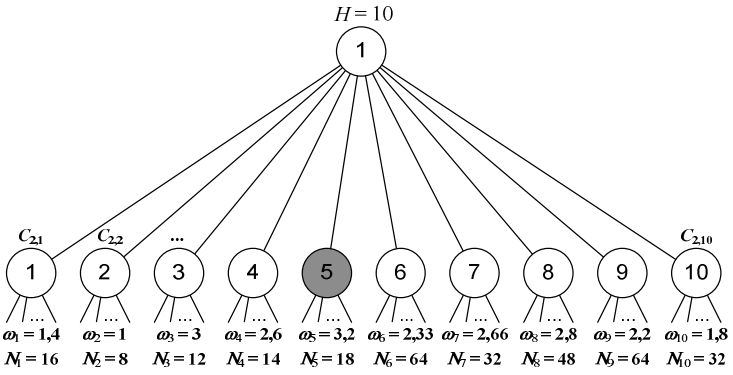


Рис. 3.6. Пространственно-распределенная ВС:

$N = 308$; $H = 10$; $[\omega_i]$ = GFLOPS

На первом шаге алгоритма (рис. 3.7) в формируемую подсистему включаются элементарные машины из подмножества C_{25} как обладающие наибольшей производительностью. Затем в подсистему последовательно включается подмножества C_{23}, C_{28}, C_{27}.

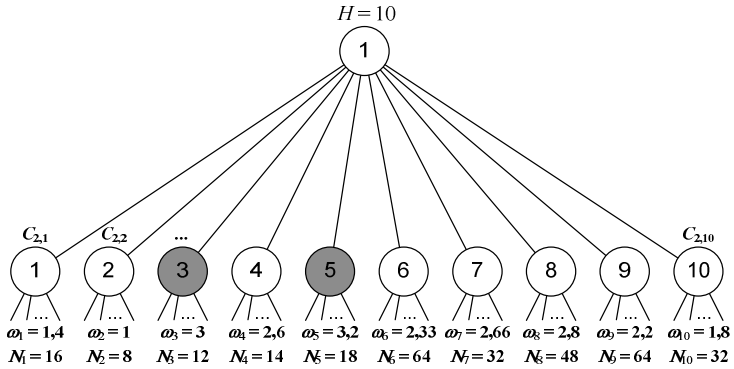
Если от формируемой подсистемы не требуется предельной производительности, то выбор подмножеств C_{lk} может начинаться с подмножества, производительность ЭМ в котором близка к среднему значению производительности ЭМ в подмножествах C_{2k}, $k \in \{1, 2, \ldots, H\}$.



$r = 18$

$V(X) = 0$

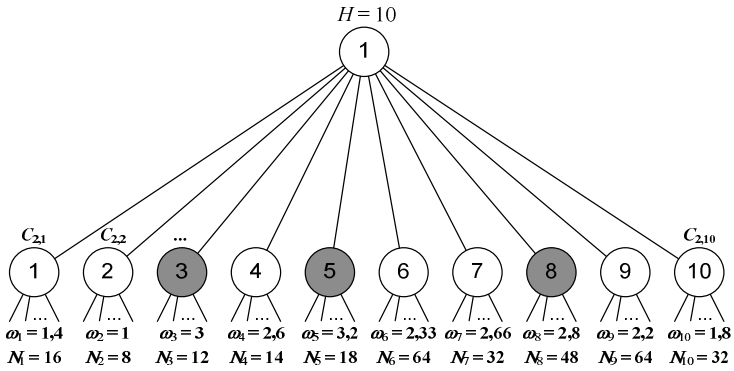
а



б

$r = 30$

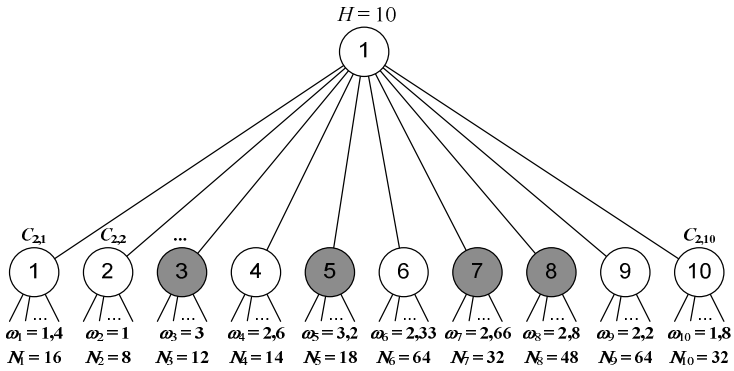
$V(X) = 4{,}56\ \%$



в

$r = 78$

$V(X) = 6{,}67\ \%$



г

$r = 110$

$V(X) = 8{,}09\ \%$

Рис. 3.7. Формирование Δ-однородной подсистемы алгоритмом PAHS:

а – включение начального подмножества;

б – включение второго подмножества; *в* – включение третьего подмножества;

г – включение четвертого подмножества

Относительно вычислительной сложности алгоритма PAHS справедливо утверждение.

Утверждение 3.3. *Вычислительная сложность алгоритма PAHS равна*

$$T_{PAHS} = O(M \cdot H).$$

Доказательство. Трудоемкость поиска начального подмножества элементарных машин равна $O(H)$. Цикл в строках 12–38 в худшем случае (если все подмножества C_{2i} содержат по одной ЭМ) выполняется M раз. Трудоемкость одной итерации цикла составляет $O(H)$. Следовательно, вычислительная сложность алгоритма равна $T_{PAHS} = O(H + M \cdot H) = O(M \cdot H)$.

Утверждение доказано.

3.4.2. Алгоритм формирования подсистем

Пусть алгоритмом PAHS из элементарных машин пространственно-распределенной ВС сформирована Δ-однородная система, в пределах которой требуется создать субоптимальную подсистему элементарных машин (в смысле показателя (3.12)). Рассмотрим алгоритм формирования подсистем.

Алгоритм. Обозначим алгоритм PADS (Processor Allocation Distributed Subsystem). В его основе лежат шаги алгоритма TMDS. В процессе работы алгоритм включает в формируемую подсистему элементарные машины из подмножеств C_{lk}. Обозначим через s количество подмножеств C_{lk}, включенных в состав подсистемы; L_i и K_i – это, соответственно, номер уровня и подмножества C_{lk}, включенного в подсистему под номером $i \in \{1, 2, \ldots, s\}$. Работа алгоритма начинается с выбора среди свободных подмножеств C_{2i}, $i = 1,2,...,H$ подмножества k с максимальным значением величины

$$B_i = \left(\left(\prod_{j=1}^{s} b(L_j, K_j, 2, i) \right)^{\frac{1}{s}} \cdot \overline{B}_{2,i} \right)^{\frac{1}{2}}.$$

Показатель B_i характеризует производительность каналов связи между ЭМ подмножества C_{2i} и производительность каналов связи до подмножеств, включенных в подсистему.

Если сумма количества элементарных машин в найденном подмножестве C_{2k} и подсистеме меньше M, то подмножество включается в подсистему и поиск подмножеств на текущем уровне продолжается. Если сумма $r + c_{2k} > M$, то осуществляется аналогичный просмотр дочерних подмножеств C_{3i}, $i \in N_{2k}$. Рекурсивный поиск среди дочерних подмножеств продолжается до тех пор, пока сумма $r + c_{lk}$ не будет меньше или равна M (или не кончатся дочерние элементы).

В результате работы алгоритма формируется список из s подмножеств $C_{L_1K_1}$, $C_{L_2K_2}$, …, $C_{L_sK_s}$. В состав подсистемы включаются первые M элементарных машины из подмножеств $C_{L_iK_i}$, в порядке их включения в подсистему. В листинге 3.6 приведен псевдокод алгоритма.

Листинг 3.6. Псевдокод алгоритма PADS

Входные данные: N, H, L, n_l, n_{lk}, N_{lk}, $N_{l,k,u}$, C, C_{lk}, c_{lk}, h_{lk}, B_{lk}, $\overline{B}_{lk}$, $b(l_1,k_1,l_2,k_2)$ – описание пространственно-распределенной ВС ($l \in \{1, 2, \ldots, L\}$, $k \in \{1, 2, \ldots, n_l\}$); M – ранг подсистемы;

Выходные данные: $x[p]$ – сформированная подсистема; $x[p] = 1$, если элементарная машина p включена в состав подсистемы, иначе $x[p] = 0$ ($p \in \{1, 2, \ldots, N\}$).

```
1   r ← 0          /* Счетчик количества ЭМ, включенных в подсистему */
2   s ← 0          /* Количество подмножеств, включенных в подсистему */
3   ProcessNode(1, 1)
4   c ← 0
```

```
5    for i ← 1 to s do
6       Enqueue(Q, C_{L_i K_i})
7       while QueueSize(Q) > 0 and c < M do
8          p ← Dequeue(Q)
9          x[p] ← 1
10         c ← c + 1
11      end while
12   end for
```

```
ProcessNode(l, k)          /* Обрабатывает дочерние узлы элемента (l,k) */
1    ComputeB(l, k)
2    while r < M do
3       i ← FindNode(l, k)
4       if c_{l+1,i} + r > M then
5          if n_{li} > 0 then
6             ProcessNode(l + 1, i)
7          else
8             AddNodeToSubsystem(k, l + 1, i)
9          end if
10      else
11         AddNodeToSubsystem(k, l + 1, i)
12      end if
13   end while
```

```
ComputeB(l, k)
1    for (для) каждого i ∈ N_{lk} do
2       B_i ← 1          /* Расстояние от подмножества i до подсистемы */
3       for j ← 1 to s do
```

4 $B_i \leftarrow B_i \cdot b(L_j, K_j, l+1, i)$

5 **end for**

6 **end for**

AddNodeToSubsystem(p, l, k) /* *Включает* C_{lk} *в подсистему* */

1 $r \leftarrow r + c_{lk}$

2 $s \leftarrow s + 1$

3 $L[s] \leftarrow 1$

4 $K[s] \leftarrow k$

5 $used[l, k] = 1$

6 **for** (для) каждого $i \in N_{l-1,p}$ **do**

7 $B_i \leftarrow B_i \cdot b(L_s, K_s, l, i)$

8 **end for**

FindNode(l, k)

1 $B_{max} \leftarrow 0$

2 **for** (для) каждого $i \in N_{lk}$ **do**

3 **if** $used[l + 1, i] = 0$ **then**

4 $B \leftarrow \overline{B}_{l+1,i}$

5 **if** $s > 0$ **then**

6 $B \leftarrow \sqrt{B \cdot (B_i)^{\frac{1}{s}}}$

7 **if** $B_{max} < B$ **then**

8 $B_{max} \leftarrow B$

9 $i_{max} \leftarrow i$

10 **end if**

11 **end if**

12 **end for**

13 **return** i_{max}

Рассмотрим пример формирования подсистемы ранга $M = 20$. На рис. 3.8 приведен пример Δ-однородной подсистемы. В качестве минимальных пропускных способностей каналов связи взяты следующие значения: Fast Ethernet – 25 МБ/с; Gigabit Ethernet – 125 МБ/с; InfiniBand – 2000 МБ/с; Myrinet – 1000 МБ/с; Shared Memory – 8000 МБ/с. Значения компонент векторов B_{lk} приведены в МБ/с.

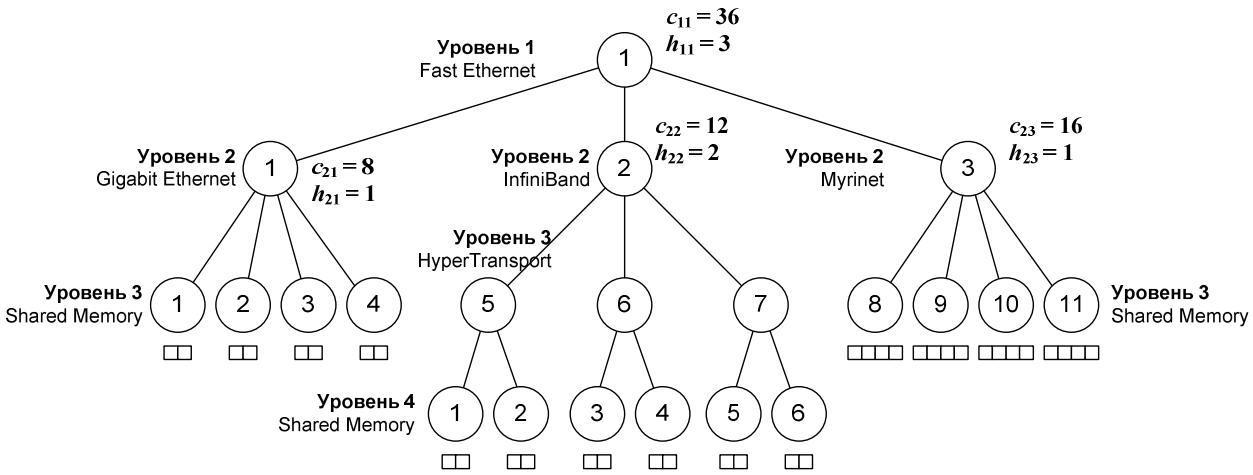


Рис. 3.8. Пространственно-распределенная ВС:

$B_{21} = (125; 8000)$; $B_{22} = (2000; 6000; 8000)$; $B_{23} = (1000; 8000)$;

$\overline{B}_{21} = 1000$; $\overline{B}_{22} = 4579$; $\overline{B}_{23} = 2828$; $b(2,1,4,3) = 25$; $b(4,5,3,6) = 6000$

$N_{2,1,3} = \{1, 2, 3, 4\}$; $N_{3,6,4} = \{3, 4\}$;

На рис. 3.9 приведены результаты работы алгоритма. Серым цветом обозначены узлы, для которых осуществлялся вызов функции *ProcessNode*, темно-серым – подмножества, включенные в состав подсистемы. Значения B_{lk} и $\overline{B}_{lk}$ аналогичны значениям на рис. 3.8.

Справедливо утверждение.

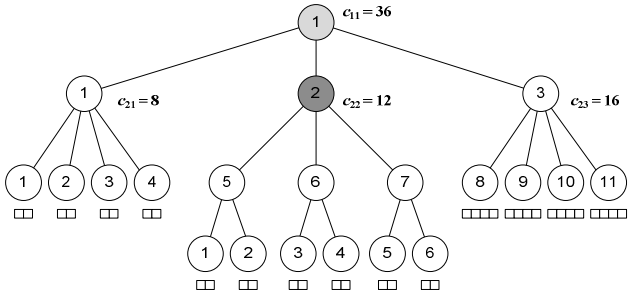
Утверждение 3.4. *Вычислительная сложность алгоритма PADS равна*

$$T_{PADS} = O(h \cdot N^2).$$

Доказательство. Доказательство аналогично доказательству трудоемкости первого этапа работы алгоритма TMDS. В худшем случае потребуется

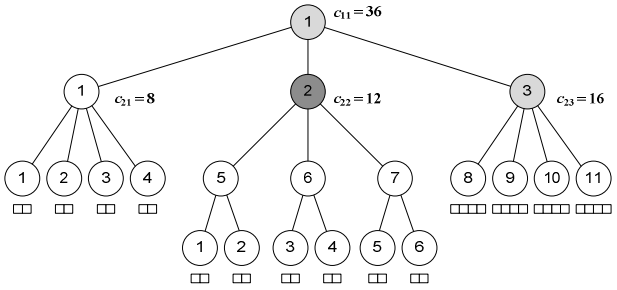
обработка всех подмножеств на h уровнях иерархии системы. Трудоемкость обработки подмножеств одного уровня (функция *ProcessNode*) равна $O(N^2 + N + N) = O(N^2)$. Суммарная трудоемкость алгоритма $T_{PADS} = O(h \cdot N^2)$.

Утверждение доказано.



$$i_{\max} = \arg\max_{i=1,2,3}\{\overline{B}_{2i}\} = 2;\ r = 12$$

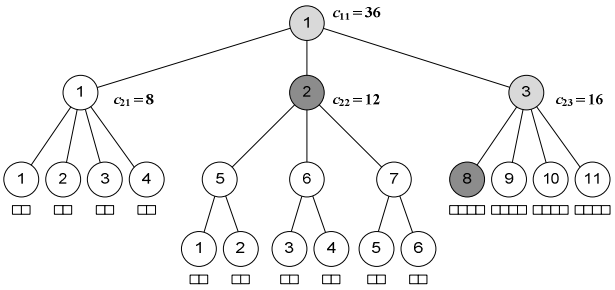
а



$$B_1 = \sqrt{125 \cdot \overline{B}_{21}} = 354;\ B_3 = \sqrt{125 \cdot \overline{B}_{23}} = 595;$$

$$i_{\max} = \arg\max_{i=1,3}\{B_i\} = 3;\ r = 12$$

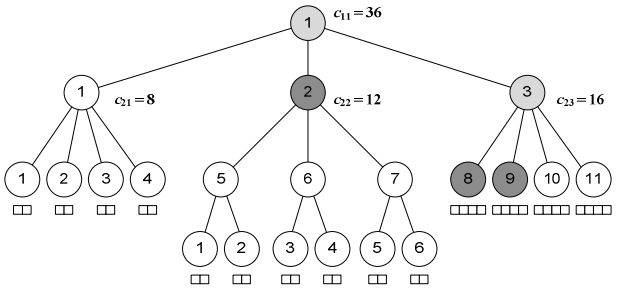
б



$$B_i = \sqrt{125 \cdot \overline{B}_{2i}},\ i = 1,2,3,4;$$

$$i_{\max} = \arg\max_{i=\overline{1,4}}\{B_i\} = 1;\ r = 16$$

в



$$B_i = \sqrt{125 \cdot \overline{B}_{2i}},\ i = 2,3,4;$$

$$i_{\max} = \arg\max_{i=2,3,4}\{B_i\} = 2;\ r = 20$$

г

Рис. 3.9. Пример формирования подсистемы алгоритмом PADS.

Итерации цикла функции *ProcessNode*:

а – первая итерация, уровень 2; *б* – вторая итерация, уровень 2;

в – первая итерация, уровень 3; *г* – вторая итерация, уровень 3

Алгоритмы PAHS и PADS могут быть рекомендованы для использования в составе систем управления ресурсами распределенных ВС для формирования однородных подсистем, допускающих эффективную реализации межмашинных обменов. Результаты моделирования работы алгоритма TMDS приведены в п. 4.4.

3.5. Выводы

1. Описаны математическая модель коммуникационных сред пространственно-распределенных ВС и задача оптимального вложения параллельных программ в такие системы. В постановке задачи учитывается время доставки программы и ее данных до ЭМ, выделенных для выполнения ветвей.

2. Предложен стохастический алгоритм TMSA субоптимального вложения параллельных программ в пространственно-распределенные системы. Для большемасштабных систем предлоежна параллельная версия TMPSA алгоритма.

3. Предложен алгоритм TMDS вложения параллельных программ в подсистемы пространственно-распределенных ВС, который обладает полиномиальной трудоемкостью и позволяет приоритезировать выбор ЭМ по их производительности и скорости передачи данных по каналам связи между ними.

4. Для формирования однородных подсистем разработаны эвристические алгоритмы PAHS и PADS.

Предложенные алгоритмы ориентированы на использование в системах управления ресурсами пространственно-распределенных ВС. В частности, алгоритмы оптимизации вложения и формирования подсистем могут быть рекомендованы для использования в составе GRID-брокеров (GRID-диспетчеров) и метапланировщиков для минимизации времени выполнения параллельных программ.

ГЛАВА 4. ПРОСТРАНСТВЕННО-РАСПРЕДЕЛЕННАЯ МУЛЬТИКЛАСТЕРНАЯ ВЫЧИСЛИТЕЛЬНАЯ СИСТЕМА

В этой главе рассматривается развитая функциональная структура пространственно-распределенной мультикластерной ВС.

Дается описание созданных программных средств оптимизации вложения параллельных MPI-программ в кластерные системы. Приведена функциональная структура пакета анализа производительности MPI-программ.

Отражены результаты моделирования работы алгоритмов, описанных в главах 2-3.

4.1. Архитектура пространственно-распределенной мультикластерной вычислительной системы

Центром параллельных вычислительных технологий Государственного образовательного учреждения высшего профессионального образования “Сибирский государственный университет телекоммуникаций и информатики” совместно с Институтом физики полупроводников им. А. В. Ржанова Сибирского отделения РАН создана и развивается пространственно-распределенная мультикластерная вычислительная система (см. прил. 2). На данный момент система объединяет 8 пространственно-распределенных кластерных ВС, причем кластеры A-F расположены в ЦПВТ ГОУ ВПО “СибГУТИ”, а кластеры G, H – в Лаборатории ВС ИФП СО РАН. Любой из кластеров способен функционировать как автономно так и в составе распределенной ВС. В качестве базовых составляющих для построения кластеров A-D, H использованы стандартные персональные компьютеры на базе процессоров семейства Intel: для кластера A – Intel Core 2 Duo, для кластеров B, C – Celeron, D – Pentium 4 / Core 2 Duo, для кластера H – Pentium 4. Кластер G состоит из 6 двухпроцессорных узлов, каждый из которых представляет собой систему NUMA на базе процессоров AMD Opteron 248, кластер E уком-

плектован 4 двухпроцессорными узлами на базе двухядерных процессоров Intel Xeon 5150. Кластер F оснащен четырехъядерными процессорами Intel Xeon E5345, в состав кластера включено 32 вычислительных ядра с суммарной производительностью 298 GFLOPS.

Коммуникационные среды вычислительных кластеров построены на базе технологий Gigabit и Fast Ethernet. Для объединения кластеров используется сеть Internet (технология VPN). Связь осуществляется через выделенные серверы сегментов ЦПВТ ГОУ ВПО “СибГУТИ” и Лаборатории ВС ИФП СО РАН. Система включает больше 100 процессоров и имеет пиковую производительность 690 GFLOPS.

Мультикластерная ВС допускает масштабирование путем организации взаимодействия с множеством других систем.

В прил. 2 приведено писание текущей конфигурации мультикластерной ВС и структурной организации коммуникационных сред сегментов G, E, F.

4.2. Программное обеспечение мультикластерной ВС

4.2.1. Стандартные компоненты

Системное программное обеспечение мультикластерной ВС основано на операционной системе GNU/Linux (рис. 4.1). На вычислительных узлах системы используются дистрибутивы CentOS и Fedora.

Поддерживается разработка и выполнение параллельных программ в двух стандартах: OpenMP (установлены компиляторы с языков C/C++, FORTRAN компаний Intel, Sun Microsystems и проекта GNU) и MPI (пакеты MPICH2 и OpenMPI).

Мультипрограммные режимы функционирования организуются оригинальными средствами, разработанными в ЦПВТ ГОУ ВПО “СибГУТИ”, и системой пакетной обработки заданий TORQUE.

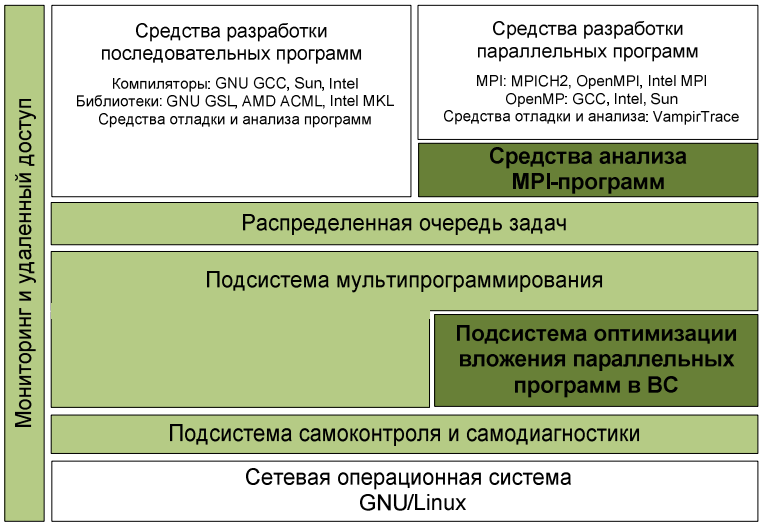


Рис. 4.1. Структура программного обеспечения пространственно-распределенной мультикластерной ВС:

Стандартные компоненты

Средства поддержки мультипрограммных режимов функционирования

Средства оптимизации вложения и анализа параллельных программ

4.2.2. Средства оптимизации вложения параллельных MPI-программ

Стандартные средства управления ресурсами кластерных ВС не учитывают мультиархитектурную организацию современных ВС. Для выполнения параллельных программ с эффективным вложением в ВС требуется разработка дополнительного системного программного обеспечения, позволяющего осуществлять оптимизацию вложения параллельных программ в распределенные ВС.

На основе предложенных алгоритмов вложения параллельных программ в распределенные ВС создано программное средство MPITaskMap, позволяющее организовывать выполнение MPI-программ с субоптимальным распределением их ветвей по ЭМ системы. Программное средство допускает интеграцию с системами пакетной обработки заданий (в частности с TORQUE) и поддерживает стандартизованный интерфейс утилиты mpiexec. На рис. 4.2 приведена схема работы пакета MPITaskMap.

Исполняемый код MPI-программ и ее паспорт, в котором содержится описание информационного графа, поступают на вход пакета MPITaskMap. В зависимости от текущих настроек и конфигурации ВС, применяется один из алгоритмов вложения. Далее формируется сценарий запуска программы с субоптимальным распределением ветвей по ЭМ и осуществляется ее выполнение.

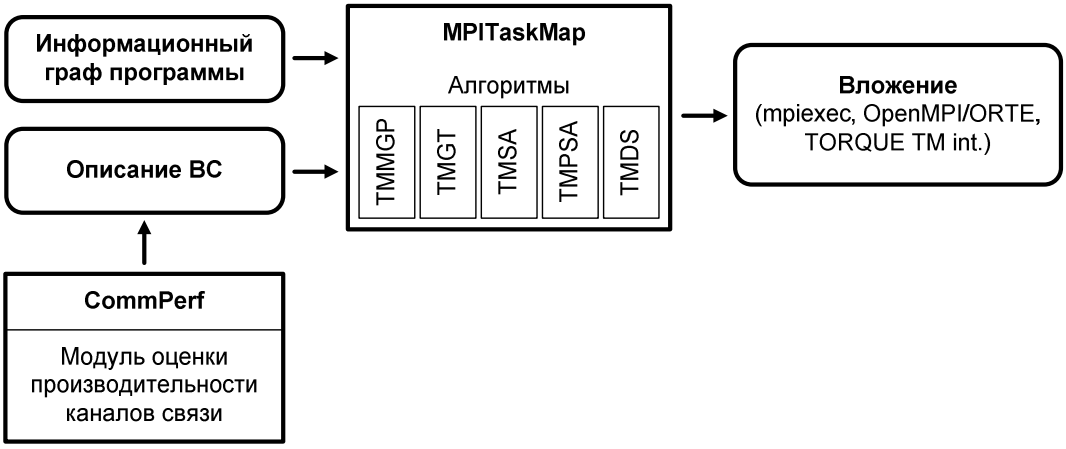


Рис. 4.2. Схема работы пакета MPITaskMap

Описание ВС формируется при помощи разработанной утилиты CommPerf, которая осуществляет оценку производительности каналов связи между элементарными машинами ВС на уровне протокола MPI.

Пакет MPITaskMap разработан на языке программирования C для операционной системы GNU/Linux.

4.2.3. Средства анализа параллельных MPI-программ

Для анализа производительности MPI-программ и автоматизации формирования их информационных графов разработано программное средство OTFStat. Поддерживаются протоколы выполнения MPI-программ в формате Open Trace Format (OTF) [161]. Созданное средство позволяет получать информацию о структуре обменов между параллельными ветвями, дисбалансе их загрузки и коэффициентах накладных расходов. Опишем основные этапы работы пакета OTFStat (рис. 4.3). Пользователь осуществляет компиляцию

MPI-программы с библиотекой инструментации пакета VampirTrace [82]. Скомпилированная MPI-программа выполняется на системе. В результате формируется протокол, содержащий информацию о вызовах коммуникационных функций. Протокол оформляется в формате OTF и поступает на вход пакета OTFStat, где происходит его предварительная обработка и анализ. Строится взвешенный граф межмашинных обменов, собирается статистика по времени пребывания программы в различных коммуникационных функциях и пр. После этапа анализа осуществляется формирование отчета с подробной информацией о выполнении программы и информационных обменах.

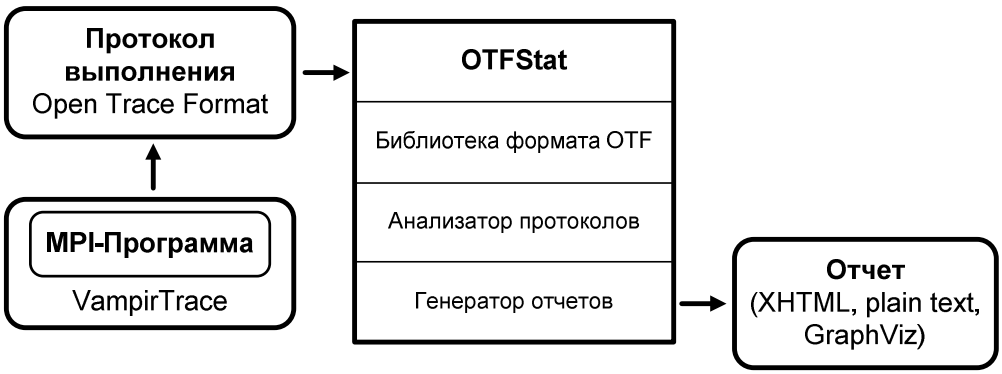


Рис. 4.3. Схема работы анализатора OTFStat

На рис. 4.4 приведен пример матрицы обменов между ветвями MPI-программы NPB Conjugate Gradient, на рис. 4.5 статистика обменов между ветвями. Отчеты сгенерированы пакетом OTFStat.

Разработанные программные средства MPITaskMap и OTFStat в композиции с известными системами управления ресурсами распределенных ВС составляют базу для организации эффективного функционирования распределенных ВС.

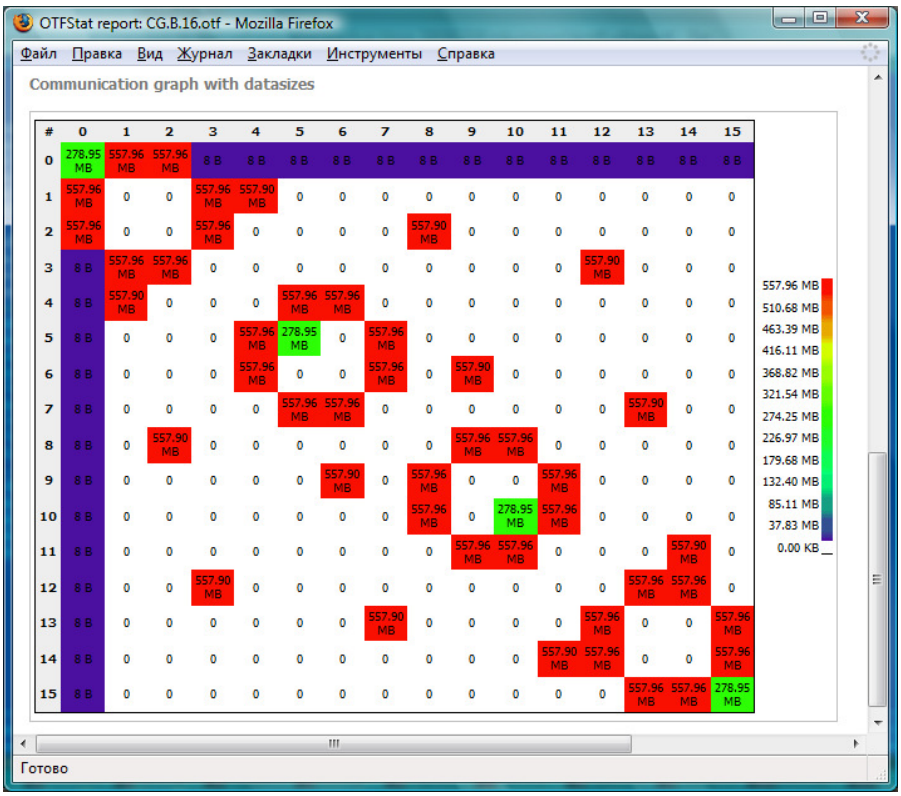


Рис. 4.4. Матрица обменов MPI-программы NPB Conjugate Gradient: $M = 16$; CLASS = B

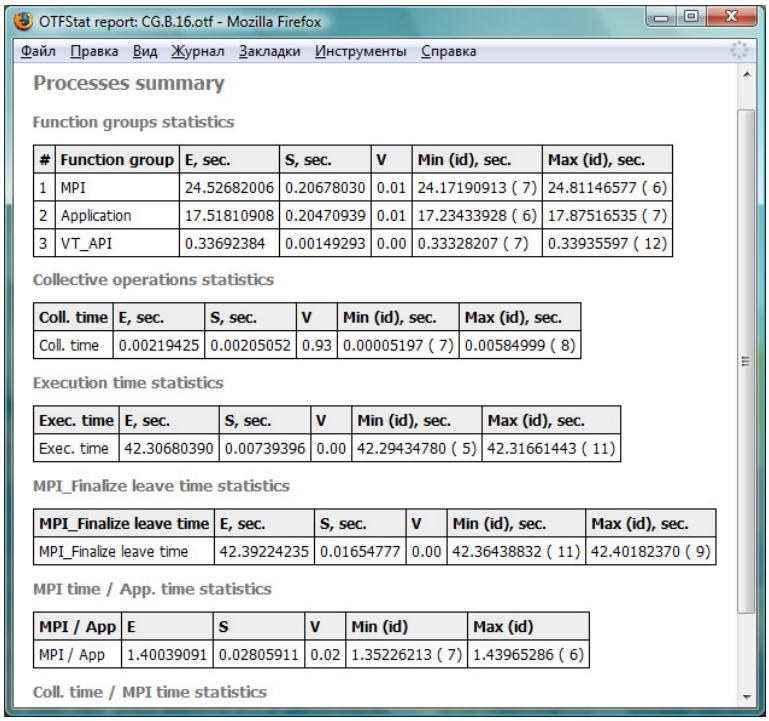


Рис. 4.5. Статистика обменов между ветвями MPI-программы NPB Conjugate Gradient: $M = 16$; CLASS = B

4.3. Моделирование алгоритмов вложения параллельных программ в распределенные ВС

Моделирование алгоритмов TMMGP, TMGT, TMSA, TMPSA, TMDS вложения параллельных программ в распределенные ВС состояло из четырех частей. В первой проводилось численно моделирование работы алгоритма TMMGP и его сравнение по времени работы и качеству формируемых вложений с известными алгоритмами TML и TMRR (см. п. 1.4.4). Результаты численного моделирования проверялись натурными экспериментами по вложению тестовых параллельных MPI-программ в действующие вычислительные кластеры с иерархической организацией коммуникационных сред.

Целью второй части моделирования было исследование работы алгоритма TMGT. Рассматривались тестовые конфигурации подсистем распределенных ВС с иерархической организацией коммуникационных сред.

В третьей части исследовались алгоритмы TMSA и TMPSA вложения параллельных программ в пространственно-распределенные ВС. Оценивались время работы алгоритмов и качество формируемых вложений. Для параллельного алгоритма TMPSA определялся коэффициент ускорения.

Четвертая часть посвящена моделированию работы алгоритма TMDS. Качество работы алгоритма исследовалось на тесовых конфигурациях пространственно-распределенных мультикластерных ВС.

Результаты моделирования подвергались статистической обработке. Строились оценки математических ожиданий значений показателей качества работы алгоритмов на тестовых наборах входных данных.

Алгоритмы реализованы на языках программирования C, C++ и моделировались на сегментах E, F, G пространственно-распределенной мультикластерной ВС.

Моделирование алгоритма TMMGP. Моделирование работы алгоритмы TMMGP осуществлялось в два этапа. На первом этапе исследовались время работы алгоритмы и качество формируемых вложений. На втором эта-

пе проводился натурный эксперимент по вложению промышленных MPI-программ в кластерные ВС.

На рис. 4.6 и 4.7 приведены графики зависимости времени работы алгоритма от количества вершин и ребер в информационном графе параллельной программы. Рассматривались графы следующих структур: 2*D*-решетка, звезда, кольцо, линейка. На рис. 4.6 представлены результаты вложения в ВС $N = 16384$, $L = 3$, $c_{L1} = 2$. Время работы алгоритма приведено для процессора Intel Xeon E5345.

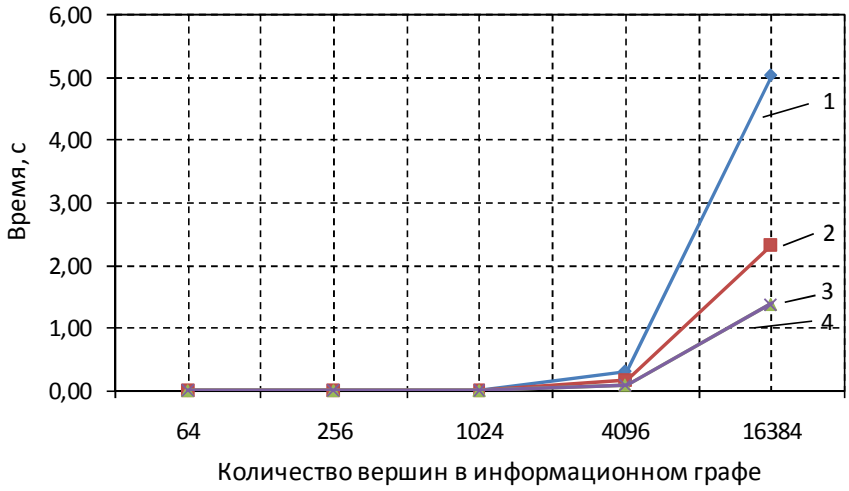


Рис. 4.6. Зависимость времени работы алгоритма TMMGP от количества *M* вершин в информационном графе: $c_{L1} = 2$; 1 – 2*D*-решетка; 2 – звезда; 3 – кольцо; 4 – линейка

На рис. 4.7 результаты вложения в систему $N = 16384$, $L = 3$, $c_{L1} = 8$.

Из графиков видно, что время работы алгоритма незначительно и для большемасштабных ВС сопоставимо с системным временем подготовки подсистемы для запуска ветвей параллельной программы. Графики подтверждают результаты утверждения 2.2. Видно, что с уменьшением k количества подмножеств разбиения время работы алгоритмы уменьшается.

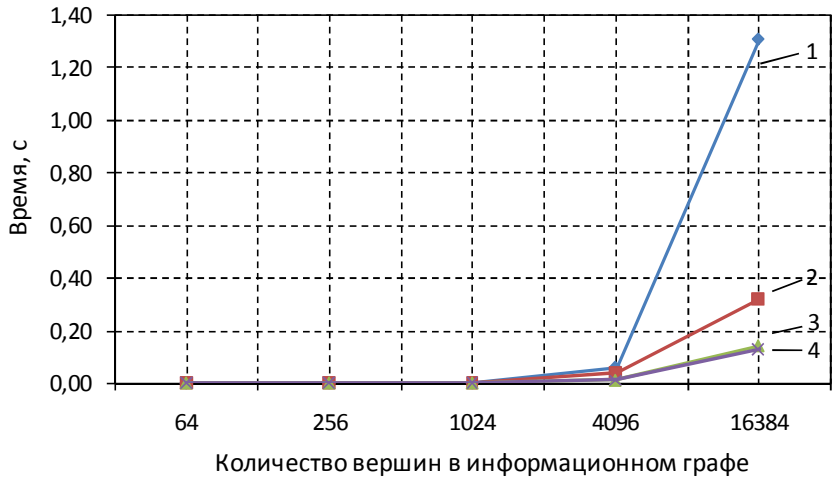


Рис. 4.7. Зависимость времени работы алгоритма TMMGP от количества M вершин в информационном графе: $c_{L1} = 8$; 1 – 2D-решетка; 2 – звезда; 3 – кольцо; 4 – линейка

Оценка качества формируемых алгоритмом TMMGP вложений осуществлялась для информационных графов со структурами: 2D-решетка, звезда, кольцо и линейка. Для сравнения приведены результаты работы алгоритмов TML и TMRR. В таблице 4.1 приведены результаты вложений параллельных программ, состоящих из $M = 512$ ветвей, в систему $N = 16384$, $L = 3$, $c_{L1} = 4$, $b_1 = 2 \cdot 2^{30}$, $b_2 = 6 \cdot 2^{30}$, $b_3 = 8 \cdot 2^{30}$.

Таблица 4.1

Структура информационного графа	Значение целевой функции T, с		
	$T(X_{TML})$, с	$T(X_{TMRR})$, с	$T(X_{TMMGP})$, с
2D-решетка	462	466	166
Звезда	125568	126525	125560
Кольцо	312	333	312
Линейка	312	333	312

В таблице 4.2 отражены результаты вложений параллельных программ ($M = 512$) в систему с параметрами: $N = 16384$, $L = 7$, $c_{L1} = 4$, $b_1 = 0{,}5 \cdot 2^{30}$, $b_2 = 1 \cdot 2^{30}$, $b_3 = 2 \cdot 2^{30}$, $b_4 = 3 \cdot 2^{30}$, $b_5 = 4 \cdot 2^{30}$, $b_6 = 6 \cdot 2^{30}$, $b_7 = 8 \cdot 2^{30}$.

Таблица 4.2

Структура информационного графа	Значение целевой функции T, с		
	$T(X_{TML})$, с	$T(X_{TMRR})$, с	$T(X_{TMMGP})$, с
2D-решетка	1712	1816	316
Звезда	432168	488700	423997
Кольцо	1062	1083	1062
Линейка	1062	1083	1062

Анализ результатов моделирования показал, что алгоритм TMMGP позволяет формировать эффективные вложения для насыщенных информационных графов с неоднородными весами ребер. Возможности алгоритма по вложению параллельных программ со структурами информационных графов в виде линей, кольца и звезды ограничены. Видно, что качество вложения зависит от разброса производительности каналов связи на различных уровнях коммуникационной среды. Чем больше уровней в коммуникационной среде и значительней разброс производительности каналов связи, тем выше эффективность вложений, формируемых алгоритмов TMMGP.

В среднем на тестовых конфигурациях систем и информационных графов алгоритм TMMGT формировал решения в 1,01 – 5 раз лучше алгоритмов TML и TMRR.

На втором этапе проводилась серия натурных экспериментов по вложению алгоритмом TMMGP промышленных MPI-программ в действующие вычислительные кластеры – сегменты G и E (см. прил. 2). Оба кластера укомплектованы двухпроцессорными узлами и имеют коммуникационную среду с иерархической организацией. В качестве тестовых параллельных задач рас-

сматривались MPI-программы из пакетов NAS Parallel Benchmarks (NPB) и High-Performance Linpack (HPL).

На рис. 4.8 представлен информационный граф теста HPL, реализующего решение системы линейных алгебраических уравнений (СЛАУ) методом LU-факторизации. На рис. 4.8 – информационный граф теста NPB Conjugate Gradient (NPB CG), осуществляющего решение СЛАУ методом сопряженных градиентов. На рис. 4.10 – информационный граф теста NPB Multigrid, строящего решение трехмерного уравнения Пуассона многосеточным методом. Информационные графы сформированы средствами пакета OTFStat. В табл. 4.3 приведены результаты выполнения MPI-программ с различными вариантами вложений на различных сетях связи. Время работы алгоритма TMMGP на всех тестовых задачах на процессоре Intel Core 2 Duo 2.13 GHz не превышало $5 \cdot 10^{-3}$ с.

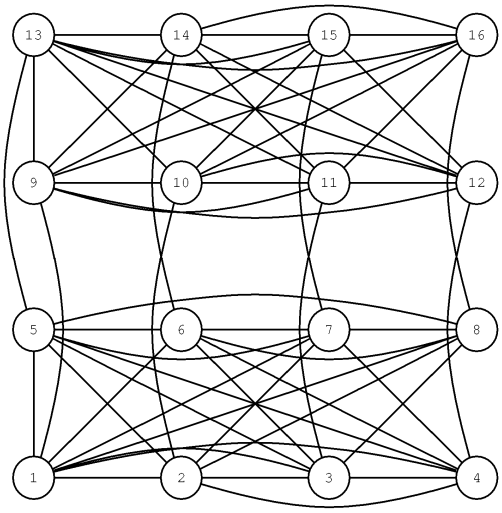


Рис. 4.8. Информационный граф теста HPL:
M = 16; PMAP = 0; BCAST = 5

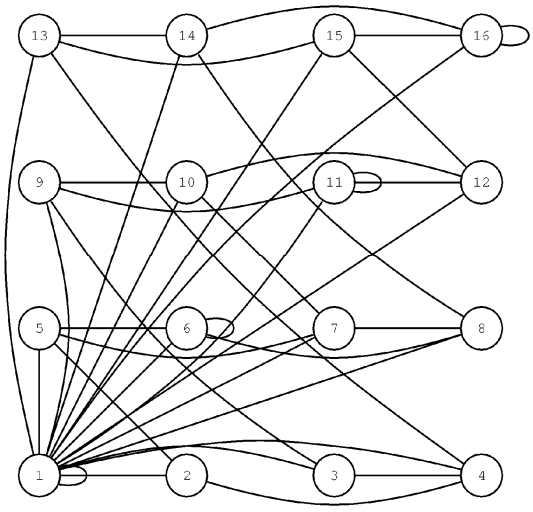


Рис. 4.9. Информационный граф теста NPB Conjugate Gradient:
M = 16; CLASS = B

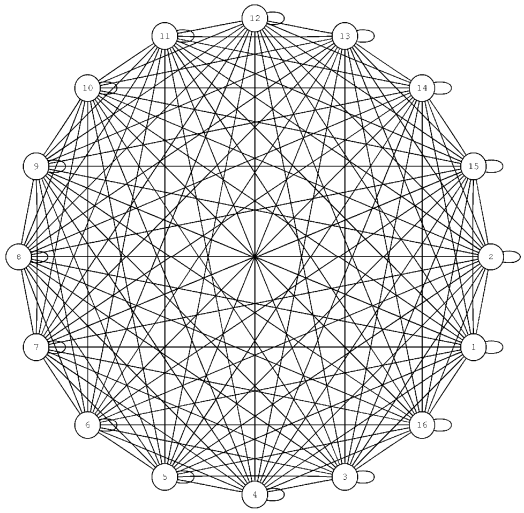


Рис. 4.10. Информационный граф теста NPB Multigrid:
M = 16; CLASS = B

Таблица 4.3

Сеть связи	Время выполнения MPI-программы, с		
	$T(X_{TMRR})$, с	$T(X_{TMMGP})$, с	$T(X_{TMRR})$ / $T(X_{TMMGP})$
High-Performance Linpack			
Fast Ethernet	1108,69	911,81	1,22
Gigabit Ethernet	263,15	231,72	1,14
NPB Conjugate Gradient			
Fast Ethernet	726,02	400,36	1,81
Gigabit Ethernet	97,56	42,05	2,32
NPB Multigrid			
Fast Ethernet	23,94	23,90	1,00
Gigabit Ethernet	4,06	4,03	1,00

Данные в табл. 4.3 подтверждают результаты численного моделирования. Видно, что качество вложения в значительной степени зависит от структуры информационного графа задачи, а также от разброса значений показателей производительности на различных уровнях коммуникационной среды. Важно отметить, что существует класс параллельных программ, для которых

возможности по оптимизации вложения существенно ограничены. К этому классу относятся программы, информационные графы которых являются полными и имеют однородные по объему передаваемых данных связи между ветвями. Примером может служить тест NPB MG.

Моделирование алгоритма TMGT. Для исследования алгоритма TMGT формировались тестовые подсистемы с иерархической организацией коммуникационных сред. Рассматривались следующие конфигурации систем: $N \in \{2048, 4096, 8192\}$, $L \in \{2, 3, \ldots, 10\}$, $c_{L1} \in \{2, 4, 8\}$. Подсистемы формировались путем случайного распределения параллельных задач различных рангов по ЭМ и их исключения из состава системы.

На рис. 4.11 приведен график зависимости времени работы алгоритма TMGT от количества ветвей в параллельной программе и структуры информационного графа. Результаты приведены для тестовой конфигурации ВС $N = 4096$ (количество доступных ЭМ), $L = 4$, $c_{L1} = 4$, $b_1 = 1 \cdot 2^{30}$, $b_2 = 2 \cdot 2^{30}$, $b_3 = 6 \cdot 2^{30}$, $b_4 = 8 \cdot 2^{30}$. Время работы алгоритма указано для процессора Intel Xeon E5345.

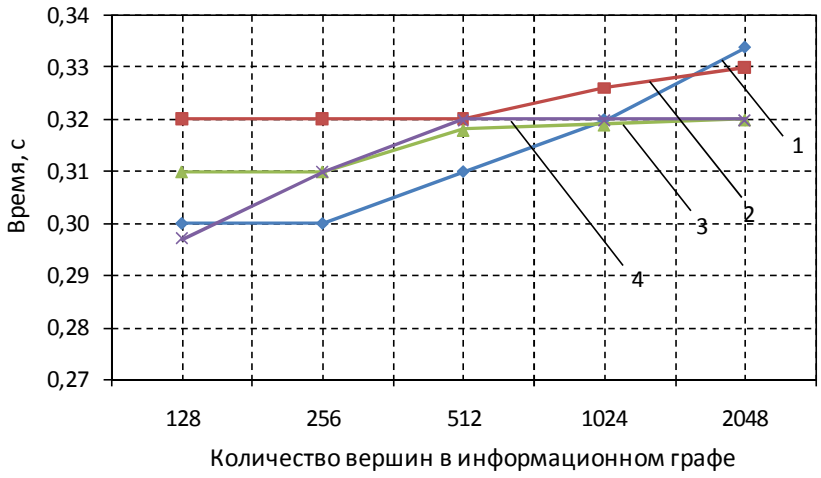


Рис. 4.11. Зависимость времени работы алгоритма TMGT от количества M вершин в информационном графе: 1 – 2D-решетка; 2 – звезда; 3 – кольцо; 4 – линейка

На рис. 4.12 приведена зависимость времени выполнения алгоритма от количества ветвей (структура графа – 2*D*-решетка) в параллельной программе и ранга подсистемы.

Из графиков видно, что время работы алгоритма TMGT находится в зависимости от ранга подсистемы и количества ребер в информационном графе параллельной программы. В целом, время работы алгоритма приемлемо для практического использования на подсистемах с рангом порядка десятка тысяч ЭМ.

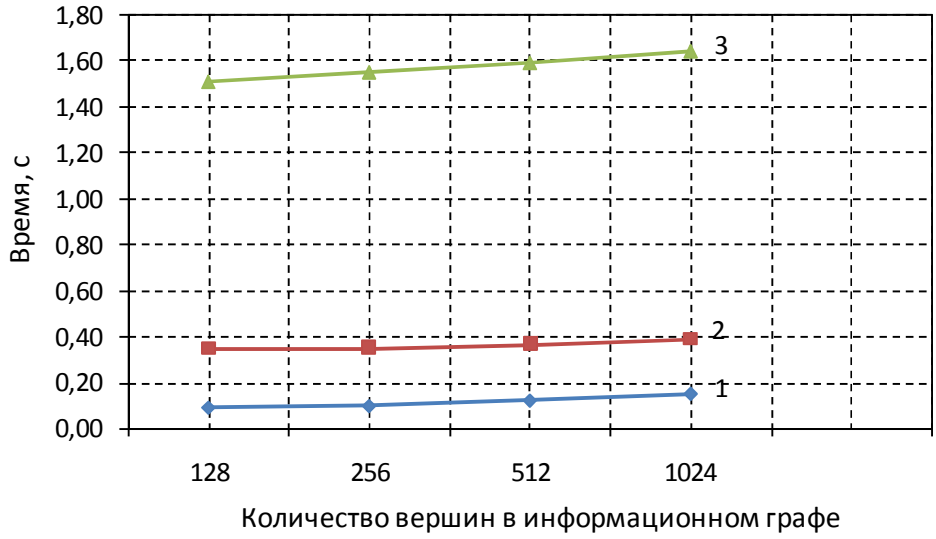


Рис. 4.12. Зависимость времени работы алгоритма TMGT от количества *M* вершин в информационном графе:
1 – $N = 2048$; 2 – $N = 4096$; 3 – $N = 8192$

В таблице 4.4 отражены результаты вложения параллельных программ ($M = 128$) в систему $N = 4096$, $L = 4$, $c_{L1} = 8$, $b_1 = 1 \cdot 2^{30}$, $b_2 = 2 \cdot 2^{30}$, $b_3 = 6 \cdot 2^{30}$, $b_4 = 8 \cdot 2^{30}$.

Качество формируемых алгоритмом TMGT вложений зависит от структуры информационного графа программы. В среднем на тестовых наборах алгоритм TMGT формировал вложения по значению целевой функции в 1,1 – 1,5 раза лучше алгоритма TML и 1,1 – 1,6 раз лучше алгоритма TMRR.

Таблица 4.4

Структура информационного графа	Значение целевой функции T, с		
	$T(X_{TML})$, с	$T(X_{TMRR})$, с	$T(X_{TMGT})$, с
2D-решетка	1062	1333	790
Звезда	238520	251583	210104
Кольцо	562	583	312
Линейка	561	580	310

В табл. 4.5 приведены результаты натурного эксперимента по вложению тестовых MPI-программ (см. моделирование алгоритма TMMGP) в подсистему сегмента E мультикластерной ВС.

Таблица 4.5

Сеть связи	Время выполнения MPI-программы, с		
	$T(X_{TMRR})$, с	$T(X_{TMGT})$, с	$T(X_{TMRR})$ / $T(X_{TMGT})$
High-Performance Linpack			
Fast Ethernet	120,92	84,35	1,43
Gigabit Ethernet	25,10	19,75	1,27
NPB Conjugate Gradient			
Fast Ethernet	727,00	390,35	1,86
Gigabit Ethernet	101,14	41,64	2,42
NPB Multigrid			
Fast Ethernet	23,9	23,9	1,00
Gigabit Ethernet	4	4	1,00

Моделирование алгоритмов TMSA и TMPSA. Алгоритмы TMSA и TMPSA реализованы на языке программирования C++. При реализации алгоритма TMPSA использовалась библиотека стандарта MPI – MPICH2. Моделирование работы алгоритмов осуществлялось со следующими параметрами: $R = \log_2 N$, $c_{\max} = \Delta F_{\max}$, $c_{\min} = 0.1$, где $\Delta F_{\max}$ – разность между верхней и

нижней оценками времени выполнения параллельной программы на ВС при заданном вложении.

Для исследования работы алгоритмов генерировались тестовые конфигурации пространственно-распределенных ВС с числом ЭМ $N = 256, 1024, 4096, 16384, 65536$ путем равновероятного случайного выбора подсистем с количеством элементарных машин $n_i \in \{64, 128, 256, 512, 1024, 2048, 4096, 16384, 65536\}$. В качестве коммуникационных сред для уровня 2 рассматривались технологии: Gigabit Ethernet, InfiniBand и Myrinet. При наличии уровня 3 брались значения показателей производительности среды доступа к общей памяти. Значения производительности каналов связи, через которые взаимодействуют подсистемы, выбирались случайно из множества {0,1 Mbps, 1 Mbps, 10 Mbps, 100 Mbps, 1000 Mbps}.

В качестве структур информационных графов параллельных программ рассматривались: линейка, кольцо, звезда и 2*D*-решетка. Для каждой структуры строились графы с количеством вершин $M = 256, 512, 1024, 2048$, с однородными и неоднородными по значениям w_i и d_{ij} вершинами и ребрами.

В качестве оценок эффективности работы алгоритмов рассматривались величины $\delta_1 = (\tilde{F} - F)/F$, $\delta_2 = (\tilde{F} - F - t)/(F + t)$, $\delta_3 = (F_0 - F)/F$, где t – время работы алгоритма, F_0 – значение целевой функции от начального решения, а F и $\tilde{F}$ – значения целевой функции от решения, получаемого алгоритмом, и случайным вложением, соответственно.

Моделирование показало, что для δ_1, δ_2, δ_3 оценки математических ожиданий и среднеквадратических отклонений для рассмотренных тестовых наборов данных составили: $\mathrm{M}[\delta_1] = 74.13$, $\sigma[\delta_1] = 17.06$, $\mathrm{M}[\delta_2] = 74.13$, $\sigma[\delta_2] = 17.06$, $\mathrm{M}[\delta_3] = 0.27$, $\sigma[\delta_3] = 0.02$. В среднем алгоритм TMSA на рассмотренных наборах модельных данных позволяет получать решение задачи в 75 раз лучше случайного вложения и в 1.27 раз лучше начального решения.

Качество вложений существенно зависит от объемов данных, передаваемых между ветвями параллельной программы, а также от размеров передаваемых сообщений. На рис. 4.13 приведен график зависимости значений оценки $\mathrm{M}[\delta_1]$ от объемов данных, передаваемых между ветвями однородной параллельной программы (конфигурация распределенной ВС однородная).

Неоднородности каналов связи на различных уровнях коммуникационной подсистемы распределенной ВС также оказывает заметное влияние на качество вложения. На рис. 4.14 представлен график зависимости значения $\mathrm{M}[\delta_1]$ от величины Δb – отношения максимального значения пропускной способности каналов связи внутри кластеров (уровень 2) к максимальному значению пропускной способности каналов связи между кластерами (уровень 1).

На рис. 4.15 приведены графики зависимости времени работы параллельного алгоритма TMPSA на сегменте E мультикластерной ВС от количества используемых процессорных ядер.

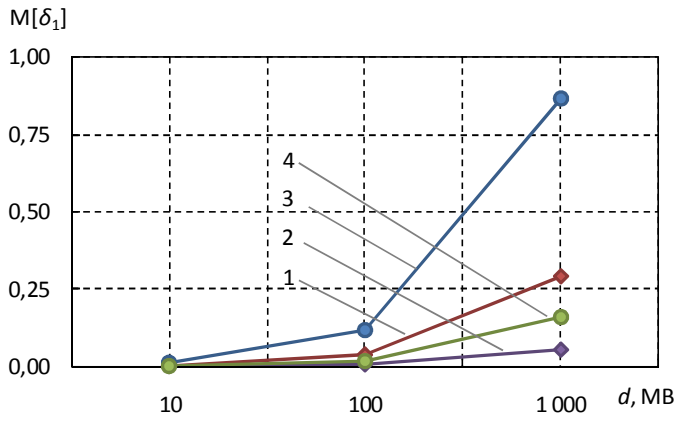


Рис. 4.13. Зависимость оценки $\mathrm{M}[\delta_1]$ от объема данных передаваемых между ветвями параллельной программы:

$N = 16384;\ n_i = 256;\ M = 1024;$

1 – линейка $m_{ij} = 1\mathrm{KB};$ 2 – линейка $m_{ij} = 1\mathrm{MB};$

3 – $2D$-решетка $m_{ij} = 1\mathrm{KB};$ 4 – $2D$-решетка $m_{ij} = 1\mathrm{MB}$

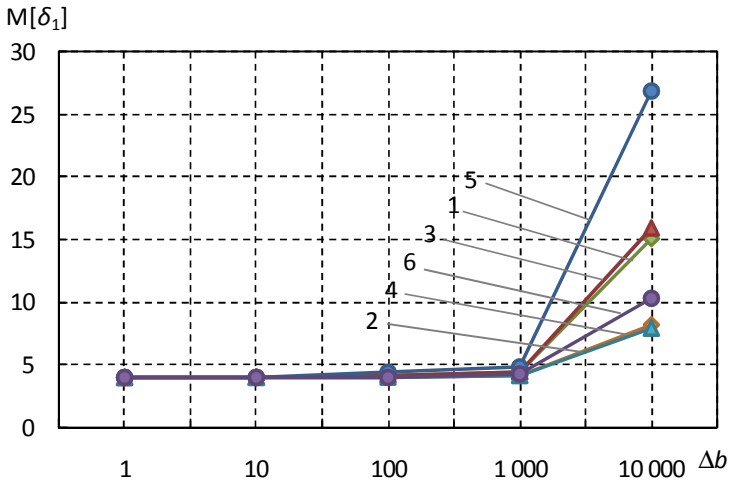


Рис. 4.14. Зависимость оценки $\mathrm{M}[\delta_1]$ от Δb: $N = 16384$; $H = 20$; $M = 1024$;

1 – однородная линейка; 2 – неоднородная линейка;

3 – однородное кольцо; 4 – неоднородное кольцо;

5 – однородная 2*D*-решетка; 6 – неоднородная 2*D*-решетка

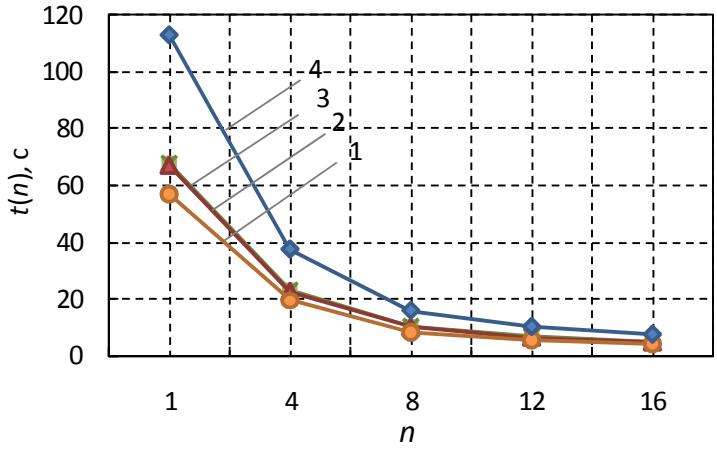


Рис. 4.15. Зависимость времени выполнения алгоритма TMPSA

от числа используемых ЭМ: $N = 131072$; $M = 2048$;

1 – линейка; 2 – кольцо; 3 – звезда; 4 – 2*D*-решетка

По результатам моделирования можно сделать вывод, что алгоритмы TMSA и TMPSA могут быть рекомендованы для использования в составе

систем управления ресурсами пространственно-распределенных ВС для оптимизации вложения сложных задач.

Моделирование алгоритма TMDS. Алгоритм TMDS исследовался на тестовых конфигурациях пространственно-распределенных мультикластерных ВС. Параметры систем задавались как равномерно распределенные случайные числа из интервалов: $H \in [2; 32]$, $L \in [3; 4]$, $h_i \in [1; 2]$, $i \in \{1, 2, \ldots, H\}$, $n_{2k} \in [4; 128]$, $n_{3k} \in [0; 2]$, $c_{Lk} \in [2; 8]$, $b_i \in [25 \cdot 2^{20}; 8 \cdot 2^{30}]$, $\omega_i \in [2 \cdot 10^9; 12 \cdot 10^9]$.

Рассматривались информационные графы следующих видов: 2D-решетка, звезда, кольцо, линейка. Формировались однородные и неоднородные по значениям w_i и d_{ij} графы с количеством вершин $M \in [256; 2048]$. Моделирование работы алгоритма проводилось для значений параметра $\delta = 0, 1$.

На рис. 4.16, 4.17 представлены графики зависимости времени работы алгоритма от количества вершин и ребер в информационном графе задачи. Время работы приведено для процессора Intel Xeon E5345.

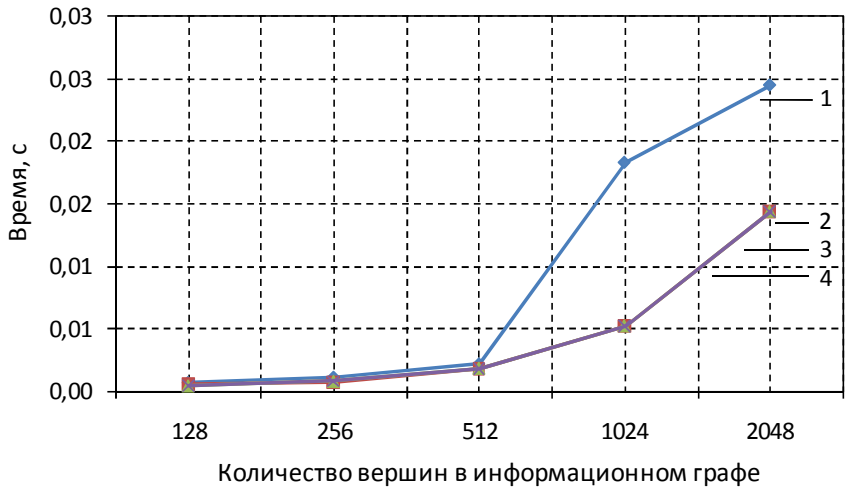


Рис. 4.16. Зависимость времени работы алгоритма TMDS от количества вершин в информационном графе: $\delta = 0$, $N = 6370$; $L = 4$; $H = 10$;
1 – 2D-решетка; 2 – звезда; 3 – кольцо; 4 – линейка

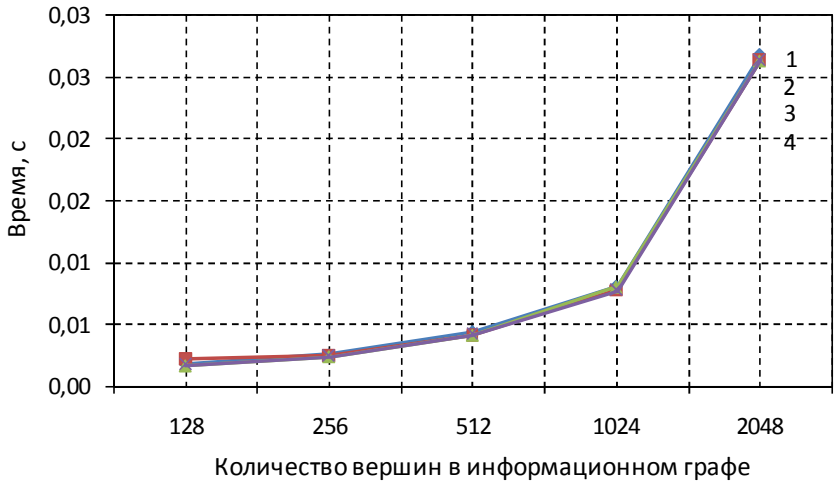


Рис. 4.17. Зависимость времени работы алгоритма TMDS от количества вершин в информационном графе: $\delta = 0$, $N = 66104$; $L = 4$; $H = 10277$;
1 – 2D-решетка; 2 – звезда; 3 – кольцо; 4 – линейка

Из графиков видно, что время работы алгоритма возрастает с увеличением ранга параллельной задачи. В случае большемасштабных систем и значительной разности между рангом подсистемы и задачи структура графа практически не влияет на время выполнения алгоритма.

В качестве оценки эффективности формируемых алгоритмом вложений рассматривался показатель $\Delta = (\tilde{F} - F)/F$, где F и $\tilde{F}$ – значения целевой функции от решения, получаемого алгоритмом и случайным вложением соответственно.

По данным статистической обработки результатов моделирования построены оценки математического ожидания и среднеквадратического отклонения показателя Δ: $\mathrm{M}[\Delta] = 90$, $\sigma[\Delta] = 16$. В среднем алгоритм TMDS на рассмотренных наборах модельных данных позволяет получать решение задачи в 90 раз лучше случайного вложения. В случае если значение параметра δ соответствовало характеру межмашинных обменов и объемам операций, выполняемых ветвями параллельной программы, то алгоритм формировал более качественные вложения: $\mathrm{M}[\Delta] = 185$, $\sigma[\Delta] = 14$. В целом, выводы по

результатам моделирования алгоритма TMDS совпадают с выводами по качеству работы алгоритма TMSA.

4.4. Моделирование алгоритмов формирования подсистем в распределенных ВС

Алгоритмы формирования подсистемы ВС исследовались на различных тестовых конфигурациях распределенных ВС. Оценивалось время работы алгоритмов и качество формируемых подсистем.

Моделирование алгоритма PAGS. Алгоритм PAGS исследовался на тестовых конфигурациях систем с однородной макроструктурой и на моделях кластерных ВС. Рассматривались макроструктуры вида: D_n-граф, гиперкуб, 2D-решетка. На рис. 4.18 представлен график зависимости времени работы алгоритма от количества N ЭМ в системе и ранга M формируемой подсистемы. Время работы приведено для процессора Intel Xeon E5345 и ВС с макроструктурой в виде 2D-решетки.

На рис. 4.19 представлены результаты формирования подсистем ранга $M = 64$ алгоритмами PAL, PAR (см. п. 1.4.4) и PAGS в системе с макроструктурой в виде 2D-решетки ($N = 1024$).

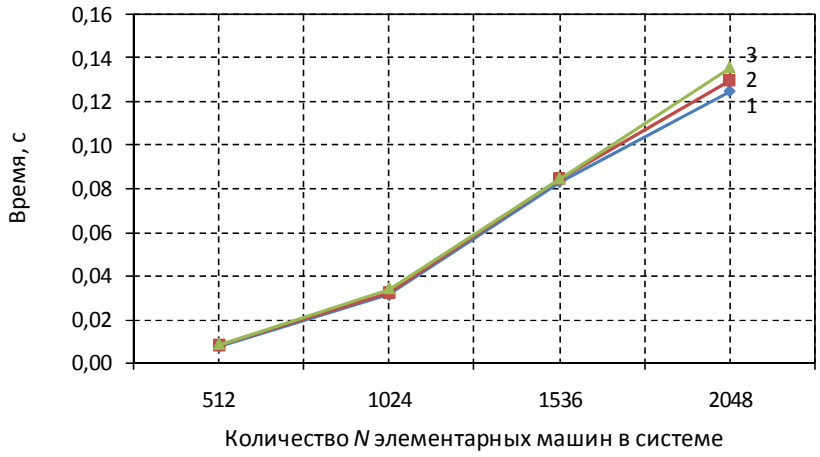


Рис. 4.18. Зависимость времени работы алгоритма PAGS от количества N ЭМ в системе и ранга M формируемой подсистемы:
1 – $M = 128$; 2 – $M = 256$; 3 – $M = 512$

На графиках показаны зависимости значений функции $L(X)$ от степени загруженности системы. Загруженность системы – это процент занятых ЭМ. Конфигурация системы с заданной загруженностью формировалась путем равновероятного случайного распределения по ЭМ параллельных программ со случайными рангами, равномерно распределенными в интервале [1, 64].

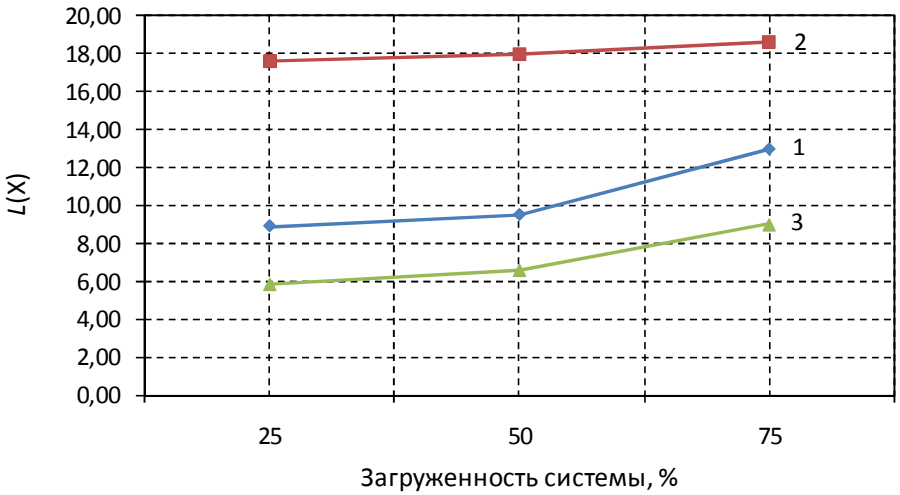


Рис. 4.19. Зависимость значений целевой функции $L(X)$
от загруженности системы:
1 – PAL; 2 – PAR; 3 – PAGS

На рис. 4.20 приведены результаты формирования подсистем в тестовых конфигурациях вычислительных кластеров. В качестве целевой функции использовался показатель $B(X)$.

В среднем алгоритм PAGS по значению целевой функции $L(X)$ формировал подсистемы в 1,46 раза лучше алгоритма PAL и в 2,6 раза лучше алгоритма PAR. На тестовых конфигурациях кластерных ВС отношения значений целевых функций от вложений, формируемых алгоритмами PAGS, PAL, PAR, в среднем принимали значения из интервалов

$$\frac{B(X_{PAGS})}{B(X_{PAL})} \in [1{,}05;\ 1{,}4],\ \frac{B(X_{PAGS})}{B(X_{PAR})} \in [1{,}2;\ 5].$$

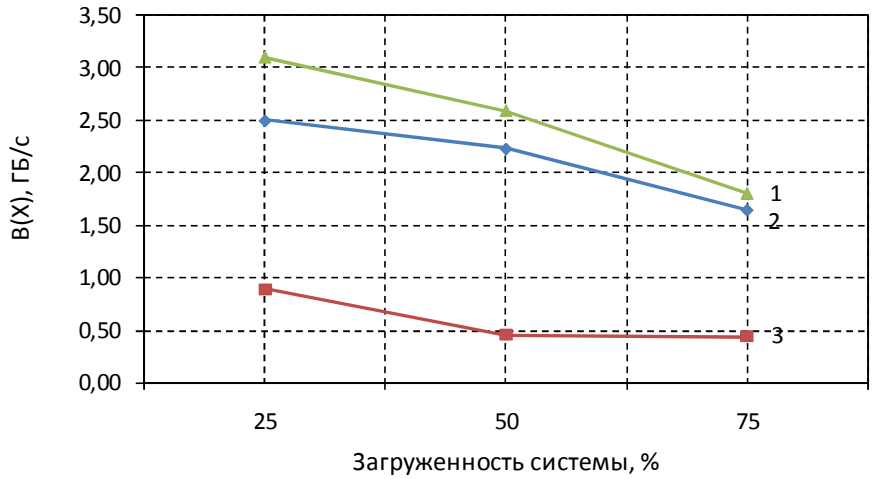


Рис. 4.20. Зависимость значений целевой функции $B(X)$
от загруженности системы: 1 – PAGS; 2 – PAL; 3 – PAR

На рис. 4.21 представлены графики зависимости значений $L(X)$ от ранга M формируемой подсистемы. Результаты приведены для системы $N = 1024$ с загруженностью 50%, макроструктура системы – 2D-решетка.

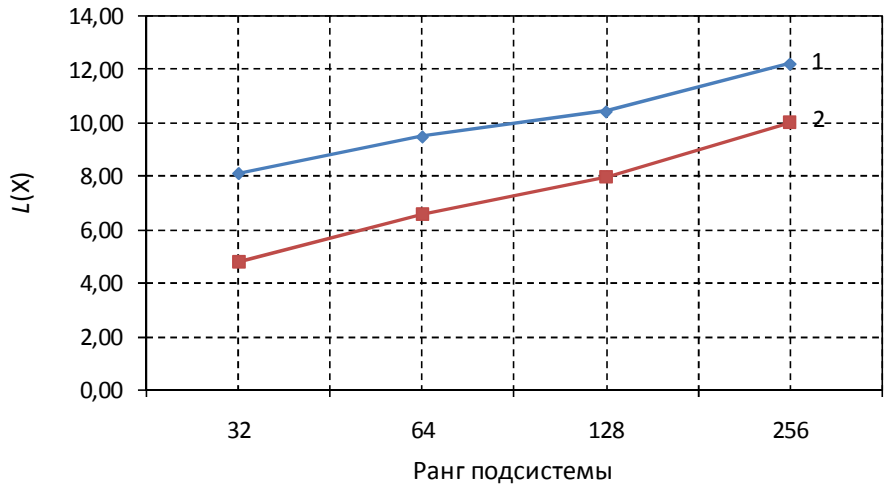


Рис. 4.21. Зависимость значений целевой функции $L(X)$
от ранга M формируемой подсистемы: $N = 1024$; 1 – PAL; 2 – PAGS

Из графиков на рис. 4.19 и 4.20 видно, что при увеличении загруженности системы и приближении ранга подсистемы к количеству свободных ЭМ, значение $L(X)$ увеличивается, а $B(X)$ уменьшается. Это объясняется сокраще-

нием возможных вариантов выбора ЭМ для включения в формируемую подсистему.

На рис. 4.22 приведены графики зависимости значений функции $L(X)$ от загруженности системы. Качество формируемых решений зависит от структурных характеристик системы. С уменьшением среднего диаметра структуры и увеличением загруженности системы качество формируемых подсистем возрастает незначительно.

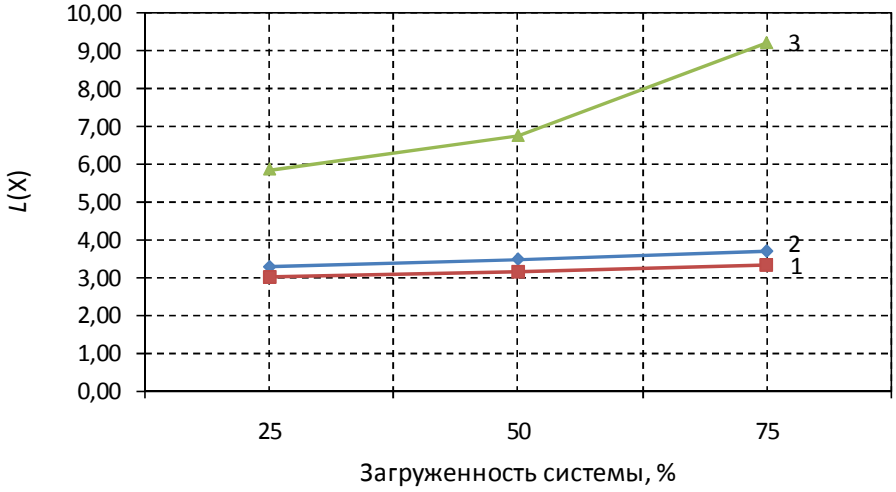


Рис. 4.22. Зависимость значений целевой функции $L(X)$
от загруженности системы: $N = 1024$; $M = 64$;
1 – D_5-граф {1024; 1, 2, 3, 4, 5}; 2 – 10D-куб; 3 – 2D-решетка

Моделирование алгоритма PADS. Для моделирования работы алгоритма PADS формировались Δ-однородные подсистемы пространственно-распределенных мультикластерных ВС. Параметры систем задавались как равномерно распределенные случайные числа из интервалов: $H \in [2; 32]$, $L \in [3; 4]$, $h_i \in [1; 2]$, $i \in \{1, 2, \ldots, H\}$, $n_{2k} \in [4; 128]$, $n_{3k} \in [0; 2]$, $c_{Lk} \in [2; 8]$, $b_i \in [25 \cdot 2^{20}; 8 \cdot 2^{30}]$, $\omega_i \in [2 \cdot 10^9; 12 \cdot 10^9]$.

В таблице 4.6 приведено время работы алгоритма по формированию подсистем в распределенной ВС $N = 103804$, $H = 16326$, $L = 4$. Время указано для процессора Intel Xeon E5345.

В качестве оценки эффективности формируемых алгоритмом подсистем рассматривался показатель $\delta = (B - \tilde{B})/\tilde{B}$, где B и $\tilde{B}$ – значения целевой функции от решения, получаемого алгоритмом PADS и случайным выделением подсистемы. По данным статистической обработки результатов моделирования построены оценки математического ожидания и среднеквадратического отклонения показателя δ: $M[\delta] = 10$, $\sigma[\delta] = 3$.

Таблица 4.6

Ранг *M* подсистемы	Время работы алгоритма PADS, с
4096	0,003271
8192	0,004590
16384	0,005475
32768	0,006536

Неоднородности каналов связи на различных уровнях коммуникационной среды распределенной ВС оказывает заметное влияние на качество формируемых подсистем. На рис. 4.23 представлен график зависимости значения $M[\delta]$ от величины Δb – отношения максимального значения пропускной способности каналов связи внутри кластеров (уровень 2) к максимальному значению пропускной способности каналов связи между кластерами (уровень 1).

Статистическая обработка результатов моделирования показала, что алгоритм PADS позволяет формировать эффективные подсистемы по значению показателя $B(X)$ в распределенных ВС с иерархически организованными коммуникационными средами. Алгоритм мало применим при формировании подсистем с рангом, не превышающим количества ЭМ в подсистемах.

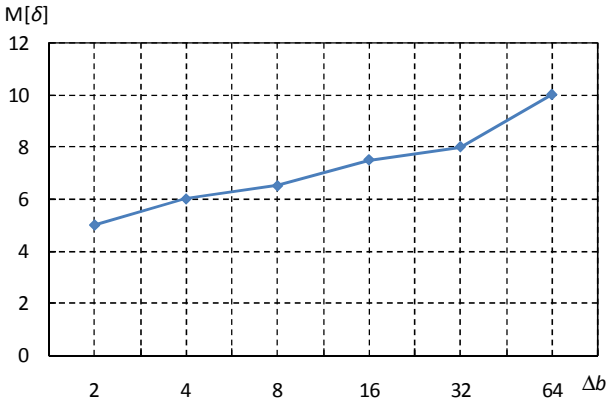


Рис. 4.23. Зависимость показателя $\mathrm{M}[\delta]$ от Δb:

$N = 51900$; $H = 8940$; $M = 512$

4.5. Выводы

1. Описана функциональная структура пространственно-распределенной мультикластерной ВС, её текущая конфигурация и программное обеспечение.

2. Для оптимизации вложения параллельных MPI-программ в кластерные ВС создан пакет MPITaskMap. Предложены средства анализа производительности параллельных MPI-программ и автоматизации формирования информационных графов. Применение описанных средств в композиции с известными средствами управления ресурсами кластерных ВС позволяет обеспечить эффективное функционирование вычислительных систем.

3. Адекватность используемых моделей и эффективность алгоритмов подтверждается результатами моделирования и натурных экспериментов. Алгоритмы могут быть рекомендованы для оптимизации вложения параллельных программ в распределенные ВС и формирования субоптимальных подсистем.

ЗАКЛЮЧЕНИЕ

Разработаны и исследованы модели, алгоритмы и программные средства оптимизации вложения параллельных программ в распределенные вычислительные системы (ВС).

1. Предложены и исследованы эвристические алгоритмы вложения параллельных программ в распределенные вычислительные системы.

1.1. Разработана математическая модель коммуникационных сред современных мультиархитектурных ВС, в которой учитывается их иерархическая организация и пропускные способности каналов связи между ресурсами (вычислительными узлами, элементарными машинами, процессорами и ядрами).

1.2. Предложены метод и эффективные алгоритмы вложения параллельных программ в иерархические ВС. Алгоритмы в сравнении со стандартными средствами сокращают время выполнения промышленных MPI-программ в 1,5 – 2 раза.

1.3. Созданы эвристические алгоритмы формирования в пределах ВС подсистем, обеспечивающих эффективную реализацию основных схем межмашинных обменов, и вложения в них параллельных программ. Алгоритмы применимы как для систем с однородными структурами сетей межмашинных связей, так и для мультиархитектурных ВС. Показано, что средние значения пропускных способностей каналов связи между элементарными машинами формируемых подсистем в 1,1 – 3 раза выше, чем при применении известных алгоритмов.

1.4. Осуществлено численное моделирование и проведены натурные эксперименты с разработанными алгоритмами. Оценена их эффективность, выработаны рекомендации к применению в системах управления ресурсами распределенных ВС и в библиотеках стандарта MPI.

2. Разработаны алгоритмы вложения параллельных программ в пространственно-распределенные вычислительные системы.

2.1. Построен стохастический алгоритм вложения параллельных программ в пространственно-распределенные ВС. В алгоритме учитываются производительность вычислительных ресурсов и время доставки данных до подсистем, выделенных для выполнения ветвей параллельных программ. Предложена параллельная версия алгоритма для большемасштабных систем, характеризующаяся ускорением близким к линейному.

2.2. Разработаны алгоритмы формирования в пределах пространственно-распределенных ВС однородных подсистем и вложения в них параллельных программ. Алгоритмы обладают полиномиальной трудоемкостью и позволяют приоритезировать выбор ресурсов по их производительности и скорости передачи данных по каналам связи между ними.

2.3. Проведено моделирование работы созданных алгоритмов. Показано, что время выполнения параллельных программ с вложением предложенными алгоритмами на один – два порядка меньше, чем при случайном распределении ветвей по элементарным машинам.

3. Созданы программные средства оптимизации вложения параллельных MPI-программ в кластерные ВС, которые, в частности, допускают интеграцию с современными системами пакетной обработки заданий и библиотеками стандарта MPI.

4. При непосредственном участии диссертанта создана пространственно-распределенная мультикластерная ВС, которая помимо стандартных средств параллельного программирования, оснащена разработанным инструментарием анализа производительности MPI-программ, позволяющим получать информацию о структуре информационных обменов между ветвями параллельных программ и автоматизировать процесс построения их информационных графов.

Основные результаты диссертации опубликованы в работах [36-59, 66, 96-98, 150-151].

ЛИТЕРАТУРА

1. Ахо, А. В. Структуры данных и алгоритмы / А. В. Ахо, В. Д. Хопкрофт, Дж. Д. Ульман ; пер. с англ. под ред. А. А. Минько. – Издательский дом “Вильямс”, 2001. – 384 с.
2. Бабаян, Б. А. Многопроцессорные ЭВМ и методы их проектирования / Б. А. Бабаян, А. В. Бочаров, А. С. Волин. – М. : Высшая школа, 1990. – 143 с.
3. Бетелин, В. Б. Архитектура цифровых процессоров обработки сигналов / В. Б. Бетелин [и др.]. – М. : РАН, 1993. – 20 с.
4. Балашов, Е. П. Микро и мини-ЭВМ учеб. пособие для ВУЗов / Е. П. Балашов, В.Л. Григорьев, Г.А. Петров. – Л. : Энергоатомиздат, 1984. – 376 с.
5. Балашов, Е. П. Микропроцессоры и микропроцессорные системы / Е. П. Балашов, Д. В. Пузанков, В. Б. Смолов. – М. : Радио и связь, 1981. – 326 с.
6. Барский, А. Б. Планирование параллельных вычислительных процессов / А. Б. Барский. – М.: Машиностроение, 1980. – 192 с.
7. Барский, А. Б. Параллельные процессы в вычислительных системах / А. Б. Барский. – М. : Изд-во “Радио и связь”, 1990. – 256 с.
8. Береснев, В. Л. Экстремальные задачи стандартизации / В. Л. Береснев, Э. Х. Гимади, В. Т. Дементьев. – Новосибирск : Наука, 1978. – 336 с.
9. Береснев, В. Л. Дискретные задачи размещения и полиномы от булевых переменных / В. Л. Береснев. – Новосибирск : ИМ СО РАН, 2005. – 408 с.
10. Бурцев, В. С. Параллелизм вычислительных процессов и развитие архитектур суперЭВМ / В. С. Бурцев. – М. : ИВВС РАН, 1997. – 352 с.
11. Бурцев, В. С. Супер-ЭВМ : сборник научных трудов / В. С. Бурцев. – М. : АН СССР, отдел вычислительной математики, 1992. – 95 с.
12. Вальковский, В. А. Распараллеливание алгоритмов и программ. Структурный подход / В. А. Вальковский. – М. : Радио и связь, 1989. – 176 с.

13. Васильев, В. В. Многопроцессорные вычислительные структуры для анализа задач на сетях / В. В. Васильев, А. Г. Додонов // Проблемы электроники и вычислительной техники. – 1976. – №4. – С. 85-97.
14. Водяхо, А. И. Высокопроизводительные системы обработки данных / А. И. Водяхо, Н. Н. Горпец, Д. В. Пузанков. – М. : Высшая школа, 1997. – 304 с.
15. Воеводин, В. В. Математические модели и методы в параллельных процессах / В. В. Воеводин. – М. : Наука, 1986. – 296 с.
16. Воеводин, В. В. Параллельные вычисления / В. В. Воеводин, Вл. В. Воеводин. – СПб. : БХВ-Петербург, 2002. – 608 с.
17. Гергель, В. П. Основы параллельных вычислений для многопроцессорных вычислительных систем / В. П. Гергель, Р. Г. Стронгин. – Нижний Новгород : Изд-во ННГУ, 2003. – 184 с.
18. Гери, М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон ; пер. с англ. – М. : Мир, 1982. – 416 с.
19. Гимади, Э. Х. Дискретные экстремальные задачи принятия решений / Э. Х. Гимади, Н. И. Глебов. – Новосибирск : НГУ, 1991. – 76 с.
20. Грэхэм, Р. Конкретная математика. Основания информатики / Р. Грэхэм, Д. Кнут, О. Поташник ; пер. с англ. – 2-е изд. – М. : Мир, 2006. – 703 с.
21. Додонов, А. Г. Введение в теорию живучести вычислительных систем / А. Г. Додонов, М. Г. Кузнецова, Е. С. Горбачик. – Киев : Наук. Думка, 1990. – 180 с.
22. Дмитриев, Ю. К. Вычислительные системы из мини-ЭВМ / Ю. К. Дмитриев, В. Г. Хорошевский. – М. : Радио и связь, 1982. – 304 с.
23. Евдокимов, В. Ф. Параллельные вычислительные структуры на основе разрядных методов / В. Ф. Евдокимов, А. И. Стасюк. – К.: Наукова думка, 1987. – 311 с.

24. Евреинов, Э. В. О возможности построения вычислительных систем в условиях запаздывания сигналов / Э. В. Евреинов // Вычислительные системы. – 1962. – № 3. – С. 3-16.
25. Евреинов, Э. В. Однородные вычислительные системы, структуры и среды / Э. В. Евреинов. – М.: Радио и связь, 1981. – 208 с.
26. Евреинов, Э. В. Однородные универсальные вычислительные системы высокой производительности / Э. В. Евреинов, Ю. Г. Косарев. – Новосибирск : Наука. Сибирское отд-е, 1966. – 308 с.
27. Евреинов, Э.В. Однородные вычислительные системы / Э.В. Евреинов, В.Г. Хорошевский. – Новосибирск : Наука. Сибирское отд-е, 1978. – 319 с.
28. Каляев, А. В. Параллельный компьютер с программируемой под структуру задачи архитектурой / А. В. Каляев [и др.] // Труды Шестого Международного семинара “Распределённая обработка информации”. – Новосибирск: СО РАН, 1998. – С. 25-29.
29. Каляев, А. В. Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений / А. В. Каляев, И. И. Левин. – М. : Янус-К, 2003. – 380 с.
30. Каляев, И. А. Реконфигурируемые мультиконвейерные вычислительные структуры / И. А. Каляев ; под. ред. И. А. Каляева. – Ростов н/Д. : ЮНЦ РАН, 2008. – 320 с.
31. Кормен, Т. Х. Алгоритмы: построение и анализ, 2-е издание / Т. Х. Кормен [и др.] ; пер. с англ. под ред. И. В. Красикова. – М. : Издательский дом “Вильямс”, 2005. – 1296 с.
32. Корнеев, В. В. Архитектура вычислительных систем с программируемой структурой / В. В. Корнеев. – Новосибирск : Наука, 1985. – 164 с.
33. Корнеев, В. В. Вычислительные системы / В. В. Корнеев. – М. : Гелиос АРВ, 2004. – 512 с.
34. Косарев, Ю. Г. Распараллеливание по циклам / Ю. Г. Косарев // Вычислительные системы. – 1967. – Вып. 24. – С. 3-20.

35. Кнут, Д. Э. Искусство программирования, том 1. Основные алгоритмы / Д. Э. Кнут ; пер. с англ. – М. : Издательский дом “Вильямс”, 2004. – 720 с.
36. Курносов, М. Г. Опыт построения кластерных вычислительных систем с удаленной загрузкой узлов / М. Г. Курносов // Материалы пятого Международного научно-практического семинара “Высокопроизводительные параллельные вычисления на кластерных системах”. – Нижний Новгород: Изд-во ННГУ, 2005. – С. 149-154.
37. Курносов, М. Г. Задача стохастически оптимального распределения машин по терминалам распределенной вычислительной системы / М. Г. Курносов // Материалы XLIV Международной научно студенческой конференции “Студент и научно-технический прогресс”. – Новосибирск: Изд-во НГУ, 2006. – С. 16-17.
38. Курносов, М. Г. Построение учебных кластерных вычислительных систем с удаленной загрузкой узлов / М. Г. Курносов // Вестник СибГУТИ “Тенденции модернизации высшего образования”. – Новосибирск: Изд-во СибГУТИ, 2006. – С. 52-54.
39. Курносов, М. Г. Об одной задаче организации стохастически оптимального функционирования распределенных вычислительных систем / М. Г. Курносов // Материалы Российской научно-технической конференции “Информатика и проблемы телекоммуникаций”. – Новосибирск: Изд-во НГУ, 2006. – С. 63-66.
40. Курносов, М. Г., Об одной задаче оптимального назначения ветвей параллельной программы на процессорные ядра распределенной вычислительной системы / М. Г. Курносов, А. В. Половинкин // Материалы XLV Международной научной студенческой конференции “Студент и научно-технический прогресс”. – Новосибирск: Изд-во НГУ, 2007. – С. 41.
41. Курносов, М. Г. Планирование выполнения параллельных программ набора на распределенной вычислительной системе / М. Г. Курносов, С. В. Рыбалко // Материалы XLV Международной научной студенческой

конференции “Студент и научно-технический прогресс”. – Новосибирск: Изд-во НГУ, 2007. – С. 42-43.

42. Курносов, М. Г. Задача назначения ветвей параллельной программы на процессорные ядра распределенной вычислительной системы / М. Г. Курносов, А. В. Половинкин // Материалы Российской научно-технической конференции “Информатика и проблемы телекоммуникаций”. – Новосибирск: Изд-во СибГУТИ, 2007. – С. 270-272.
43. Курносов, М. Г. Синтез расписания запуска параллельных программ на распределенной вычислительной системе / М. Г. Курносов, С. В. Рыбалко // Материалы Российской научно-технической конференции “Информатика и проблемы телекоммуникаций”. – Новосибирск: Изд-во СибГУТИ, 2007. – С. 272-274.
44. Курносов, М. Г. Оптимизация выполнения параллельных MPI-программ на мультикластерных вычислительных системах / М. Г. Курносов // Материалы четвертой Сибирской школы-семинара по параллельным и высокопроизводительным вычислениям. – Томск: Изд-во ТГУ, 2007. – С. 168-175.
45. Курносов, М. Г. Назначение ветвей параллельной программы на процессорные ядра распределенной вычислительной системы / М. Г. Курносов // Материалы Международной научно-технической конференции “Многопроцессорные вычислительные и управляющие системы”. – Таганрог: Изд-во ТТИ ЮФУ, 2007. – Т. 1. – С. 227-231.
46. Курносов, М. Г. Алгоритмы распределения ветвей параллельных программ по процессорным ядрам мультикластерных вычислительных систем / М. Г. Курносов // Материалы Всероссийской научной конференции “Наука. Технологии. Инновации”. – Новосибирск: Изд-во НГТУ, 2007. – С. 258-260.
47. Курносов, М. Г. Разработка программного средства анализа параллельных MPI-программ / М. Г. Курносов // Тезисы докладов VIII Всероссийской

конференции молодых ученых по математическому моделированию и информационным технологиям. – Новосибирск: Изд-во ИВТ СО РАН, 2007. – С. 101.

48. Курносов, М. Г. Об оптимизации распределения ветвей параллельных MPI-программ по процессорным ядрам вычислительного кластера / М. Г. Курносов, А. А. Пазников // Материалы седьмой Международной конференции-семинара “Высокопроизводительные вычисления на кластерных системах”. – Нижний Новгород: Изд-во ННГУ, 2007. – С. 218-225.
49. Курносов, М. Г. Параллельный алгоритм отображения информационного графа MPI-программы на процессорные ядра распределенной вычислительной системы / М. Г. Курносов // Труды Международной научной конференции “Параллельные вычислительные технологии”. – Челябинск: Изд-во ЮУрГУ, 2008. – С. 532.
50. Курносов, М. Г. Оптимизация вложения параллельных программ в однородные вычислительные кластеры / М. Г. Курносов // Материалы IV Всероссийской научно-практической конференции “Молодежь и современные информационные технологии”. – Томск: Изд-во ТПУ, 2008. – С. 43-44.
51. Курносов, М. Г. Применение средств анализа производительности MPI-программ при подготовке специалистов в области параллельных вычислительных технологий / М. Г. Курносов // Тезисы докладов XLIX научно-методической конференции “Проблемы перехода на многоуровневую систему образования”. – Новосибирск: Изд-во СибГУТИ, 2008. – С. 39.
52. Курносов, М. Г. Вложение параллельных программ в однородные вычислительные системы / М. Г. Курносов // Материалы Российской научно-технической конференции “Информатика и проблемы телекоммуникаций”. – Новосибирск: Изд-во СибГУТИ, 2008. – Т. 1 – С. 138-140.

53. Курносов, М. Г. Средства оптимизации вложения MPI-программ в вычислительные кластеры / М. Г. Курносов, А. Н. Парыгин // Материалы Российской научно-технической конференции “Информатика и проблемы телекоммуникаций”. – Новосибирск: Изд-во СибГУТИ, 2008. – Т. 1. – С. 140-141.
54. Курносов, М. Г. Применение иерархических методов разбиения графов для вложения параллельных программ в вычислительные системы / М. Г. Курносов // Материалы III Международной научно-технической конференции “Инфокоммуникационные технологии в науке, производстве и образовании (Инфоком-3)”. – Ставрополь: СевКавГТУ, 2008. – Ч. 1. – С. 76-82.
55. Курносов, М. Г. Об организации функционирования вычислительных систем с оптимизацией вложения параллельных программ / М. Г. Курносов // Материалы XLVI Международной научной студенческой конференции “Студент и научно-технический прогресс”. – Новосибирск: Изд-во НГУ, 2008. – С. 217-218.
56. Курносов, М. Г. Организация функционирования вычислительных систем с учетом архитектуры их коммуникационных сред / М. Г. Курносов // Материалы II Международной научной студенческой конференции “Научный потенциал студенчества – будущему России”. – Ставрополь: СевКавГТУ, 2008. – Т. 3. – Режим доступа: http://science.ncstu.ru/conf/past/2008/stud/informatics/16.pdf.
57. Курносов, М. Г. Оптимизация вложения параллельных программ при диспетчеризации заданий в GRID-системах / М. Г. Курносов // Материалы Всероссийской конференции “Современные информационные технологии для научных исследований”. – Магадан: СВНЦ ДВО РАН, 2008. – С. 87-89.
58. Курносов, М. Г. Организация функционирования вычислительных систем с учетом их структур / М. Г. Курносов // Тезисы докладов Седьмой Рос-

сийской конференции с международным участием “Новые информационные технологии в исследовании сложных структур (ICAM-2008)”. – Томск : НТЛ, 2008. – 69 с.

59. Курносов, М. Г. Эвристический алгоритм формирования подсистем в вычислительных системах // Материалы 9 Международной научно-практической конференции “Искусственный интеллект-2008. Интеллектуальные системы”. – Изд-во: НИИ МВС ЮФУ, 2008. – Ч. 2. – С. 346-351.
60. Левин, В. К. Высокопроизводительные мультипроцессорные системы / В. К. Левин // Информационные технологии и вычислительные системы. – 1995. – №1. – С. 12-21.
61. Левин, В. К. Современные суперкомпьютеры семейства МВС [Электронный ресурс] / В. К. Левин. – Электрон. текст. дан. – Б. м., 2002. – Режим доступа : http://www.parallel.ru/mvs/levin.html.
62. Левин, И. И. Алгоритм коммутации элементов многопроцессорной системы со структурно-процедурной организацией вычислений / И. И. Левин, Л. М. Сластен // Материалы Всероссийской научно-технической конференции “Методы и средства обработки информации”, 2003.
63. Левитин, А. В. Алгоритмы: введение в разработку и анализ / А. В. Левитин ; пер. с англ. под ред. И. В. Красикова. – М. : Издательский дом “Вильямс”, 2006. – 576 с.
64. Лопатин, А. С. Метод отжига / А. С. Лопатин. – Стохастическая оптимизация в информатике. – 2005. – Вып. 1. – С. 133-149.
65. Малышкин, В. Э. Параллельное программирование мультикомпьютеров / В. Э. Малышкин, В. Д. Корнеев. – Новосибирск : НГТУ, 2006. – 296 с.
66. Мамойленко, С. Н. Использование мультикластерной вычислительной системы для подготовки специалистов в области параллельных вычислительных технологий / С. Н. Мамойленко, М. Г. Курносов [и др.] // Тезисы докладов XLIX научно-методической конференции “Проблемы перехода

на многоуровневую систему образования". – Новосибирск: Изд-во Сиб-ГУТИ, 2008. – С. 38.

67. Миллер, Р. Последовательные и параллельные алгоритмы: Общий подход / Р. Миллер, Л. Боксер ; пер. с англ. – М. : БИНОМ. Лаборатория знаний, 2006. – 406 с.

68. Миренков, Н. Н. Параллельное программирование для многомодульных вычислительных систем / Н. Н. Миренков. – М. : Радио и связь, 1989. – 319 с.

69. Монахов, О. Г. Параллельные системы с распределенной памятью: структуры и организация взаимодействий / О. Г. Монахов, Э. А. Монахова. – Новосибирск : Изд-во СО РАН, 2000. – 242 с.

70. Монахов, О. Г. Параллельные системы с распределенной памятью: управление ресурсами и заданиями / О. Г. Монахов, Э. А. Монахова. – Новосибирск : Изд-во ИВМиМГ СО РАН, 2001. – 168 с.

71. Павский, В. А. Организация функционирования однородных вычислительных систем и стохастическое программирование / В. А. Павский, В. Г. Хорошевский // Вычислительные системы. – 1975. – Вып. 63. – С. 3-16.

72. Панфилов, И. В. Вычислительные системы / И. В. Панфилов, А. М. Половко. – М. : Изд-во "Советское радио", 1980. – 302 с.

73. Поспелов, Д. А. Введение в теорию вычислительных систем / Д. А. Поспелов. – М. : Изд-во "Советское радио", 1972. – 280 с.

74. Пискунов, С. В. Построение микропрограммных описаний однородных вычислительных структур / С. В. Пискунов // Моделирование дискретных управляющих и вычислительных систем. – Свердловск : СвердГУ, 1981. – С. 55-57.

75. Прангишвили, И. В. Параллельные вычислительные системы с общим управлением / И. В. Прангишвили, С. Я. Виленкин, И. Л. Медведев. – М. : Энергопромиздат, 1983. – 313 с.

76. Прангишвили, И. В. Многопроцессорные и локальные сети микро-ЭВМ в распределенных системах управления / И. В. Прангишвили. – М. : Энергоатомиздат, 1985. – 272 с.
77. Пухов, Г. Е. Разрядно-аналоговые вычислительные системы / Г. Е. Пухов, В. Ф. Евдокимов, М. В. Синьков. – М.: Советское радио, 1978. – 255 с.
78. Сайт проекта MPICH2 [Электронный ресурс]. – Режим доступа : http://www.mcs.anl.gov/research/projects/mpich2/, свободный.
79. Сайт проекта OpenMPI [Электронный ресурс]. – Режим доступа : http://www.open-mpi.org, свободный.
80. Сайт проекта Top500 [Электронный ресурс]. – Режим доступа : http://www.top500.org, свободный.
81. Сайт проекта TORQUE [Электронный ресурс]. – Режим доступа : http://www.clusterresources.com/pages/products/torque-resource-manager.php, свободный.
82. Сайт проекта VampirTrace [Электронный ресурс]. – Режим доступа : http://tu-resden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/software_werkzeuge_zur_unterstuetzung_von_programmierung_und_optimierung/vampirtrace?set_language=en&cl=en, свободный.
83. Таненбаум, Э. Распределенные системы: принципы и парадигмы / Эндрю Таненбаум, Стен М. Ван; пер. с англ. А. Леонтьев. – СПб. : Питер, 2003. – 877 с.
84. Тарков, М.С. Вложение структур параллельных программ в структуры живучих распределенных вычислительных систем / М. С. Тарков // Автометрия. – 2003. – Том 39, № 3. – С. 84-96.
85. Таха, Х. А. Введение в исследование операций, 7-е издание / Х. А. Таха ; пер. с англ. под ред. А. А. Минько. – М. : Издательский дом “Вильямс”, 2005. – 912 с.
86. Томилин, А. Н. Применение метода математического моделирования к выбору структурной схемы машины БЭСМ-6 и разработки программы

диспетчера машины БЭСМ-6: автореф. дис. ... канд. физ.-мат. наук. / Александр Николаевич Томилин. – М., 1969. – 23 с.

87. Топорков, В. В. Модели распределенных вычислений / В. В. Топорков. – М. : ФИЗМАТЛИТ, 2004. – 320 с.

88. Фортов, В. Е. Создание и применение высокопроизводительных вычислений на базе высокопроизводительных вычислительных технологий / В. Е. Фортов, Г. И. Савин, В. К. Левин // Информационные технологии и вычислительные системы. – 2001. – № 1. – С. 3-10.

89. Харари, Ф. Теория графов / Ф. Харари ; пер. с англ. под ред. Г. П. Гаврилова. – М. : КомКнига, 2006. – 296 с.

90. Хетагуров, Я. А. Основы проектирования управляющих вычислительных систем / Я. А. Хетагуров. – М. : Радио и связь, 1991. – 287 с.

91. Хокни, Р. Параллельные ВС. Архитектура, программирование и алгоритмы / Роджер Хокни, Карл Джессоуп; пер. с англ. Д. Н. Абашкина. – М. : Радио и связь, 1986. – 392 с.

92. Хокни, Р. Параллельные ЭВМ / Роджер Хокни, Карл Джессоуп; пер. с англ. Д. Н. Абашкина. – М. : Радио и связь, 1986. – 390 с.

93. Хорошевский, В. Г. Вычислительная система МИКРОС / В. Г. Хорошевский. – Новосибирск : Препринт ИМ СО АН СССР 38 (ОВС-19), 1983. – 52 с.

94. Хорошевский, В. Г. Инженерный анализ функционирования вычислительных машин и систем / В. Г. Хорошевский. – М. : Радио и связь, 1987. – 256 с.

95. Хорошевский, В. Г. Архитектура вычислительных систем / В. Г. Хорошевский. – М. : МГТУ им. Н. Э. Баумана, 2008. – 520 с.

96. Хорошевский, В. Г. Алгоритмы распределения ветвей параллельных программ по процессорным ядрам вычислительных систем / В. Г. Хорошевский, М. Г. Курносов // Автометрия. – 2008. – Т. 44, № 2. – С. 56-67.

97. Хорошевский, В. Г. Моделирование алгоритмов вложения параллельных программ в структуры распределенных вычислительных систем / В. Г. Хорошевский, М. Г. Курносов // Труды Международной научной конференции “Моделирование-2008” (Simulation-2008). – Киев: Изд-во ИПМЭ им. Г.Е. Пухова, 2008. – Т. 2. – С. 435-440.

98. Хорошевский, В. Г. Архитектурные концепции, анализ и организация функционирования вычислительных систем с программируемой структурой / В. Г. Хорошевский, С. Н. Мамойленко, М. Г. Курносов // Труды Международной научно-технической конференции “Информационные технологии и математическое моделирование систем”. – М.: Изд-во РАН, 2008.

99. Элементы параллельного программирования / В. А. Вальковский, В. Е. Котов, А. Г. Марчук, Н. Н. Миренков. – М.: Радио и связь, 1983. – 240 с.

100. Эндрюс, Г. Основы многопоточного, параллельного и распределенного программирования / Г. Эндрюс; пер. с англ. А. С. Подосельника и др. – М. : Вильямс, 2003. – 512 с.

101. Языки и параллельные ЭВМ : сб. ст. / А. А. Самарский. – М. : Наука, 1990. – 91 с.

102. Яненко, Н. Н. Параллельные вычисления в задачах математической физики на вычислительных системах с программируемой структурой / Н. Н. Яненко, В. Г. Хорошевский, А. Д. Рычков // Электронное моделирование. – 1984. – Т. 6, № 1. – С. 3-8.

103. Agarwal, T. Topology-aware task mapping for reducing communication contention on large parallel machines / T. Agarwal [et al.] // Parallel and Distributed Processing Symposium. – 2006. – P. 10

104. Ahmad, I. A Parallel Algorithm for Optimal Task Assignment in Distributed Systems / I. Ahmad, M. Kafil // Proceedings of the 1997 Advances in Parallel and Distributed Computing Conference. – 1997. – P. 284.

105. Baraglia, R. A static mapping heuristic for mapping parallel applications to heterogeneous computing systems / R. Baraglia, R. Ferrini, P. Ritrovato // Concurrency and Computation: Practice & Experience. – Chichester, 2005. – Vol. 17, № 13. – P. 1579-1605.

106. Bani-Mohammed, S. An efficient non-contiguous processor allocation strategy for 2D mesh connected multicomputers / S. Bani-Mohammed [et al.] // Information Sciences: an International Journal. – 2007. – Vol. 177, № 14. – P. 2867-2883.

107. Barnard, S. T. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems / S. T. Barnard, H. D. Simon. – Concurrency - Practice and Experience. – 1994. – Vol. 5, № 2. – P. 107-117.

108. Bender, M. A. Communication-Aware Processor Allocation for Supercomputers / M. A. Bender [et al.] // Algorithmica. – 2008. – Vol. 50, № 2. – P. 279-298.

109. Bokhari, S. H. On the mapping problem / S. H. Bokhari // IEEE Transactions on Computers. – 1981. – Vol. 30, №3. – P. 207-214.

110. Bouvry, P. Mapping and load balancing on distributed memory systems / P. Bouvry [et al.] // Proceedings of Microprocessing. – Budapest, 1994.

111. Bui, T. N. Genetic algorithm and graph partition / T. N. Bui, B. R. Moon // IEEE Transactions on Computers. – 1996. – Vol. 45, № 7. – P. 841-855.

112. Chen, Hu. MPIPP: An Automatic Profileguided Parallel Process Placement Toolset for SMP Clusters and Multiclusters / Hu Chen // Proceedings of the 20th annual international conference on Supercomputing. – 2006. – P. 353-360.

113. Chen, S. A fast recursive mapping algorithm / S. Chen, M. M. Eshaghian // Concurrency: Practice and Experience. – 1995. – Vol. 7, № 5. – P. 391-409.

114. Cho, S.-Y. Dynamic task assignment in heterogeneous linear array networks for metacomputing / S.-Y. Cho, K. H. Park // Heterogeneous Computing Workshop. – 1994. – P. 66-71.

115. Choo, H. Processor Scheduling and Allocation for 3D Torus Multicomputer Systems / H. Choo [et al.] // IEEE Transactions on Parallel and Distributed Systems. – 2000. – Vol. 11, № 5. – P. 475–484.

116. Coli, M. Global execution time minimization by allocating tasks in parallel systems / M. Coli, P. Palazzari // Proceedings of the 3rd Euromicro Workshop on Parallel and Distributed Processing. – 1995. – P. 91.

117. De Rose, C. A. F. Distributed algorithms for processor management in message-passing / C. A. F. De Rose, H.-U. Heiss // In Proc. 12th Int. Symp. on Computer and Information Sciences. – 1997. – P. 221-229.

118. De Rose, C. A. F. Distributed processor allocation in large PC clusters / C. A. F. De Rose [et al.] // High-Performance Distributed Computing. – 2000. – P. 288-289.

119. Distributed and parallel systems: cluster and GRID computing / ed. by Z. Juhasz [et al.]. – Boston : Springer. – 212 p.

120. Dongarra, J. Sourcebook of parallel computing / J. Dongarra [et al.]. – San Francisco : Morgan Kaufmann Publ., 2003. – 842 p.

121. El-Rewini, H. Advanced computer architecture and parallel processing / H. El-Rewini, M. Abd-El-Barr. – New Jersey : John Wiley & Sons, 2005. – 272 p.

122. Ercal, F. Task allocation onto a hypercube by recursive mincut bipartitioning / F. Ercal, J. Ramanujan, P. Sadayappan // Journal of Parallel and Distributed Computing. – 1990. – Vol. 10, № 1, – P. 35-44.

123. Fiduccia, C. M. A linear-time heuristic for improving network partitions / C. M. Fiduccia, R. M. Mattheyses // Proc. of conference “Design Automation”. – 1982. – P. 175-181.

124. Flynn, M. Very high-speed computing system / M. Flynn // Proc. of IEEE, 1966. – № 54. – P. 1901-1909.

125. Flynn, M. Some Computer Organisations and Their Effectiveness // IEEE Trans. Computers. – 1972. – Vol. 21, № 9. – P. 948-960.

126. Foster, I. Globus: A Metacomputing Infrastructure Toolkit / I. Foster, C. Kesselman. – 1997. – Intl J. Supercomputer Applications. – Vol. 11, №3. – P. 115–128.

127. Foster, I. The Grid: Blueprint for a New Computing Infrastructure / I. Foster, C. Kesselman. – Morgan-Kaufmann, 1998.

128. Foster, I. The Anatomy of the Grid: Enabling Scalable Virtual Organizations / I. Foster // Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing, 2001. – P. 1-4.

129. Foster, I. What is the Grid? A Three Point Checklist [Электронный ресурс]. –Режим доступа : http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf, свободный.

130. Foster, I. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration / I. Foster [et al.] // Global Grid Forum, 2002.

131. Garey, M. Some simplified NP-complete graph problems / M. Garey, D. Johnson, L. Stockmeyer. – Theoretical Computer Science. – 1976. – Vol. 1. – P. 237-267.

132. Grama, A. Introduction to parallel computing / A. Grama. – Harlow : Addison Wesley, 2003. – 856 p.

133. Heiss, H.-U. Mapping large parallel simulation programs to multicomputer systems / H.-U. Heiss, M. Dormanns // Proceedings of the 1994 simulation multiconference on Grand challenges in computer simulation. – 1994. – P. 285-290.

134. Hendrickson, B. A multilevel algorithm for partitioning graphs / B. Hendrickson, R. Leland // Proc. of ACM/IEEE conference on Supercomputing. – San Diego : IEEE Press, 1995.

135. Hluchy, L. Static Mapping Methods for Processor Networks [Электронный ресурс] / L. Hluchy, M. Dobrovodsky, M. Dobrucky // CiteSeer.IST, 2008. – Режим доступа : http://citeseer.ist.psu.edu/472693.html.

136. Hockney, R.. Parallel Computers 2: Architecture, Programming and Algorithms / R. Hockney, K. Jesshope. – Philadelphia : Hilger, 1988.

137. Hockney, R. The communication challenge for MPP: Intel Paragon and Meiko CS-2 / R. Hockney. – Parallel Computing. – 1994. – Vol. 20, № 3. – P. 389-398.

138. Hsu, H.-S. Task Allocation on a Network of Processors / H.-S. Hsu [et al.] // IEEE Transactions on Computers. – 2000. – Vol. 49, № 12. – P. 1339-1353.

139. Huang, K.-C. An Integrated Processor Allocation and Job Scheduling Approach to Workload Management on Computing Grid / K.-C. Huang [et al.] // Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. – 2006. – P. 703-709.

140. Jacob, J. C. Task Spreading and Shrinking on Multiprocessor Systems and Networks of Workstations / J. C. Jacob, S.-Y. Lee // IEEE Transactions on Parallel and Distributed Systems. – 1999. – Vol. 10, № 10. – P. 1082-1101.

141. Jose, A. An approach to mapping parallel programs on hypercube multiprocessors / A. Jose // Proceedings of Workshop Parallel and Distributed Processing. – 1999. – P. 221-225.

142. Ingber, L. Simulated annealing: Practice versus theory / L. Ingber. – Mathl. Comput. Modelling. – 1993. – Vol. 18, № 11. – P. 29-57.

143. Kafil, M. Optimal Task Assignment Heterogeneous Distributed Computing Systems / M. Kafil, I. Ahmad // IEEE Concurrency. – 1998. – Vol. 6, № 3. – P. 42-51.

144. Kalinowski, T. Solving the mapping problem with a genetic algorithm on the MasPar-1 / T. Kalinowski // Proceedings of the First International Conference on Massively Parallel Computing Systems. – 1994. – P. 370-374.

145. Kartik, S. Task Allocation Algorithms for Maximizing Reliability of Distributed Computing Systems / S. Kartik, C. S. R. Murthy // IEEE Transactions on Computers. – 1997. – Vol. 46, № 6. – P. 719-724.

146. Karypis, G. Multilevel k-way partitioning scheme for irregular graphs / G. Karypis, V. Kumar // Journal of Parallel and Distributed computing. – 1998. – Vol. 48. – P. 96-129.

147. Karypis, G. Analysis of multilevel graph partitioning / G. Karypis, V. Kumar // Proc. of ACM/IEEE conference on Supercomputing. – San Diego : IEEE Press, 1995.

148. Karypis, G. A fast and high quality multilevel scheme for partitioning iregular graphs / G. Karypis, V. Kumar // Journal on Scientific computing. – 1998. – Vol. 20, № 1. – P. 359-392.

149. Khan, M. S. Fast graph partitioning algorithms / M. S. Khan, K. F. Li // Proc. of Int. IEEE conference "Communications, Computers, and Signal Processing". – Victoria : IEEE Press, 1995. – P. 337-342.

150. Khoroshevsky, V.G. Space-distributed multi-cluster computer system for training in parallel computational technologies / V. G. Khoroshevsky [et al.] // Proceedings of 7th International Siberian Workshop and Tutorial (EDM-2006). – Erlagol: IEEE Press, 2006. – P. 218-219.

151. Khoroshevsky, V. G. Algorithms for Assigning Parallel Program Branches to Computer System Processor Cores / V. G. Khoroshevsky, M. G. Kurnosov // Optoelectronics, Instrumentation and Data Processing. – 2008. – Vol. 44, № 2. – P. 135-143.

152. Kirkpatrick, S. Optimization by Simulated Annealing / S. Kirpatrick, C. D. Gelatt, M. P. Vecchi. – Science. – 1983. – Vol. 220, № 4598. – P. 671-680.

153. Koziris, N. An efficient algorithm for the physical mapping of clustered taskgraphs onto multiprocessor architectures / N. Koziris [et al.] // Proceedings of Parallel and Distributed Processing. – 2000. – P. 406-413.

154. Lastovetsky, A. L. Parallel Computing on Heterogeneous Networks / A. L. Lastovetsky. – John Wiley & Sons, 2003. – 423 p.

155. Lee, C. On the mapping problem using simulated annealing / C. Lee, L. Bic // Proceedings of Computers and Communications. – 1989. – P. 40-44.

156. Lee, C.-H. Optimal task assignment in homogeneous networks / C.-H. Lee, K. G. Shin // IEEE Transactions on Parallel and Distributed Systems. – 1997. – Vol. 8, № 2. – P. 119-129.

157. Leiserson, C. E. Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing / C. E. Leiserson // Proc. of. ICPP, 1985. – P. 393-402.

158. Leung, V. J. Processor Allocation on Cplant: Achieving General Processor Locality Using One-Dimensional Allocation Strategies / V. J. Leung [et al.] // Proceedings of the IEEE International Conference on Cluster Computing. – 2002. – P. 296.

159. Lo, V. Noncontiguous Processor Allocation Algorithms for Mesh-Connected Multicomputers / V. Lo [et al.] // IEEE Transactions on Parallel and Distributed Systems. – 1997. – Vol. 8, № 7. – P. 712-726.

160. Mache, J. Minimizing Message-Passing Contention in Fragmentation-Free Processor Allocation / J. Mache [et al.] // In Proceedings of the 10th International Conference on Parallel and Distributed Computing Systems. – 1997. – P. 120-124.

161. Malony, A. D. The open trace format (OTF) and open tracing for HPC / A. D. Malony, W. E. Nagel // Proceedings of the 2006 ACM/IEEE conference on Supercomputing, 2006. – P. 24.

162. Miquel, A. Clustering and reassignment-based mapping strategy for message-passing architectures / A. Miquel [et al.] // Journal of Systems Architecture. – 2003. – Vol. 48, № 8-10. – P. 267-283.

163. Moh, S. Mapping strategies for switch-based cluster systems of irregular topology / S. Moh [et al.] // Proceedings of Parallel and Distributed Systems. – 2001. – P. 733-740.

164. Optimizing task layout on the Blue Gene/L supercomputer / G. Bhanot [et al.] // IBM Journal of Research and Development. – 2005. – Vol. 49, № 2. – P. 489-500.

165. Perego, R. A mapping heuristic for minimizing network contention / R. Perego // Journal of Systems Architecture. – 1998. – Vol. 45, № 1. – P. 65-82.

166. Perego, R. Minimizing network contention for mapping tasks onto massively parallel computers / R. Perego, G. De Petris // Proceedings of the 3rd Euromicro Workshop on Parallel and Distributed Processing. – 1995. – P. 210.

167. Sadayappan, P. Cluster-partitioning approaches to mapping parallel programs onto a hypercube / P. Sadayappan, F. Ercal // Proceedings of the 1st International Conference on Supercomputing. – 1988. – P. 475-497.

168. Salcedo-Sanz, S. Hybrid meta-heuristics algorithms for task assignment in heterogeneous computing systems / S. Salcedo-Sanz, Y. Hu, X. Yao // Computers and Operations Research. – 2006. – Vol. 33, № 3. – P. 820-835.

169. Schloegel, K. Graph partitioning for high-performance scientific simulations / K. Schloegel, G. Karypis, V. Kumar // Sourcebook of parallel computing. – San Franciso : Morgan Kaufmann Publish, 2003. – P. 491-541.

170. Shen, C.-C. A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion / C.-C. Shen, W.-H. Tsai // IEEE Transactions on Computers. – 1985. – Vol. 34, № 3. – P. 197-203.

171. Simon, H. D. How good is recursive bisection / H. D. Simon, S.-h. Teng // SIAM Journal on Scientific computing. – 1997. – Vol. 18. – P. 1436-1445.

172. Sloan, J. D. High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI / J. D. Sloan. – O'Reilly, 2004. – 360 p.

173. Srinivasan, S. Safety and Reliability Driven Task Allocation in Distributed Systems / S. Srinivasan, N. K. Jha // IEEE Transactions on Parallel and Distributed Systems. – 1999. – Vol. 10, № 3. – P. 238-251.

174. Subramani, V. Selective Buddy Allocation for Scheduling Parallel Jobs on Clusters / V. Subramani [et al.] // In Proceedings of the International Conference on Cluster Computing. – 2002. – P. 107-116.

175. Sunderam, V. The PVM Concurrent Computing System: Evolution, Experiences, and Trends / V. Sunderam [et al.]. – Parallel Computing. – 1994. – Vol. 20, № 4. – P. 531-547.

176. Tarkov, M. S. Mapping parallel programs onto distributed computer systems with faulty elements / M. S. Tarkov [et al.] // Lecture Notes in Computer Science: Computational Science. – 2001. – P. 148-157.

177. Tarkov, M. S. Mapping parallel programs onto distributed robust computer systems / M. S. Tarkov // Proc. Of 15th IMACS World Congress on Scientific Computation. – 1997. – P. 365-370.

178. Traff, J. L. Implementing the MPI Process Topology Mechanism / J. L. Traff // Proceedings of the ACM/IEEE conference on Supercomputing. – 2002. – P. 1-14.

179. Parhami, B. Introduction to Parallel Processing: Algorithms and Architectures / B. Parhami. – New York : Kluwer Academic Publ., 2002. – 532 p.

180. Prakash, D. Cluster-based multiple task allocation in distributed computing system / D. Prakash [et al.] // Parallel and Distributed Processing Symposium. – 2004. – P. 239.

181. Ucar, B. Task assignment in heterogeneous computing systems / B. Ucar [et al.] // Journal of Parallel and Distributed Computing. – 2006. – Vol. 66, № 1. – P. 32-46.

182. Yau, S. A task allocation algorithm for distributed computing systems / S. Yau, V. R. Satish // Proceedings of Computer Software and Applications Conference. – 1993. – P. 336-342.

183. Yu, H. Topology Mapping for Blue Gene/L Supercomputer / H. Yu, I-H. Chung, J. Moreira // Proceedings of ACM/IEEE Conference Supercomputing. – 2006. – P. 52.

184. Zabrodin, A. V. The massively parallel computer system MBC-100 / A. V. Zabrodin, V. K. Levin, V. V. Korneev // Parallel Computing Technologies, 1995. – P. 341-355.

ПРИЛОЖЕНИЯ

Приложение 1. Сводные таблицы алгоритмов

В приложении приведена сводная таблица алгоритмов и их вычислительные сложности.

Условные обозначения:

N – количество ЭМ в системе;

M – количество ветвей в параллельной программе;

T_{GP} – вычислительная сложность алгоритма разбиения взвешенного графа на k неперсикающихся подмножества;

E – множеств ребер информационного графа программы;

k – количество подмножеств разбиения информационного графа (параметр алгоритма TMMGP);

H – количество подсистемы в пространственно-распределенной ВС;

R – длина последовательности $\{c_k\}$ в алгоритме TMSA;

K – количество допустимых решений перебираемых в окрестости текущего (параметр алгоритма TMSA);

h – высота дерева коммуникаицонной среды в модели пространственно-распределенной ВС;

n – количество ЭМ в макроструктуре ВС, достпуных для реализации ветвей параллельных программ;

E' – множество ребер макроструктуры ВС.

В табл. 1 приведена информация об алгоритмах оптимизации вложения параллельных программ в распределенные ВС.

В табл. 2 перечислены алгоритмы формирования подсистем в распределенных ВС.

Таблица 1

№	Задача	Алгоритм	Трудоемкость алгоритма
1	Вложение параллельных программ в ВС с иерархической организацией коммуникационных сред	TMGP	$O(T_{GP} + M\})$
2		TMMGP	$O(\|E\| \log_2 k)$
3	Вложение параллельных программ в подсистемы ВС с иерархической организацией	TMGT	$O(N^2 + M \cdot \|E\|)$
4	Вложение параллельных программ в пространственно-распределенные ВС	TMSA	$O(\max\{M + H, R \cdot K \cdot M^2\})$
5	Вложение параллельных программ в подсистемы пространственно-распределенных ВС	TMDS	$O(\max\{hN^2, M + \|E\|\})$

Таблица 2

№	Задача	Алгоритм	Трудоемкость алгоритма
1	Формирование подсистем в ВС с иерархической организацией	PAGS	$O(n^2)$
2		PAGCS	$O(n^2 + n \cdot \max\{n, \|E'\|\})$
3	Формирование подсистем в пространственно-распределенных ВС	PAHS	$O(M \cdot H)$
4		PADS	$O(h \cdot N^2)$

Приложение 2. Структурная организация сегментов мультикластерной вычислительной системы

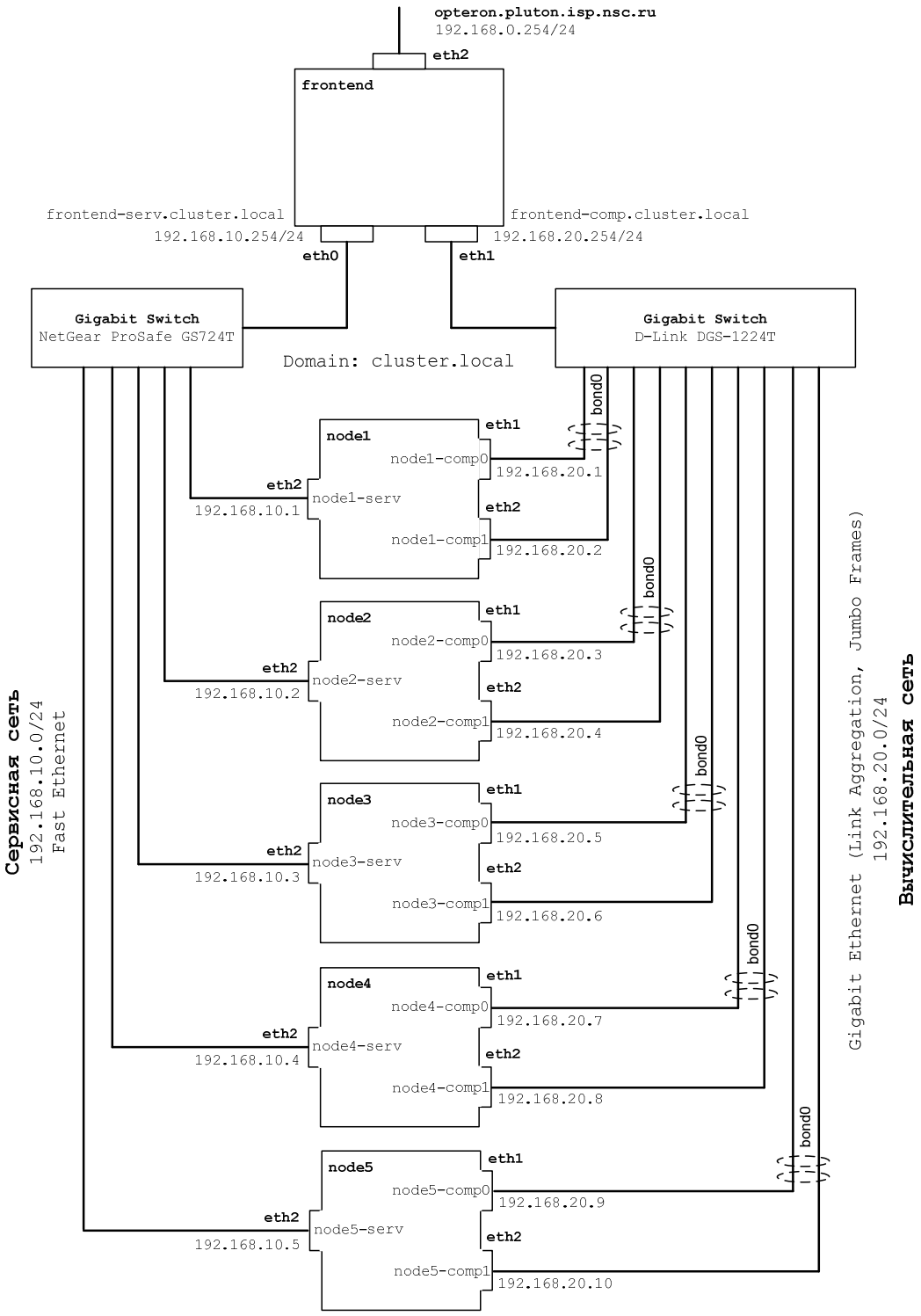


Рис. 1. Структурная организация сегмента G:
5 двухпроцессорных узлов на базе AMD Opteron 248-252

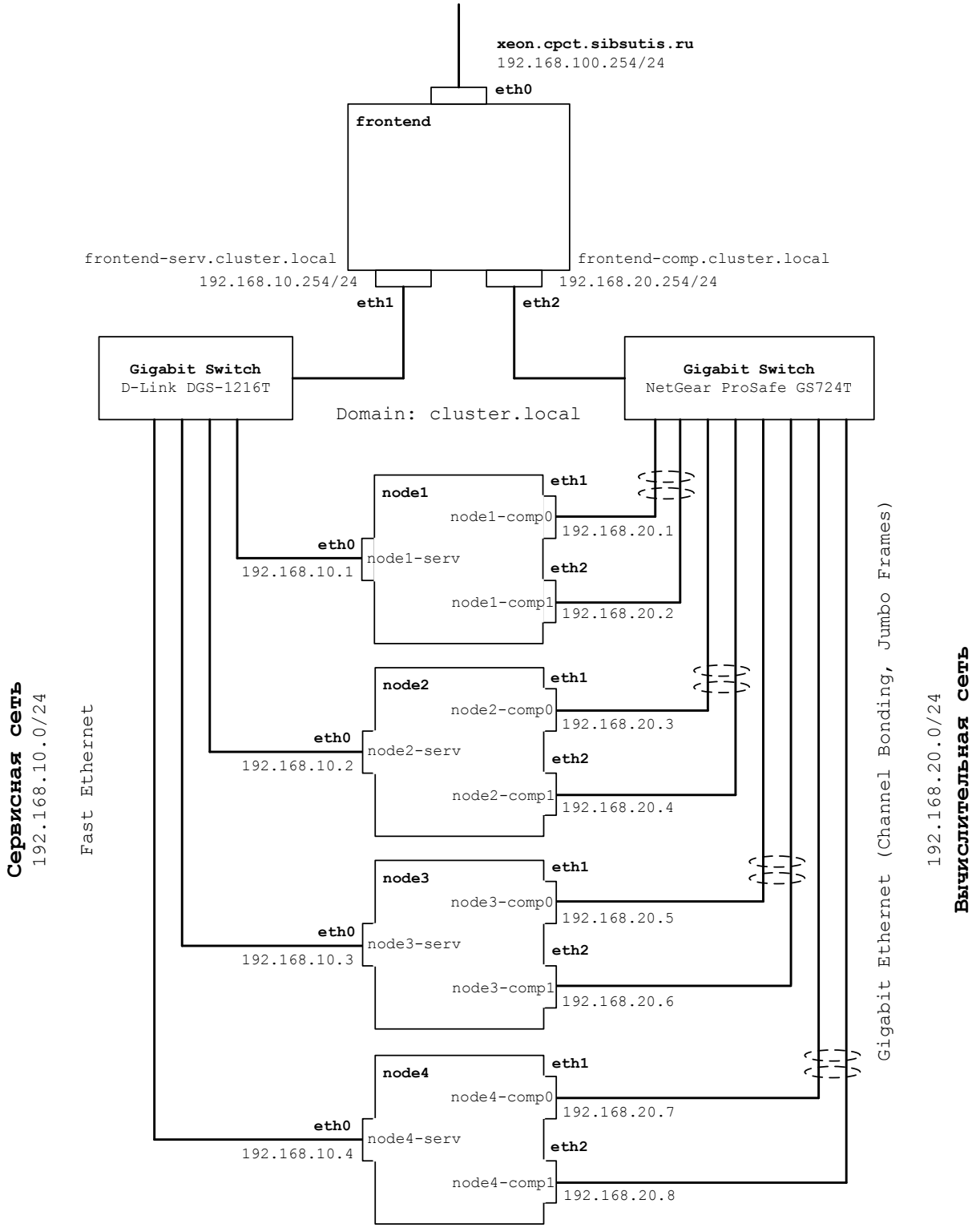


Рис. 2. Структурная организация сегмента E:
4 двухпроцессорных узла на базе Intel Xeon 5150

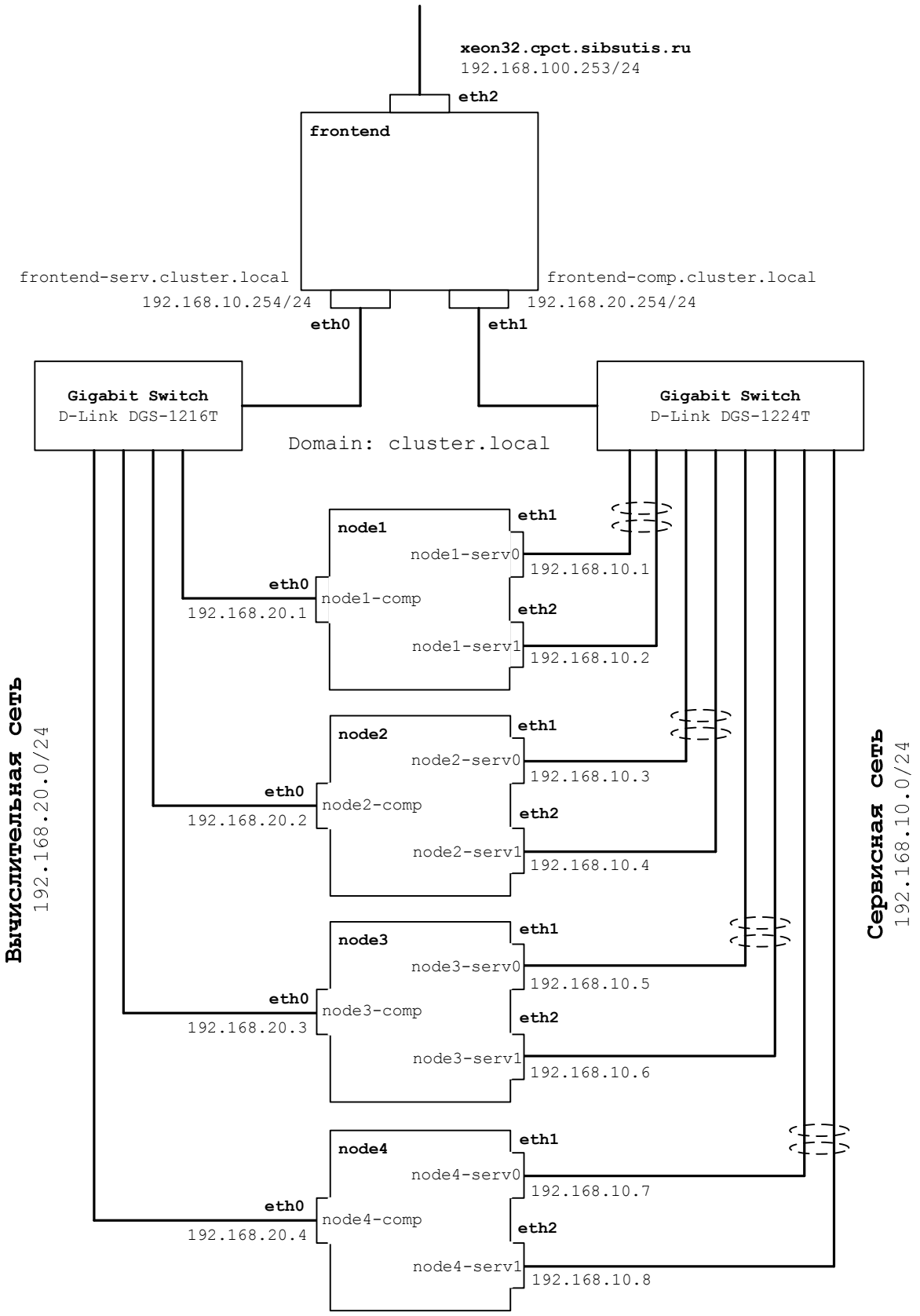


Рис. 3. Структурная организация сегмента F:

4 двухпроцессорных узла на базе Intel Xeon E5345

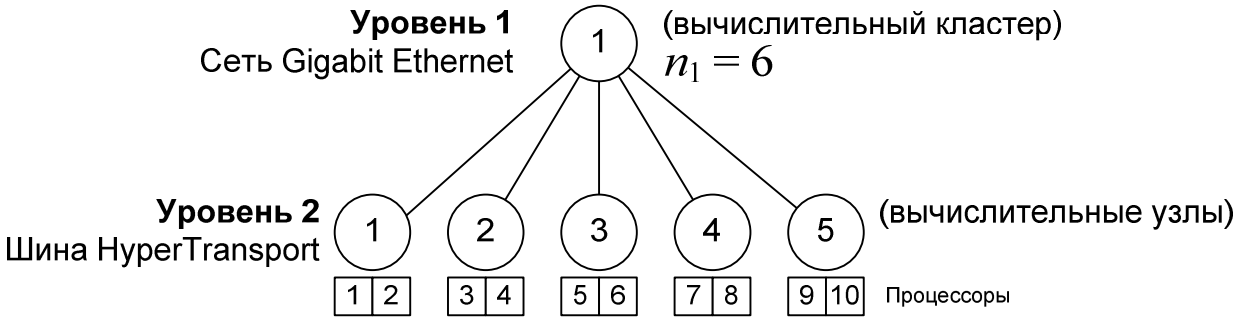


Рис. 4. Иерархическая организация коммуникационной среды сегмента G: $N = 12$; $L = 2$

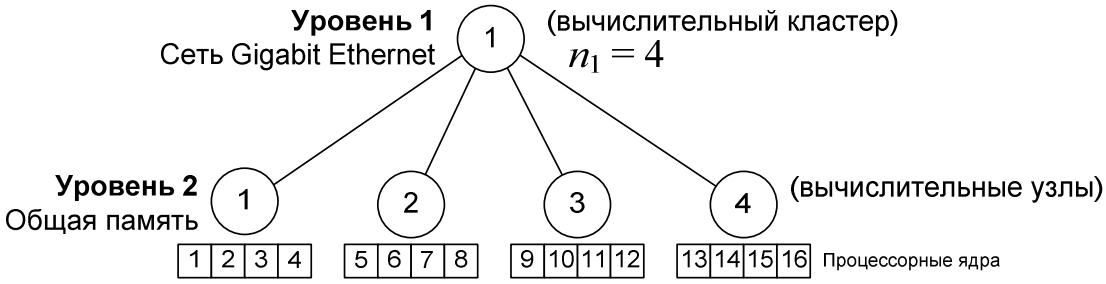


Рис. 5. Иерархическая организация коммуникационной среды сегмента E: $N = 16$; $L = 2$

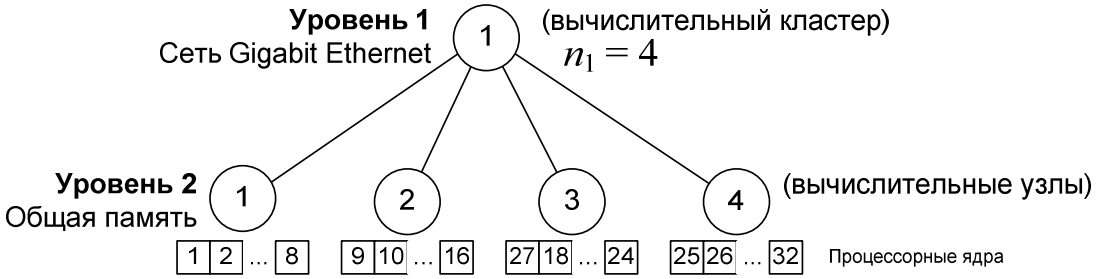


Рис. 6. Иерархическая организация коммуникационной среды сегмента F: $N = 32$; $L = 2$

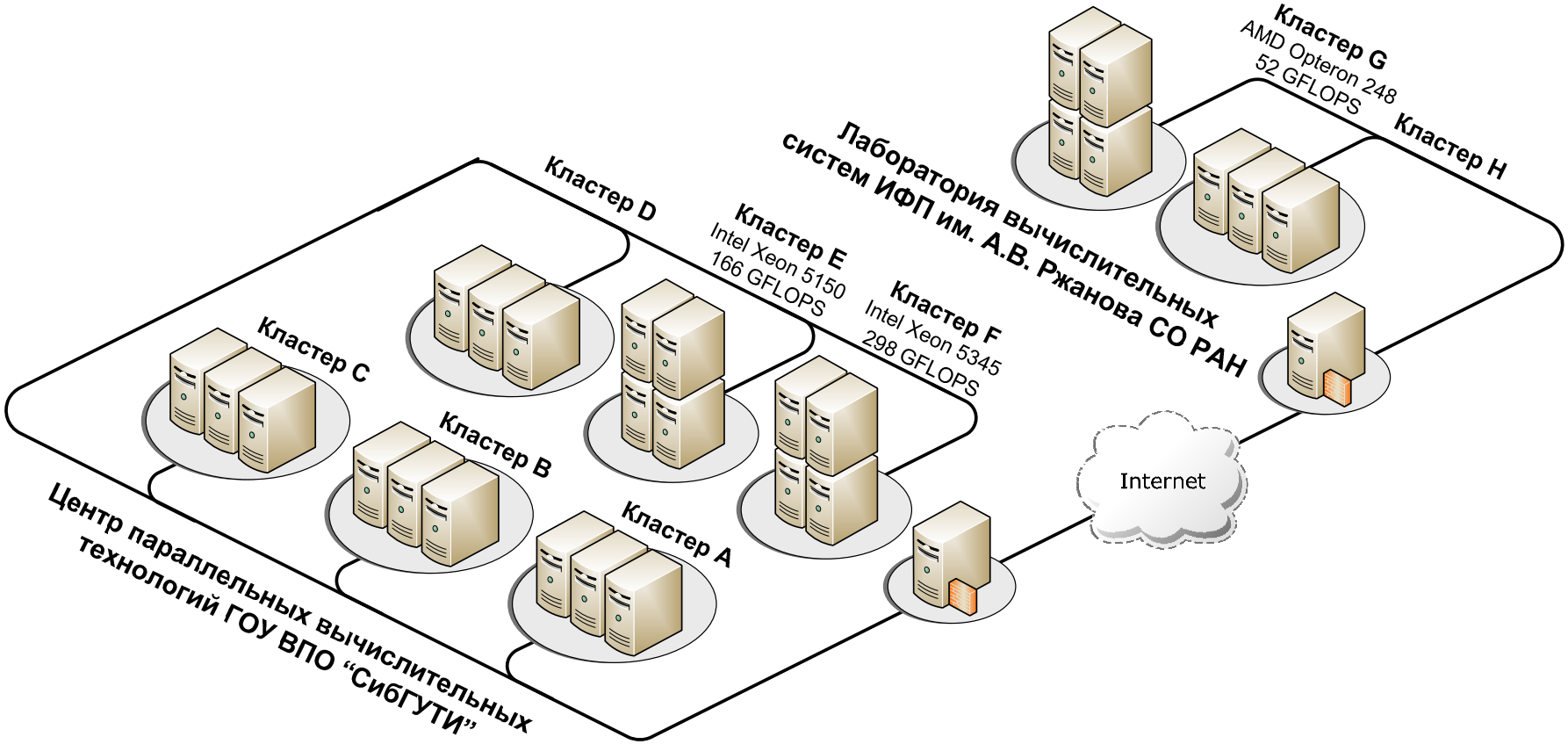


Рис. 7. Конфигурация пространственно-распределенной мультикластерной ВС