Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

УДК

№ госрегистрации 2002 2468

УТВЕРЖДАЮ

Проректор по научной работе

д-р хим. наук

_____С.К. Рахманов

“ 23 ” декабря 2002 г.

ОТЧЕТ

О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

РАЗРАБОТКА МЕТОДОВ АНАЛИЗА И ОБРАБОТКИ ДАННЫХ В СИСТЕМАХ С РАСПРЕДЕЛЕННЫМИ РЕСУРСАМИ

(заключительный)

г.б. НИР №540/18

Зам. декана факультета радиофизики и электроники
канд. физ.-матем. наук, доцент — В. И. Демидчик

Зав. кафедрой
системного анализа,
д-р физ.-матем. наук, профессор — В. В. Апанасович

Куратор проекта, ст. преподаватель
кафедры системного анализа — В. М. Лутковский

Научный руководитель
аспирант 1-го года обучения — П. В. Назаров

Минск 2002

СПИСОК ИСПОЛНИТЕЛЕЙ

Руководитель работы, ответственный исполнитель, аспирант 1-го года обучения	П. В. Назаров (3.1, 3.2, 3.3, 3.4, 3.5, 3.7, 4.1.4, 4.1.5, 4.2, 4.4, заключение)
Исполнитель, студентка магистратуры	А. А. Ковтонюк (1.4)
Исполнитель, студентка магистратуры	Е. В. Лутковская (Введение, 1.1, 1.3, 1.5)
Исполнитель, студент 5-го курса	А. В. Гурбо (2.1, 2.2, 2.3)
Исполнитель, студент 5-го курса	А. Е. Верхотуров (4.3)
Исполнитель, аспирант 1-го года обучения	Р. О. Кожемякин (1.2.1)
Исполнитель, аспирант 1-го года обучения	А. Л. Томилин (1.2.2)
Исполнитель, студентка 4-го курса	Е. В. Макарова (3.6)
Исполнитель, студент 4-го курса	А. М. Поплетеев (4.1.1, 4.1.2, 4.1.3)
Нормоконтролер	Т. Н. Долгая

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ, ЕДИНИЦ И ТЕРМИНОВ

АЛУ – арифметико-логическое устройство;

ГА – генетический алгоритм;

ИМ – имитационное моделирование;

ИНС – искусственная нейронная сеть;

ЛВС – локальная вычислительная сеть;

МКОД (MISD) – множественный поток команд – одиночный поток данных;

МКМД (MIMD) – множественный поток команд – множественный поток данных;

МП – микропроцессор;

ОКОД (SISD) – одиночный поток команд – одиночный поток данных;

ОКМД (SIMD) – одиночный поток команд – множественный поток данных;

ОЗУ – оперативное запоминающее устройство;

ОС – операционная система;

ПК – персональный компьютер;

ЭВМ − электронно-вычислительная машина.

ВВЕДЕНИЕ

Персональные компьютеры (ПК) стали массовым инструментом обработки информации практически во всех отраслях науки и техники. Однако многие задачи, возникающие при проведении научных исследований и разработке новой техники для различных промышленных отраслей (электроники и автомобилестроения, фармакологии и метеорологии, сельского хозяйства и биоинженерии) уже не могут быть решены даже с помощью самых современных персональных компьютеров и рабочих станций. Решение этой проблемы возможно с использованием средствам обработки информации более высокого уровня. К их числу относятся компьютерные системы с распределенными ресурсами.

К проектированию высокопроизводительных средств обработки информации в зависимости от их назначения исторически сложились различные подходы. В настоящее время самостоятельно развиваются суперкомпьютеры и многопроцессорные вычислительные системы с параллельной обработкой информации [1–14], нейрокомпьютеры и распределенные системы управления информационными потоками больших предприятий [15–24].

Суперкомпьютеры класса CRAY XMP или GYBER 205 обладали большой вычислительной мощностью, но они были слишком дорогими. Высокие скорости обработки информации в них достигались благодаря использованию векторных процессоров, конвейерных спецпроцессоров и сжатого времени цикла. Они были построены на основе лучших достижений технологии своего времени, а дальнейшее увеличение быстродействия сдерживалось физическими ограничениями скорости распространения сигналов в элементах ЭВМ. Выход из этого «технологического тупика» был найден при разработке ЭВМ Intel Hypercube, Connection Machine и др. Увеличение быстродействия достигалось в результате использования новых архитектурных решений для «мелкозернистой» обработки данных: одиночный поток команд – множественный поток данных (ОКМД или SIMD) и множественный поток команд – множественный поток данных (МКМД или MIMD) [9].

Нейрокомпьютер с точки зрения традиционной вычислительной техники – это вычислительная система с архитектурой типа ОКМД или МКМД с предельно упрощенными процессорными элементами, максимально развитыми связями и качественно отличающимися принципами программирования, основанными на изменении связей между отдельными процессорными элементами [15].

Основная цель настоящего проекта заключается в исследовании возможности дальнейшего развития систем обработки данных на базе ПК, реализующих принципы распределения вычислительного процесса и данных между ресурсами отдельных компьютеров.

В первом разделе отчета дается анализ принципов построения систем с распределенными информационными и вычислительными ресурсами, а также их основных характеристик.

Во втором разделе рассматриваются характеристики вычислительного кластера и требования стандарта MPI (Message Passing Interface) к специализированному программному обеспечению, с помощью которого можно эффективно воспользоваться возможностью распределенной обработки данных.

В третьем разделе рассмотрены разработанные в рамках проекта алгоритмы и методы обработки и анализа информации в системах с распределенными ресурсами. Все алгоритмы предназначены для решения задачи определения неизвестных параметров (идентификации) сложных стохастических систем.

В четвертом разделе приведены примеры реализации разработанных алгоритмов моделирования и обработки данных в системах с распределенными ресурсами. В этом же разделе дается описание разработанных программных средств параллельного имитационного моделирования PARSIM. Эти средства созданы для организации параллельного имитационного моделирования (ИМ) на компьютерах под операционными системами Windows 9X/NT/XP, соединенных в локальную сеть. Их высокая эффективность достигается в результате параллельной обработки информации.

1. СТРУКТУРА И КЛАССИФИКАЦИЯ СИСТЕМ С РАСПРЕДЕЛЕННЫМИ РЕСУРСАМИ

Супер-ЭВМ это компьютеры, имеющие в настоящее время не только максимальную производительность, но и максимальный объем оперативной и дисковой памяти. Суперкомпьютеры воплощают идею параллельной обработки данных, существующую в двух разновидностях: конвейерности и параллельности [2].

Критерии классификации параллельных ЭВМ могут быть разными: вид соединения процессоров, способ функционирования процессорного поля, область применения. Одна из наиболее известных классификаций параллельных ЭВМ предложена Флинном [13, 15] и отражает форму реализуемого ЭВМ параллелизма. Основными понятиями классификации являются «поток команд» и «поток данных». Под потоком команд упрощенно понимают последовательность команд одной программы. Поток данных – это последовательность данных, обрабатываемых одной программой.

Согласно классификации Флинна имеется четыре больших класса ЭВМ:

1) ОКОД (одиночный поток команд – одиночный поток данных) или SISD (Single Instruction – Single Data). Это последовательные ЭВМ, в которых выполняется единственная программа, т. е. имеется только один счетчик команд и одно арифметико-логическое устройство (АЛУ).

2) ОКМД (одиночный поток команд – множественный поток данных) или SIMD (Single Instruction – Multiple Data). В таких ЭВМ выполняется единственная программа, но каждая ее команда обрабатывает много чисел. Это соответствует векторной форме параллелизма.

3) МКОД (множественный поток команд – одиночный поток данных) или MISD (Multiple Inslruciion Single Data). Подразумевается, что в данном классе несколько команд одновременно работает с одним элементом данных.

4) МКМД (множественный поток команд – множественный поток данных) или MIMD (Multiple Instruction – Multiple Data). В таких ЭВМ одновременно и независимо друг от друга выполняются несколько программных ветвей, обменивающихся данными в определенные промежутки времени. Обычно это многопроцессорные системы.

1.1. Эволюция структуры систем с распределенными ресурсами

1.1.1. Комплексирование однопроцессорных ЭВМ

Идея объединения (комплексирования) ЭВМ с целью увеличения производительности обработки данных возникла достаточно давно. Комплексирование

однопроцессорных ЭВМ серии ЕС ЭВМ осуществлялось по следующим четырем уровням [4]:

1. на уровне процессов для синхронизации и управления;
2. на уровне каналов ввода/вывода через адаптер «канал–канал» (аппаратное устройство, используемое для связи двух каналов);
3. на уровне оперативной памяти (за счет многовходовой памяти);
4. на уровне внешней памяти.

При комплексировании на первых трех уровнях, что характерно для малых и средних машин, создаются многомашинные комплексы. При комплексировании на уровне оперативной памяти, с сохранением возможности комплексирования на других уровнях создавались многопроцессорные системы.

Отметим, что программная организация работы процессоров с общим полем оперативной памяти принципиально сложнее, нежели взаимодействие процессоров на уровне внешнего оборудования, хотя является наиболее гибкой и быстрой.

Обмен информацией между процессорами или между процессором и внешней памятью (управляющей или синхронизирующей информацией) использует средства прямого управления, к которым относятся команды прямой записи и прямого чтения. Связь между процессорами или процессором и внешней памятью осуществляется с помощью указанных команд и механизма внешних прерываний.

При комплексировании на уровне каналов используется адаптер, который имеет два выхода на стандартный интерфейс ввода/вывода и подключается к селекторным (или мультиплексным) каналам двух модулей ЭВМ.

При комплексировании на уровне внешней памяти использовались двух-канальные переключатели, позволяющие подсоединять внешние накопители данных к двум каналам различных ЭВМ, в результате чего образуется общее поле памяти на управляемых ими накопителях. Возможно взаимодействие процессоров по телефонным и телеграфным линиям связи.

В случае прямого управления обмен информацией выполняется по командам прямого управления. По ним из одного процессора передается один байт информации, прерывающий работу другого процессора. Такая система эффективна, если объем и скорость передачи информации небольшие.

При организации многомашинного комплекса, связанного общим полем внеш-ней памяти, передача информации одним процессором и прием ее другим происходят не одновременно. В результате производительность взаимодействующих систем не снижается. Эта связь эффективнее объединения с помощью линий прямого управления.

В случае связи через адаптер «канал–канал» прямая связь между каналами ЭВМ осуществляется через стандартный интерфейс. Скорость обмена информацией определяется пропускной способностью каналов.

Структурная схема адаптера «канал–канал» дана на рис. 1. Каждый канал обслуживается своим блоком управления. Эти блоки связаны как непосредственно при помощи сигнальных линий, так и через общий *однобайтовый* буферный регистр.

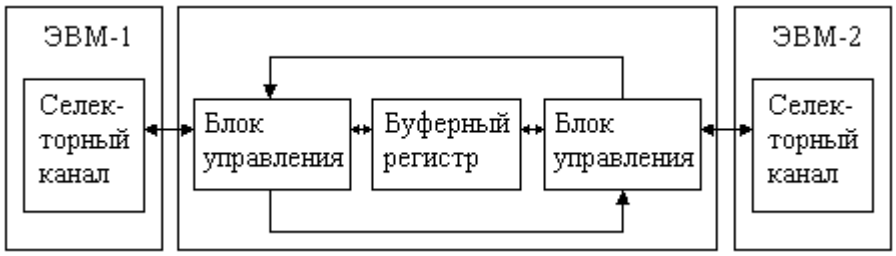


Рис. 1 Структурная схема адаптера «канал–канал»

Режим 1. Оба компьютера принимают информацию с внешних устройств и обрабатывают ее, выдачу информации осуществляет только основная ЭВМ, а вторая резервирует ее. Если первая машина не может выполнять планируемые функции, ее работу выполняет вторая.

Режим 2. Первый компьютер выполняет свои основные функции, а второй выходит на профилактику. Это режим работы без резервирования. В случае надобности (например, если компьютер вышел из строя) вторая машина может быть выведена из профилактики и переведена в рабочий режим.

Режим 3. Оба компьютера работают независимо.

Текущее состояние каждой ЭВМ фиксируется и как байт состояния хранится в блоке состояния многомашинного комплекса. Задание режима работы многомашинного комплекса и перевод в требуемый режим осуществляются оператором с пульта ЭВМ. Изменить байт состояния компьютера в блоке состояния многомашинного комплекса можно и программным путем с помощью команд прямого управления. Блок состояния, в свою очередь, связан с ЭВМ по двум стандартным интерфейсам прямого управления. Адаптеры «канал–канал» предназначены как для передачи массивов данных между каналами ввода–вывода ЭВМ данного многомашинного комплекса, так и для связи с другими системами.

Адаптер передает информацию в автономном режиме со скоростью более медленного канала из двух соединенных. Он играет роль устройства управления для канала: реагирует на запросы канала, принимает и расшифровывает команды каналов,

но использует все полученные сведения не для управления устройствами ввода–вывода, а для обеспечения связи между каналами и синхронизации их работы.

Для создания общего поля оперативной памяти в системе содержались двухходовые устройства оперативной памяти. В двухпроцессорной системе с общим полем оперативной памяти задание режима работы системы, физического распределения ресурсов системы между процессорами и исключения неисправных устройств осуществляется с помощью пульта реконфигурация системы.

Прямо адресуемая память одного из двух процессоров во избежание наложения ее на прямо адресуемую память другого процессора смещена путем префиксации адреса. Основные сведения о комплексировании компьютеров даны в таблице 1.

Таблица 1

Устройства ЭВМ. Уровни комплексиро-вания	Технические и программные средства комплексирования	Реализация комплекси-рования	Результат комплексирования
Процессор	Средства прямого управления (включая шины прямого управления, линии синхронизации и спецкоманды) и механизм внешних прерываний	Прямая связь между двумя процессорами	Обмен управляющей информацией между процессорами и необходимая при этом синхронизация их работы
Оперативная память	Пульт реконфигурации, средства операционной системы и средства прямого управления	Общее поле оперативной памяти и прямая связь между двумя процессорами	Быстрый параллельный доступ процессоров к программной и числовой информации в общем поле оперативной памяти, реализуемый с участием расширенных средств прямого управления
Каналы	Адаптер «канал–канал» и средства операционной системы	Соединение канала одной ЭВМ с каналом другой ЭВМ	Быстрая синхронизированная передача программной и числовой информации из оперативной памяти одной ЭВМ в оперативную память другой ЭВМ
Внешняя память	Двухканальный переключатель (входной коммутатор) устройств управления внешними устройствами и средствами операционной системы	Общее поле внешней памяти	Одновременный и разнесенный во времени доступ процессоров к большим объемам информации в общем поле внешней памяти

Современные микропроцессоры используют тот или иной вид параллельной обработки [12]. В ядре Pentium IV на разных стадиях выполнения может одновременно находиться до 126 микроопераций. Так на Pentium IV вычислительные программы, специально оптимизированные для этого процессора, работают примерно на 50% быстрее не оптимизированных. Самый наглядный пример – кодер DivX 4.12.). Идеи параллелизма появились давно. Изначально они внедрялись в самых передовых, а потому единичных компьютерах своего времени. Затем, после отработки технологии и удешевления производства, они спускались в компьютеры среднего класса, и, наконец, сегодня все это в полном объеме воплощается в рабочих станциях и персональных компьютерах.

Так, разрядно-параллельная память и разрядно-параллельная арифметика впервые появились в компьютерах IBM 701 (1953 год) и IBM 704 (1955), независимые процессоры ввода/вывода – в IBM 709 (1958), опережающий просмотр вперед и расслоение памяти – в IBM Stretch (1961), конвейер команд – в Atlas (1963), независимые функциональные устройства – в CDC 6600 (1964), независимые конвейерные функциональные устройства – в CDC 7600 (1969), матричные процессоры – в ILLIAC IV (1974), векторно-конвейерные процессоры – в Cray1 (1976).

1.1.2. Направления развития вычислительной техники

Развитие высокопроизводительной вычислительной техники в настоящее время идет по четырем основным направлениям [2].

1. Векторно-конвейерные компьютеры. Конвейерные функциональные устройства и набор векторных команд – две главные особенности таких машин. В отличие от традиционного подхода векторные команды оперируют целыми массивами независимых данных, что позволяет эффективно загружать доступные конвейеры, то есть команда вида A=B+C может означать сложение двух массивов, а не двух чисел. Характерный представитель этой группы – семейство векторно-конвейерных компьютеров Cray, куда входят, например, Cray EL, Cray J90 и Cray T90.

2. Параллельные компьютеры с общей памятью. Оперативная память таких компьютеров разделяется нескольким и одинаковыми процессорами, благодаря чему снимаются проблемы предыдущего класса, но добавляются новые: число процессоров, имеющих доступ к общей памяти, по чисто техническим причинам нельзя сделать большим. В эту группу входят многие современные многопроцессорные SMP-компьютеры или, например, отдельные узлы компьютеров HP Exemplar и Sun StarFire.

3. Массивно-параллельные компьютеры с распределенной памятью. Идея построения компьютеров этого класса тривиальна: возьмем серийные микропроцессоры, снабдим каждый своей локальной памятью, соединим посредством некоторой коммуникационной среды – вот и все. Такая архитектура имеет много преимуществ: если нужна высокая производительность – можно добавить еще процессоров; если ограничены финансы или заранее известна потребная вычислительная мощность – легко подобрать оптимальную конфигурацию и т. п. Ее недостаток заключается в том, что межпроцессорное взаимодействие в компьютерах этого класса идет намного медленнее локальной обработки данных самими процессорами. Именно поэтому создать эффективную программу для таких компьютеров очень сложно, а для некоторых алгоритмов иногда просто невозможно. К этому классу можно отнести компьютеры Intel Paragon, IBM SP1, Parsytec, в какой-то степени IBM SP2 и Cray T3D/T3E (причем острота вышеупомянутого недостатка в них значительно ослаблена), а также компьютерные сети, которые все чаще рассматривают как дешевую альтернативу крайне дорогим суперкомпьютерам.

4. *Кластерная архитектура*. Это направление включает в себя комбинацию трех предыдущих. Из нескольких процессоров (традиционных или векторно-конвейерных) и общей для них памяти формируют вычислительный узел. Если полученной вычислительной мощности недостаточно – объединяют несколько узлов высокоскоростными каналами. По такому принципу построены Cray SV1, HP Exemplar, Sun StarFire, NEC SX-5, последние модели IBM SP2 и др.

Поскольку кластерное решение позволяет достичь наилучшего соотношения цены и производительности, именно оно является в настоящее время наиболее перспективным для конструирования компьютеров с рекордными показателями производительности. Не зря самый мощный российский суперкомпьютер, установленный в Межведомственном суперкомпьютерном центре (768 процессоров Alpha 21264 в узлах, связанных между собой коммуникационной сетью Myrinet), создан по кластерной технологии, а в последнюю, 18-ю, редакцию списка пятисот самых мощных компьютеров мира вошли уже 43 кластерные системы.

1.2. Системы с распределенными информационными ресурсами

Широкое распространение локальных и глобальных компьютерных сетей привело к необходимости распределения информационных ресурсов между различными частями информационных систем. Примерами служат системы обработки данных, применяемые в страховом и банковском деле.

В настоящее время в индустрии технологий обработки данных одним из наиболее востребованных направлений развития являются распределенные информационные системы [16]. Это связано, в первую очередь, со значительным увеличением объемов информации, распространением и расширением глобальной сети Интернет, и, как следствие, с все большим вовлечением различных организаций, корпораций и пользователей информационных ресурсов в процесс обмена, обработки и передачи информации. Совершенствование каналов связи и увеличение их пропускной способности, создание принципиально новых устройств, позволяющих осуществлять доступ к удаленной информации, количественное и качественное повышение свойств аппаратных средств дали новый толчок развитию распределенных архитектурных решений при проектировании различного рода информационных систем.

К программным комплексам нового поколения выдвигаются принципиально новые требования, отличные от требований, предъявляемых к классическим системам [17-19]. В первую очередь эти требования касаются возможности обработки информации, поступающей не только от участников процесса функционирования системы, расположенных локально, например, в той же подсети предприятия, которое данная система обслуживает, но и от гораздо более удаленных клиентов, зачастую находящихся в значительном удалении от ядра системы. Единственным средством взаимодействия в данной ситуации является сеть Интернет. При построении систем, функционирующих в рамках данной среды, необходимо учитывать совершенно новые ограничения и условия, возникновение которых трудно было предположить ранее [16,21]. Эти требования касаются увеличения загрузки каналов связи при обработке данных, необходимости обработки разнородной информации и различных форматов, повышенных требований к безопасности систем, поддержки различных типов клиентов.

Известно большое количество различных решений, пригодных для построения распределенных систем управления и обработки данных [20,22,23]. Такие решения существуют в виде готовых платформ конкретных производителей, в виде спецификаций, выдвигаемых к производителям и обеспечивающих возможность функционирования целевой системы на решении любого из производителей, а также в виде моделей, позволяющих строить комплексные системы с помощью традиционных программных средств. Все решения такого рода различны по многим критериям. Сюда можно отнести и вопросы производительности, стоимости, расширяемости, скорости разработки и многие другие. Решение о том, какие из критериев являются наиболее

критичными и значимыми, осуществляется в каждом конкретном случае отдельно. Соответственно и выбор решения является индивидуальным в каждом из случаев.

1.2.1. Хранение информации

В любой информационной системе одной из основных задач является задача хранения и доступа к данным, подлежащим обработке [19,24]. Данные, по сути, являются основным и ключевым звеном системы рассматриваемого типа. Необходимо обеспечить не только надежное и защищенное хранение информации, но и предоставить возможность быстрого и надежного доступа к данным с целью их модификации или чтения.

В рамках реализации задачи хранения информации важно учесть ряд факторов, влияющих на функционирование системы, происхождение которых связано с иными, не связанными с хранением данных, задачами.

В первую очередь, исходя из специфики задачи, нужно учитывать, что объемы данных, с которыми ведется работа, могут быть очень большими, а именно для некоторых ключевых таблиц количество записей может достигать при промышленном использовании системы нескольких сотен тысяч записей. Такие объемы предполагают использование мощных серверов баз данных, удовлетворяющих всем необходимым в таких случаях требованиям: разделение ресурсов, устойчивость, надежность хранения, кэширование (хранение наиболее часто используемых данных в специальных блоках оперативной памяти), быстрота доступа. Кроме того, в данном случае может возникнуть необходимость разнесения хранимой информации по нескольким источникам. Такая необходимость возникает, во-первых, когда необходимо снизить нагрузку на сервер базы данных и перенести часть данных на другой сервер (или несколько серверов), а во-вторых, когда некоторые компоненты системы могут обращаться к различным, в общем случае, разнородным источникам данных. В последнем случае источниками информации могут выступать как сервера баз данных, так и другие приложения, предполагающие собственный метод работы с данными (так называемые «унаследованные» системы) [17,19,24]. Кроме того, необходимо обеспечить целостность данных при работе с несколькими источниками данных, избежать конфликтов чтения/записи при одновременном доступе к разнородным источникам данных.

Во-вторых, с учетом того, что система работает с разнородными в общем случае источниками данных, функционирование приложения должно быть прозрачным и не должно зависеть от структуры источников информации, их типа, формата и способа

доступа к данным, то есть необходимо обеспечить переносимость уровня хранения информации. Это означает, что приложение должно быть спроектировано таким образом, чтобы избежать какой либо жесткой привязки к способу хранения информации, серверу баз данных, формату хранения и т. д. и может быть легко перенесено на иной (иные) сервер баз данных, унаследованное приложение или некий иной стандартный метод хранения данных. Архитектура и модель приложения при такой замены остается неизменным, а изменяется только описание доступа к источникам хранимой информации. О способах решения такой проблемы речь пойдет ниже. Особенно актуальной необходимость обеспечения этого требования становится в свете следующего требования: система должна уметь работать в различных режимах, а именно режим сервера (так называемый «Online» режим) когда идет работа одновременно большого числа пользователей, осуществляющих доступ удаленно, и нагрузка на сервер базы данных максимальна; корпоративный режим («Intranet» режим), когда осуществляется доступ одновременно небольшого круга людей, возможно, сотрудников одной компании, работающих в рамках корпоративной сети и нагрузка на сервер значительно ниже; локальный режим («Offline» режим), когда все приложение установлено на отдельном персональном компьютере, возможно мобильном, и отдельный пользователь индивидуально ведет работу с небольшим объемом данных, а затем переносит информацию со своей машины на основной сервер. Очевидно, что каждый режим предполагает различные требования к проектируемой системе и аппаратному обеспечению, равно как и аппаратное обеспечение и специфика режима накладывает свои ограничения на возможности системы. В первом случае система максимально реализует возложенные на нее задачи, что предполагает максимальное использование ресурсов баз данных и аппаратного обеспечения. При этом необходимо вести работу с наиболее мощным и надежным аппаратным обеспечением и программным обеспечением и, в частности, с программным обеспечением сервера базы данных. При втором режиме нагрузка на ресурсы ниже, что означает отсутствие необходимости разделения информации по нескольким серверам и использование менее мощных серверов. В третьем режиме, напротив, из-за ограниченности ресурсов система должна удовлетворять требованиям компактности, экономичности в использовании ресурсов, а это в свою очередь предполагает использование специального программного обеспечения, в частности легких баз данных. Важно то, что в каждом случае используется один и тот же продукт, а его настройка под конкретный режим работы не должна приводить к модификации самого продукта.

1.2.2. Сетевой доступ

Новая версия системы, получившая название «e-Silva», предполагает не только возможность работать локально на компьютере агента или в локальной сети предприятия-заказчика данной системы, но также и в глобальной среде Интернет [22]. Данный факт означает использование совершенно новых концепций при разработке архитектуры системы. Во-первых, необходимо учитывать, что в данном случае используются совершенно иные в отличие от локальных методов взаимодействия протоколы и средства связи. Для системы это означает более низкую скорость передачи данных, ограничения, связанные с обеспечением надежности, безопасности передачи информации, а также с разнородностью среды, включающую как различие платформ, используемых различными пользователями, так и различие географического положения, предполагающее использование разными пользователями разных языков и региональных настроек, что необходимо учитывать при обработке передаваемых данных.

Кроме того, данное требование включает и еще одно, не менее важное, свойство системы, а именно обеспечение возможности одновременного доступа значительного числа пользователей в систему, причем в зависимости от конкретной реализации как минимальное, так и максимальное число пользователей может варьироваться. Следовательно, необходимо обеспечить такой режим работы, когда любой пользователь максимально быстро получает необходимые ему данные, система ведет себя устойчиво на протяжении длительного срока работы и исключены задержки при обработке данных. Достичь этого можно различными методами, например введением в действие не одного сервера приложений, обеспечивающего обработку запросов пользователей, а нескольких одновременно, однако при этом они должны работать как единое целое, что является более сложной задачей.

1.3. Системы с распределенными вычислениями

Параллельная обработка данных, воплощая идею одновременного выполнения нескольких действий, имеет две разновидности: конвейерность и собственно параллельность. Оба вида параллельной обработки достаточно понятны, поэтому ниже даны лишь небольшие пояснения.

1.3.1. Типы параллельной обработки данных.

Параллельная обработка. Если устройство выполняет одну операцию за единицу времени, то тысячу операций оно выполнит за тысячу единиц. Если имеется пять таких же независимых устройств, способных работать одновременно, то тысячу операций система из пяти устройств может выполнить уже за двести единиц времени. Аналогично, системе из N устройств на ту же работу понадобится $1000/N$ единиц времени.

Конвейерная обработка. Для сложения двух вещественных чисел, представленных в форме с плавающей запятой, требуется множество мелких операций – сравнение порядков, выравнивание порядков, сложение мантисс, нормализация и т. п. Первые процессоры для каждой пары аргументов выполняли операции последовательно, одна за другой, пока не доходили до окончательного результата. Лишь после этого переходили к обработке следующей пары слагаемых.

Идея конвейерной обработки заключается в разбиении процесса выполнения общей операции на отдельные этапы. Их называют микрооперациями или ступенями. Причем каждая микрооперация, выполнив свою работу, передает результат следующей, одновременно принимая новую партию входных данных. Получается очевидный выигрыш в скорости обработки за счет совмещения прежде разнесенных во времени операций. Конвейерную обработку вполне можно заменить параллельной, для чего нужно продублировать основное устройство столько раз, сколько ступеней конвейера предполагается выделить. Однако стоимость и сложность получившейся системы будет несопоставима со стоимостью и сложностью конвейерного варианта. В противном случае производительность вычислений не повысится.

При разработке параллельной программы, необходимо выделить в ней части, которые могут одновременно вычисляться разными процессорами, функциональными устройствами или же разными ступенями конвейера. Возможность разбиения программы на части определяется наличием или отсутствием в ней истинных информационных зависимостей. Две операции программы (в данном случае под операцией можно понимать как отдельный оператор, так и более крупные куски кода) называются информационно зависимыми**,** если результат выполнения одной операции используется в качестве аргумента в другой. Таким образом, чтобы распараллелить программу, нужно найти в ней информационно независимые операции, распределить их между вычислительными устройствами и обеспечить их синхронизацию и коммуникацию.

1.3.2. Закон Амдала

Используя параллельную систему с p вычислительными устройствами, теоретически можно ожидать ускорения выполнения программы в p раз по сравнению с последовательным вариантом. В действительности достичь этого предела не удается.

Предположим, что мы определили структуру информационных зависимостей программы (что в общем случае это очень непростая задача), и доля операций, которые нужно выполнять последовательно, равна f (при этом под долей понимается не статическое число строк кода, а время выполнения последовательной программы). Крайние случаи в значениях f соответствуют полностью параллельным *(f=0)* и полностью последовательным *(f=1)* программам. Тогда для того, чтобы оценить, какое ускорение S может быть получено на компьютере из p процессоров при данном значении f, можно воспользоваться законом Амдала

$$S \le \frac{1}{f + (1 - f)/p} \qquad (1)$$

Например, если 90% программы исполняется параллельно, а 10% по-прежнему последовательно, то ускорения более чем в десять раз получить в принципе невозможно, независимо от качества реализации параллельной части кода и числа процессоров. Отсюда следует вывод о том, что эффективно может быть распараллелена не любая программа, а только та, в которой доля информационно независимых операций достаточно велика. Как показывает практика, большинство вычислительных алгоритмов устроено в этом смысле довольно хорошо.

Однако даже если все процессоры одинаковы, существуют другие проблемы. Процессоры выполнили свою работу, но результатами чаще всего надо обмениваться для продолжения вычислений, а на передачу данных уходит время, и в это время процессоры опять простаивают. Есть и другие факторы, влияющие на эффективность выполнения параллельных программ, причем все они действуют одновременно, а значит, должны в той или иной степени учитываться при распараллеливании.

Таким образом, необходимо тщательно согласовывать структуру программ и алгоритмов с архитектурой параллельных вычислительных систем, чтобы достичь максимальной эффективности обработки данных.

1.3.3. Основные технологии реализации параллельных алгоритмов

Суперкомпьютеры работают быстро потому, что выполняют значительную часть операций параллельно. Кроме того, используется конвейеризация, предварительная выборка информации и иные технические решения. Для разных видов параллелизма (их классификация приведена выше) существуют свои особенности реализации параллельных алгоритмов [11]:

- POSIX (The Portable Operating System Interface) – стандарт интерфейса С-функций, предназначенный для облегчения переноса программных продуктов между различными операционными системами. POSIX – интерфейс для организации так называемых нитей (Pthreads). Поддерживается практически всеми Unix-системами, но не подходит для практического параллельного программирования. Реализуется на слишком низком уровне, изначально не разрабатывался для проведения параллельных вычислений и неудобен для вызова посредством Fortran-программ. Упоминание Fortran здесь не случайно. Этот язык программирования остается основным и самым удобным языком для написания эффективных вычислительных приложений.
- OpenMP – стандарт библиотек для написания приложений, работающих на общем поле памяти (SMP, NUMA). Предусматривает реализацию распараллеливания путем использования POSIX-threads. В отличие от последнего, у OpenMP есть интерфейс как к С-, так и к Fortran-программам. Существуют открытые реализации стандарта, например OdinMP.
- MPI – открытый пакет библиотек и программ, реализующие модель передачи сообщений. Стандарт MPI получил более широкое распространение, и существует масса коммерческих и открытых продуктов на его основе. Из них следует отметить два свободно распространяемых пакета:
- MPICH, разработанный в Argonne National Laboratory и широко известный в России;
- LAM, созданный в Ohio Supercomputer Center и в настоящее время поддерживаемый Laboratory for Scientific Computing of the University of Notre Dame. Пакет MPICH хорошо работает на множестве различных платформ и поддерживает кластеры на базе SMP-узлов. LAM хорошо показывает себя при использовании на гетерогенных UNIX-кластерах.

Самые популярные технологии на сегодня – это OpenMP и MPI, причем уже заметна тенденция к их слиянию [14]. Одна из них ориентирована на компьютеры с общей памятью, другая – с распределенной (сюда которым относятся кластеры).

Подавляющее большинство кластерных систем используют стандарт MPI. Существует также множество прикладных пакетов со встроенной поддержкой параллелизма.

1.4. Примеры применения систем с распределенными ресурсами

Доказательством высокой эффективности систем с распределенными ресурсами служат следующие примеры их реального применения.

1.4.1. Испытания криптографической стойкости ключей защиты информации

Проект RC5 «Bovine» (бык) продолжался около двух лет и завершился осенью 2002 г. Он проводился с целью определения времени поиска 64-битного ключа методом «грубой атаки» на RC5 кодирование. Основой проекта служили координационные RC5 сервера, которые распределяли блоки ключей, требующих проверки, между участниками проекта, у которых запущена специальная программа-клиент. Время выполнения проекта сильно зависело от количества участников, так как дешифровка ведется методом простой проверки всех возможных ключей.

В проекте RC5, который является основной деятельностью организации distributed.net, участвовало более чем 11000 команд со всего мира, включая команду факультета радиофизики и электроники Белорусского государственного университета.

Технология работы участников проекта достаточно проста. Они устанавливают на своем компьютере программу-клиент, выполняющую перебор ключей в фоновом режиме (совершенно не загружая компьютер, так как она работает только в момент отсутствия загрузки процессора). В момент выхода в сеть он обменивается с distributed.net блоками данных (около 10 секунд). Варианты программы имеются практически для любой операционной системы.

Группа участников из России занимала 3-е место по численности среди всех команд участвующих в проекте. Она объединила в себе более 3500 человек из различных уголков нашей планеты. Проект показал, что 64-битный ключ уже не обеспечивает достаточную криптографическую стойкость, и продемонстрировал эффективность решения сложных задач с использованием распределенных компьютерных систем.

В конце 2002 г. начат аналогичный проект с 72-битным ключом, который по предварительным оценкам до полного завершения может потребовать около 30 лет.

1.4.2. Задачи имитационного моделирования

Система параллельного программирования «ПАРСЕК» использовалась для моделирования процесса распыления твердых тел [10]. Большая часть исследований по проблеме распыления посвящена получению данных и результатов протекания процессов для различных условий. Решение таких задач требует больших объемов вычислений. Применение методов Монте-Карло позволяет получить решения для многокомпонентных и многослойных мишеней, в том числе мишеней сложной геометрии, что обеспечивает моделирование современных технологических процессов изделий электронной промышленности.

В работе [10] предложена методика реализации ряда моделей, целью которой является снижение времени моделирования. Рассмотрены возможности ускорения вычислений при использовании аналитических моделей, реализация которых связана с решением жестких систем обыкновенных дифференциальных уравнений, а также машинного моделирования методами Монте-Карло, главное достоинство которого в том, что он позволяет учитывать любой физический процесс непосредственно.

Апробация параллельной программы моделирования ионного распыления в системах с различным количеством микропроцессоров (МП) позволила получить следующие результаты (см. табл. 2).

Таблица 2

Число ионов	Число процессоров	Число МП	Время моделирования, с	Коэффициент повышения быстродействия
100000	1	–	3290	1,00
	2	–	1747	1,88
	1	1	234	14,00
	1	2	130	25,00
	1	4	70	47,00

1.5. Выводы

Результаты данного раздела указывают на целесообразность использования кластерных технологий и нейронных сетей для решения сложных вычислительных задач.

2. ВЫЧИСЛИТЕЛЬНЫЙ КЛАСТЕР

2.1. Основные характеристики кластера

Под термином «кластер» обычно понимается массив из отдельных машин (рабочих станций или персональных компьютеров), соединенных каналом связи, предназначенный для работы параллельных приложений по модели передачи сообщений. Фактически кластер представляет собой дешевую реализацию массивно-параллельного компьютера [11].

Кластеры можно классифицировать как по типам узлов, так и по видам используемых коммуникаций. Под узлом понимается отдельный компьютер, входящий в состав кластера.

Узлы кластера могут быть одно- или двух-, реже – четырехпроцессорными. Кластеры можно разделить на гомогенные (в которых машины одинаковы по мощности и используемой операционной системой) и гетерогенные (в которых машины разные – или по производительности, или по программному обеспечению). В гетерогенных кластерах возникает дополнительная проблема балансировки загрузки узлов.

Для построения вычислительных кластеров используют самое разное сетевое оборудование, начиная с Fast Ethernet и заканчивая специализированными средствами коммуникации. Все они обычно характеризуются двумя параметрами:

1. Пропускная способность. Скорость передачи данных между двумя узлами после того, как связь установлена. Производитель обычно заявляет пиковую пропускную способность, которая в полтора–два раза выше реально наблюдаемой в приложениях.

2. Латентность. Это среднее время между вызовом функции передачи данных и самой передачей. Оно идет на адресацию информации, срабатывание промежуточных сетевых устройств (коммутаторов) и прочие накладные расходы.

Фактически два этих параметра не только характеризуют кластер, но и ограничивают класс задач, которые могут эффективно решаться на нем. Так, если задача требует частой передачи данных, кластер, использующий сетевое оборудование с большой латентностью, будет большую часть времени тратить даже не на передачу данных между процессами, а на установление связи, узлы же будут простаивать (недогрузка узлов кластера), и мы не получим значительного увеличения производительности.

2.2. Стандарт MPI

Стандарт MPI (Message Passing Interface – взаимодействие через передачу сообщений) является наиболее распространенным при написании вычислительных и иных параллельных приложений для массивно-параллельных суперкомпьютеров и кластеров. Кроме упоминавшихся свободных пакетов MPICH и LAM существуют и другие реализации. Коммерческие чаще всего делаются для Windows NT, а реализации от производителей (Hewlett-Packard, Sun, Cray, Silicon Graphics,) ориентированы на приложения для своих MPI-суперкомпьютеров. Все эти пакеты содержат одинаковый базовый набор MPI-функций, поэтому приложение, отлаженное с использованием одного из них, должно компилироваться с применением других. При этом нужно учитывать, что идеальной совместимости не бывает. Поэтому вполне возможно использование некоторых нестандартных функций – расширений стандарта, снижающих переносимость. Кроме того, отлаженное для одной платформы приложение может быть неэффективным на другой.

К достоинствам пакета MPICH можно отнести почти абсолютную переносимость. В руководстве по его установке приведены десятки различных программно-аппаратных платформ, на которых может быть установлен этот пакет. Кроме того, MPICH распространяется в исходных кодах и собирается на месте, с учетом особенностей конкретной платформы и среды коммуникации. После установки пакет работает со своими библиотеками и встроенными компиляторами C/C++ и Fortran 77/90.

Независимо от реализации стандарт MPI предлагает следующую модель программирования: параллельное приложение состоит из нескольких одновременно выполняемых процессов, которые обмениваются между собой данными с помощью сообщений. Механизм сообщений реализуется посредством функций MPI, скрыт от пользователя и совершенно не зависит от физической привязки процессов, которые могут выполняться на процессорах разных узлов, на разных процессорах одного узла и даже на одном и том же процессоре, благодаря чему параллельные программы можно отлаживать на обычном ПК. Таким образом, появляется возможность создавать хорошо масштабируемые приложения, которые с разной эффективностью выполняются на многих системах.

2.3. Проблемы разработки эффективных параллельных приложений

Для создания эффективного параллельного приложения необходимо выполнить следующие условия.

• Алгоритм должен быть разбит на относительно независимые, требующие примерно одинакового времени выполнения блоки – они будут реализованы как отдельные процессы.

• Параллельно работающие части алгоритма должны составлять большую по времени выполнения часть программы. При уменьшении параллельной части приложения резко снижается его эффективность.

• Количество и объем передаваемых сообщений должны быть минимизированы; кроме того, допустимая частота сообщений может ограничиваться используемыми коммуникациями.

Этим условиям идеально соответствует алгоритм суммирования какого-либо ряда. В нем практически полностью отсутствует последовательная часть приложения, то есть параллелизм приближается к 100%. Число сообщений равно удвоенному числу процессов: вначале каждому процессу рассылается информация о длине суммируемого им участка, и в конце от каждого процесса передается результат, а полученные значения складываются.

В состав MPI входят: библиотека программирования (заголовочные и библиотечные файлы для языков C/C++ и Fortran) и загрузчик приложений.

Кроме того, может присутствовать справочная система (manual pages for Unix), командные файлы для облегчения компиляции/компоновки программ. В стандарте отсутствует все лишнее, например, нет средств автоматического переноса и построения копий исполняемого файла в сети. Это нужно, если MPI-приложение предстоит выполнять сетью машин – но это можно выполнить и утилитами системы Unix. В стандарте нет никаких средств автоматической декомпозиции, нет отладчика (правда, есть функции хронометража и предусмотрена возможность профилирования). То есть это система межпроцессовой связи в чистом виде, и не более того.

Дополнительно включаются: профилирующий вариант библиотеки (используется на стадии тестирования параллельного приложения для определения оптимальности распараллеливания); загрузчик с графическим и сетевым интерфейсом для X-Windows.

Для MPI принято писать программу, содержащую код всех ветвей сразу. MPI-загрузчиком запускается указываемое количество экземпляров программы. Каждый экземпляр определяет свой порядковый номер в запущенном коллективе, и в зависимости от этого номера и размера коллектива выполняет ту или иную ветку алгоритма. Такая модель параллелизма называется *Single program/Multiple data* (SPMD), и является частным случаем модели MIMD. Каждая ветвь имеет пространство

данных, полностью изолированное от других ветвей. Обмениваются данными ветви только в виде сообщений MPI.

Все ветви запускаются загрузчиком одновременно как процессы Unix. Количество ветвей фиксировано – в ходе работы порождение новых ветвей невозможно. Если MPI-приложение запускается в сети, запускаемый файл приложения должен быть построен на каждой машине.

Хотя с теоретической точки зрения ветвям для организации обмена данными достаточно всего двух операций (прием и передача), на практике все обстоит гораздо сложнее. Для управления коммуникациями «точка-точка» (т.е. такими, в которых ровно один передающий процесс и ровно один принимающий) имеется около 40 функций. Пользуясь ими, программист имеет возможность выбрать способ зацепления процессов. В случае неодновременного вызова двумя процессами парных функций приема и передачи могут быть произведены:

- автоматический выбор одного из трех нижеприведенных вариантов;
- буферизация на передающей стороне – функция передачи заводит временный буфер, копирует в него сообщение и возвращает управление вызвавшему процессу. Содержимое буфера будет передано в фоновом режиме;
- ожидание на приемной стороне, завершение с кодом ошибки на передающей стороне;
- ожидание на передающей стороне, завершение с кодом ошибки на приемной стороне.
- способ взаимодействия коммуникационного модуля MPI с вызывающим процессом.

Две простейшие (но и самые медленные) функции – MPI_Recv и MPI_Send – выполняют блокирующую приемопередачу с автоматическим выбором зацепления (кстати сказать, все функции приема совместимы со всеми функциями передачи).

В MPI хорошо продумано объединение ветвей в коллективы. В сущности, такое деление служит той же цели, что и введение идентификаторов для сообщений: помогает надежнее отличать сообщения друг от друга. В большинстве функций MPI имеется параметр типа «коммуникатор», который можно рассматривать как дескриптор (номер) коллектива. Он ограничивает область действия данной функции соответствующим коллективом. Коммуникатор коллектива, который включает в себя все ветви приложения, создается автоматически при старте и называется MPI_COMM_WORLD.

Идентификаторы программист назначает сообщениям вручную, и существует вероятность, что вследствие его ошибки два разных сообщения получат одинаковые идентификаторы. Коллективы создаются функциями самого MPI, так, чтобы гарантированно избежать случайных совпадений.

В качестве идентификатора ожидаемого сообщения функции приема может быть передан так называемый «джокер» – принять первое пришедшее сообщение независимо от его идентификатора и/или отправителя. Такой вызов может по ошибке перехватывать и те сообщения, которые должны быть приняты и обработаны в другом месте ветви. Для коммуникаторов «джокера» не существует, поэтому работа разных функций через разные коммуникаторы гарантированно предохраняет их от утечки информации: коммуникаторы являются не сообщающимися сосудами.

Коммуникаторы, помимо собственного номера и состава входящих в них ветвей, хранят и другие данные. Например, кроме обязательной линейной нумерации, ветвям коллектива может быть дополнительно назначена нумерация картезианская или в виде произвольного графа – это может оказаться удобным для решения некоторых классов задач. Коллективу может быть назначена функция-обработчик ошибок взамен назначаемой по умолчанию. Пользуясь механизмом «атрибутов», для коллектива пользователь может завести набор совместно используемых его ветвями данных.

Все функции приемопередачи в MPI оперируют не количеством передаваемых байт, а количеством ячеек, тип которых задается параметром функции, следующим за количеством: MPI_INTEGER, MPI_REAL и т.д. Это переменные типа MPI_Datatype (тип «описатель типов», каждая его переменная описывает для MPI один тип). Они имеются для каждого базового типа, имеющегося в используемом языке программирования.

Однако, пользуясь базовыми описателями, можно передавать либо массивы, либо одиночные ячейки (как частный случай массива). А как передавать данные агрегатных типов, например, структуры? В MPI имеется механизм конструирования пользовательских описателей на базе уже имеющихся (как пользовательских, так и встроенных). Кроме того, в MPI имеется механизм конструирования новых типов даже более универсальный, чем имеющийся в языке программирования.

В спецификации MPI приведен пример создания пользовательского описателя типа, передача матрицы с использованием которого приводит к ее транспонированию.

Выигрыш от использования механизма конструирования типов очевиден: лучше один раз вызвать функцию приема и передачи со сложным шаблоном, чем двадцать раз с простыми шаблонами.

Сложность (многочисленность функций и обилие аргументов у большинства из них) является ценой за компромисс между эффективностью и универсальностью. С одной стороны должны существовать способы получить *почти* столь же высокую скорость при обмене данными между ветвями, как и при традиционном программировании через разделяемую память и семафоры. С другой стороны, все функции должны работать на любой платформе.

Таким образом, программист заинтересован в инструментах, которые бы облегчали проведение декомпозиции и запись ее в терминах MPI.

В данном случае это средства, генерирующие на базе тех или иных входных данных текст программы на стандартном C или Fortran. При этом программа должна обладать явным параллелизмом, выраженным в терминах MPI, содержать вызовы MPI-процедур, наиболее эффективные в окружающем контексте. Такие средства, в частности, делают написание программы не только легче, но и надежнее, так как:

- ошибки, которые MPI в принципе, не может обнаружить в момент выполнения, генератор имеет возможность обнаруживать в момент построения программы
- некоторые ошибки исключаются вообще, например, одинаковый числовой идентификатор для *разных* сообщений, или неверный (больше общего количества ветвей в коллективе) номер ветви в аргументах функции приемопередачи – в отличие от человека, программа-генератор, если только она сама написана правильно, не в состоянии описаться.

Назовем некоторые перспективные типы такого инструментария, который лишал бы программиста необходимости вообще помнить о присутствии MPI.

Средства автоматической декомпозиции. Идеалом является такое оптимизирующее средство, которое на входе получает исходный текст некоего последовательного алгоритма, написанный на обычном языке программирования, и выдает на выходе исходный текст этого же алгоритма на этом же языке, но уже в распараллеленном на ветви виде, с вызовами MPI. Что ж, такие средства созданы (например, в состав полнофункционального пакета Forge входит, наряду с прочим, и такой препроцессор), но до сих пор, насколько мне известно, никто не торопится раздавать их бесплатно. Кроме того, вызывает сомнение их эффективность.

Языки программирования. Это наиболее популярные на сегодняшний день средства полуавтоматической декомпозиции. В синтаксис универсального языка программирования (C или Fortran) вводятся дополнения для записи параллельных конструкций кода и данных. Препроцессор переводит текст в текст на стандартном

языке с вызовами MPI. Примеры таких систем: mpC (massively parallel C) и HPF (High Performance Fortran).

Общим недостатком инструментов, производящих преобразование «текст в текст», является то, что синтаксическому разбору подвергаются оба текста: и исходный (его обрабатывает распараллеливающий препроцессор), и генерируемый (его обрабатывает компилятор). Это уменьшает скорость построения программы, и, кроме того, необходимость делать синтаксический разбор усложняет написание препроцессора. Поэтому те фирмы-производители, которые поставляют свои ЭВМ вместе с транслятором языка FORTRAN, встраивают HPF прямо в компилятор машинно-зависимого кода. Для расширений языка Си аналогичное решение может быть найдено в использовании GNU C.

Оптимизированные библиотеки для стандартных языков. В этом случае оптимизация вообще может быть скрыта от проблемного программиста. Чем больший объем работы внутри программы отводится подпрограммам такой библиотеки, тем большим будет итоговый выигрыш в скорости ее (программы) работы. Собственно же программа пишется на обычном языке программирования безо всяких упоминаний об MPI, и строится стандартным компилятором. От программиста потребуется лишь указать для компоновки имя библиотечного файла MPI, и запускать полученный в итоге исполняемый не непосредственно, а через MPI-загрузчик. Популярные библиотеки обработки матриц, такие как Linpack, Lapack и ScaLapack, уже переписаны под MPI.

2.3. Выводы

В разделе рассмотрены возможности MPI при разработке программных средств для распределенных систем.

3. ИДЕНТИФИКАЦИЯ ПАРАМЕТРОВ СЛОЖНЫХ СИСТЕМ С ИСПОЛЬЗОВАНИЕМ РАСПРЕДЕЛЁННЫХ ВЫЧИСЛЕНИЙ

3.1 Введение

В данном разделе рассмотрены разработанные в рамках проекта алгоритмы и методы обработки и анализа информации в системах с распределенными ресурсами. Все алгоритмы предназначены для решения задачи определения неизвестных параметров (идентификации) сложных стохастических систем.

3.2 Постановка задачи

Задача определения неизвестных параметров различных экспериментальных систем является одной из основополагающих во многих областях науки. Важнейший этап при этом – адекватный анализ экспериментальных данных, носящих, как правило, статистический характер. Однако из-за сложности проведения анализа всех доступных данных, на практике приходится прибегать к упрощенной трактовке эмпирической информации и фокусировке на отдельных интегральных значениях [25]. Особенно остро данная проблема проявляется при изучении биомолекулярных образований, обладающих высокой степенью сложности, стохастической природой, а также зависимостью от различных физических и химических факторов среды. Одним из наиболее перспективных методов интерпретации экспериментальных данных является метод имитационного или статистического моделирования [26]. Для построения имитационной модели сложной системы достаточно обладать информацией об элементарных процессах, происходящих в ней, и иметь представление о структуре системы, тогда как стандартное математическое моделирование предполагает наличие полного аналитического описания поведения системы и знание законов распределения всех стохастических параметров, используемых при моделировании, что редко осуществимо на практике.

В то же время, определение параметров стохастических систем с помощью имитационного моделирования (ИМ) затруднено, поскольку сопряжено со значительными вычислительными затратами. Как правило, идентификация системы осуществляется стандартными методами многопараметрической оптимизации [27], в которых имитационная модель выступает в качестве стохастической функции, аппроксимирующей результаты эксперимента. Очевидно, что в этом случае число пусков моделирования совпадает или даже превосходит число вычислений функции невязок. Во многих случаях высокие временные затраты не позволяют применять такой

подход для идентификации систем. Поэтому, актуальной является разработка методов и алгоритмов, позволяющих снизить вычислительные и временные затраты при определении параметров систем с помощью статистического моделирования.

3.3 Подходы к применению распределённых вычислений при идентификации процессов и систем

В этой работе предложено несколько путей снижения временных затрат при идентификации сложных систем, в том числе:

- применение параллельных вычислений при имитационном моделировании;
- построение нейросетевой аппроксимации имитационной модели;
- применение параллельных методов оптимизации.

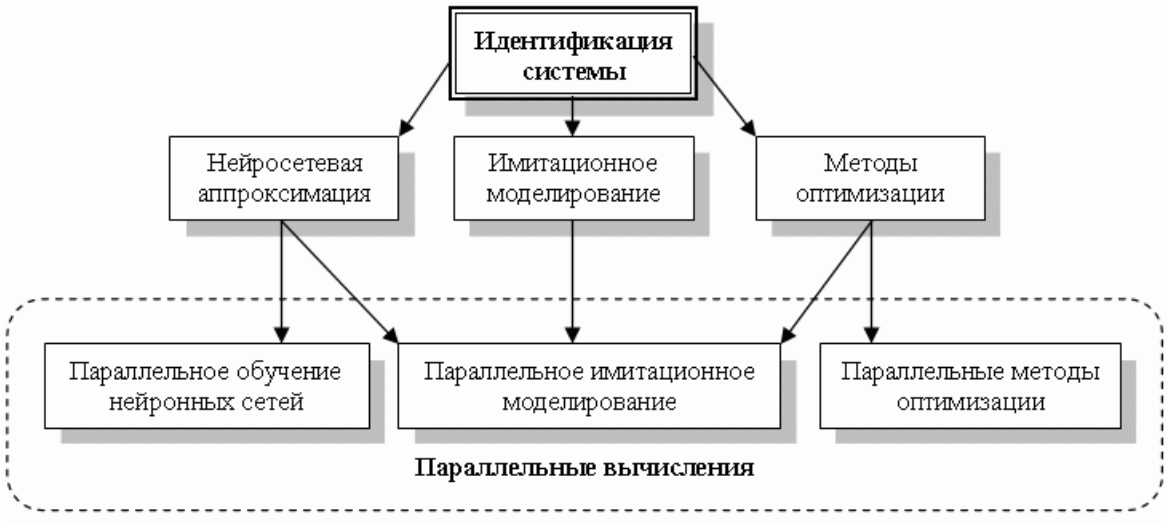


Рис. 2 Параллельные вычисления при идентификации параметров систем

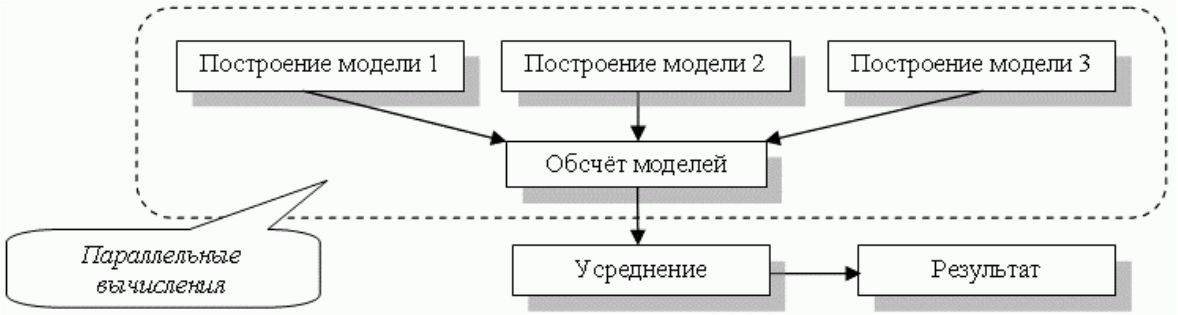
Из схемы, представленной на рис. 2, видно, что наиболее универсальным методом распараллеливания является именно ИМ в распределённых системах. Представленные методы подробнее рассмотрены ниже.

3.4. Распределенное имитационное моделирование

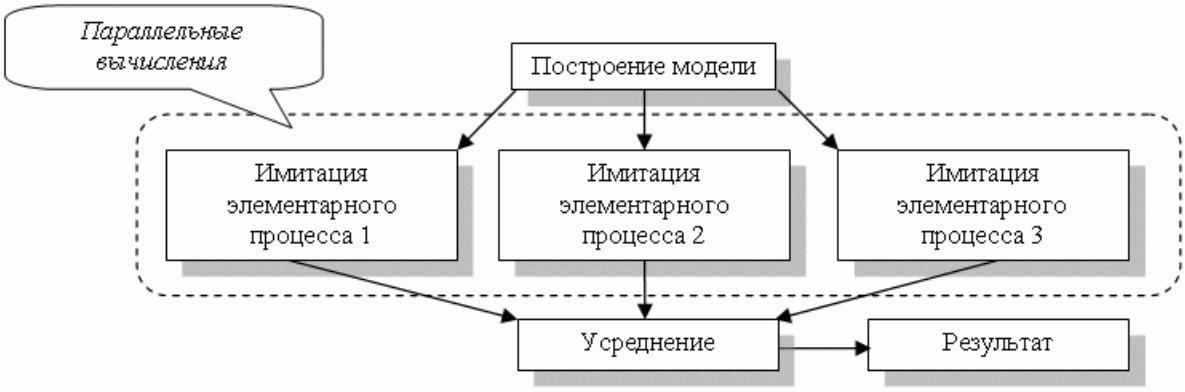
Одним из свойств ИМ является высокий параллелизм. Это позволяет успешно применять параллельные технологии для обсчёта имитационных моделей. Являясь наиболее универсальным подходом, параллельное ИМ, в то же время, предъявляет максимально высокие требования к скорости передачи данных между узлами кластера. Только в случае, когда обсчёт модели занимает значительное время (от нескольких секунд) удается получить выигрыш по скорости.

Рассмотрим следующую задачу реальную задачу. Пусть существует некоторая двумерная молекулярная система (биологическая мембрана), в которой происходят процессы безызлучательного переноса энергии. Построение модели для такой системы происходит в два этапа:

1. Создаётся формализованная пространственная модель системы;
2. Стоится имитационная модель наблюдаемого процесса.



а



б

Рис. 3 Способы распараллеливания ИМ на различных этапах:
а – построение модели; *б* – моделирование процессов методом Монте-Карло

Реальная система может быть стохастичной как на уровне структуры, так и на уровне процессов. Соответственно, как формализованная модель, так и модель процессов может содержать элемент случайности. Для того чтобы уменьшить неопределённость моделирования необходимо либо производить усреднения результатов, либо увеличивать размеры системы. Последнее условие ведет к резкому увеличению вычислительных затрат и приемлемо только до определённого уровня. Для дальнейшего уточнения результатов моделирования следует применять усреднение по набору пространственных моделей (смотри рис. 3, *а*) или (и) по набору результатов моделирования элементарных процессов.

3.5. Распределенные вычисления и нейросетевая аппроксимация

3.5.1. Теория метода

Рассмотрим следующую задачу. У некоторой экспериментальной системы имеется набор входных параметров *P,* часть из которых известна (P_0), а часть предстоит определить (P_X), и набор выходных значений *F*. В этом случае можно сказать, что рассматриваемая система выполняет некоторое преобразование **Θ**:

$$\Theta(P_0, P_X) = F \tag{2}$$

Пусть для этой системы можно построить адекватную имитационную модель. Тогда эта модель будет выполнять преобразование **Ξ**:

$$\Xi(P_0, P_X) = F* \tag{3}$$

где *F** – смоделированные выходные значения.

Алгоритм определения параметров с использованием ИМ выглядит следующим образом:

1. Экспериментально получают некоторую выборку выходных значений *F* при различных входных параметрах системы.
2. Делаются начальные приближения неизвестных параметров P_X.
3. Запускается алгоритм оптимизации, который, изменяя значения P_X, минимизирует ошибку.

Как уже упоминалось, наиболее серьёзной проблемой в этой схеме являются временные затраты на выполнение ИМ. Для того чтобы существенно ускорить этот процесс, авторами данной работы предложено заменить имитационную модель искусственной нейронной сетью.

Известно, что непрерывные функции могут быть с любой заданной точностью аппроксимированы линейной комбинацией и суперпозицией сигмоидальных функций [28]. На практике это может быть реализовано с использованием многослойной искусственной нейронной сети (ИНС) [29]. В нашем случае это означает, что операция ИМ **Ξ** может быть аппроксимирована нейросетевым преобразованием

$$\Psi(P_0, P_X) = F* \tag{4}$$

Вычислительные затраты при этом будут значительно ниже, чем при вычислении результатов ИМ.

Предлагаемый подход представлен на рис. 4.

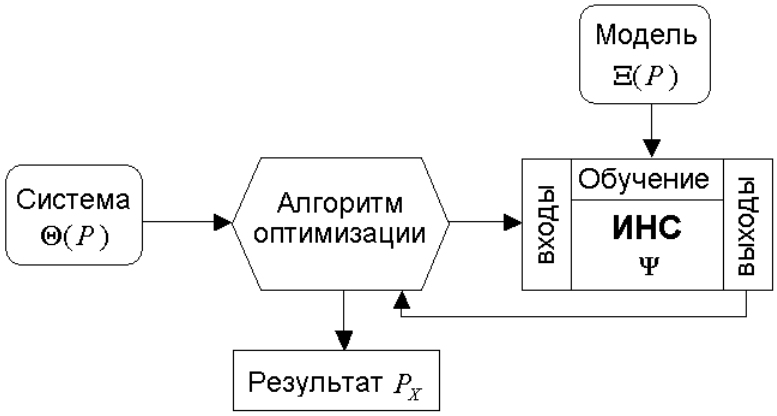


Рис. 4 Применение ИНС для замены имитационной модели

Перед началом работы необходимо, используя имитационную модель, создать обучающую выборку и обучить ИНС. Причём, и генерацию обучающей выборки, и обучение приходится повторять при любом изменении модели, например, при её коррекции либо усложнении.

3.5.2. Генерация обучающей выборки

Генерация обучающей выборки является наиболее ресурсоемкой задачей при построении нейросетевой аппроксимации имитационной модели.

Для выбора узловых точек была применёна следующая схема.

Шаг 1. Генерация «граничных» узловых точек. Для каждого параметра выбирались: минимальное, максимальное и среднее (в статистическом смысле) значение, и брались все возможные их комбинации. Размер этой выборки для 3, 4 и 5 варьируемых параметров составлял 27, 81 и 243 соответственно.

Шаг 2. Генерация основного обучающего множества. На этом шаге значения параметров выбираются по разработанному алгоритму:

1. Пусть n – размерность пространства, N – число уже выбранных узлов. Задаётся некоторая константа $a \in [0,1]$, определяющая равномерность заполнения пространства узловыми точками. Значение a=0 соответствует неупорядоченному заполнению.

2. Случайным образом из диапазона допустимых значений выбирается набор параметров (точка многомерного пространства параметров).
3. Производится обход всех ранее сгенерированных узлов с вычислением расстояния r до нового узла.
4. Если выполняется условие

$$\min(r) > \frac{a}{\sqrt[n]{N}}, \tag{5}$$

где r – евклидово расстояние между рассматриваемым узлом и одним из раннее сгенерированных, то узел принимается.

5. Проверяется критерий останова. Если он не выполнен, алгоритм возвращается на шаг 2.

Ниже приведён результат работы такого алгоритма для n =2. Видно, что использование алгоритма позволило заполнить пространство значительно равномернее. Кроме того, заполнение осталось случайным и легко может быть продолжено.

Очевидно, что задача генерации обучающего множества может быть без труда распределена между компьютерами в кластере. В этом случае сервер выполняет поиск оптимальных узловых точек, а клиенты – вычисления значений в этих точках.

Результат выбора 100 узловых точек для функции 2-х переменных представлен на рис. 5.

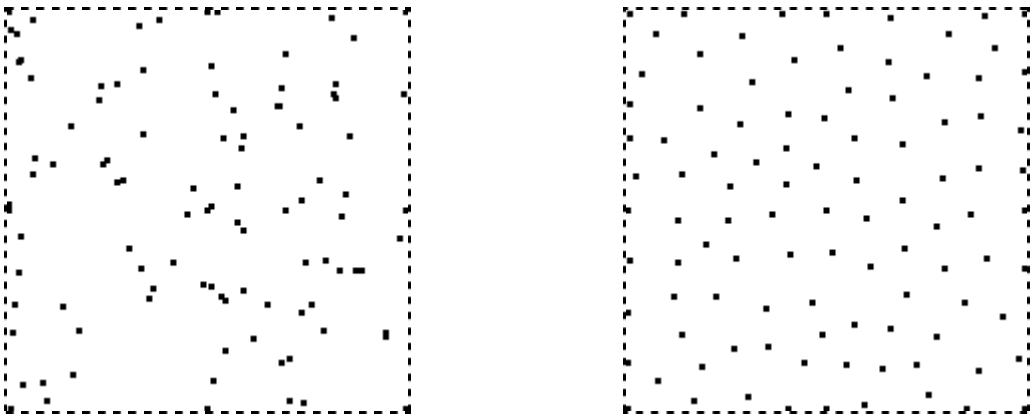


Рис. 5 Узловые точки в двумерном пространстве:
полностью случайный набор с a=0 (слева); неслучайный набор с a=0.8 (справа)

3.5.3. Параллельное обучение

При обучении нейронных сетей, в частности многослойных персептронов, существует две не решённые (в общем случае) проблемы:

1. Проблема выбора оптимальной конфигурации сети, т.е. числа скрытых слоёв нейронов и количества нейронов в каждом слое.
2. Проблема выбора начальных весовых коэффициентов. Как правило, начальные веса задаются случайным образом. В то же время эффективность алгоритмов обучения в значительной степени зависит от того, насколько удачным оказался выбор начальных весовых коэффициентов.

Для аппроксимации имитационных моделей было решено использовать трехслойные персептроны. Они показали наибольшую гибкость в сочетании с хорошими обобщающими свойствами. Поиск оптимального числа нейронов проводился методом локальных вариаций (использование более сложные методов не имело смысла из-за ограниченности решений). Поиск начинался из области малых размеров слоёв (например, по 4 нейрона в каждом) и постепенно сдвигался к оптимальному значению. Для вычисления невязки в узле осуществлялось обучение сети на небольшой, максимально репрезентативной выборке. Число итераций обучения равнялось числу настраиваемых весовых коэффициентов.

Начальные значения весовых коэффициентов выбирались подбором. При этом производилось несколько пусков алгоритма обучения с различными исходными весами (в опытах – порядка 100 пусков) и выбиралась конфигурация с лучшим результатом работы на некотором контрольном множества данных.

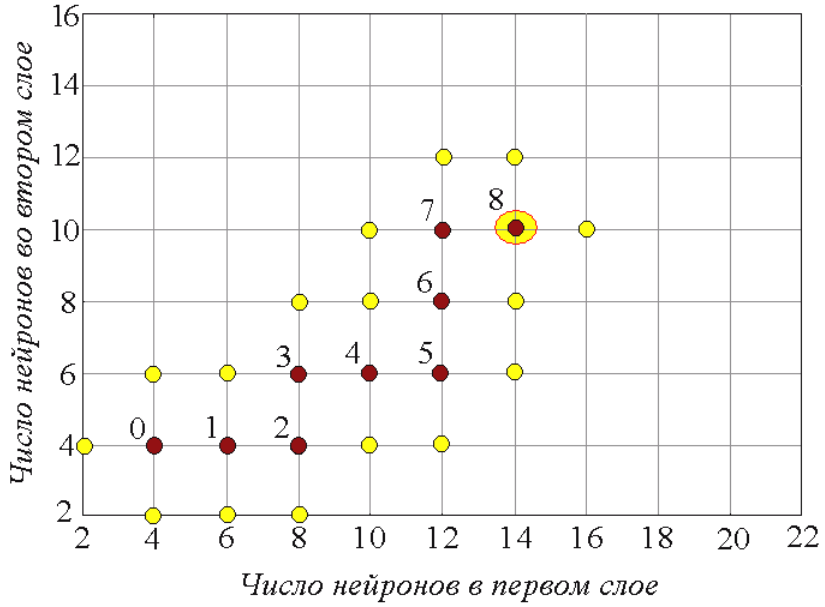


Рис. 6 Оптимизация числа нейронов трехслойного персептрона

Для каждой сети производилось 200 итераций. Операция поиска оптимальных весов выполнялась параллельно на нескольких машинах. В результате ошибка обучения снижалась в сравнении со средним значением в ~2.1 раза (и до 4.4 раз по отношению к наиболее неудачному выбору).

Результаты работы нейросетевой аппроксимации приведены в следующем разделе.

3.6. Параллельные методы оптимизации – генетические алгоритмы

3.6.1. Общие сведения о генетических алгоритмах

Генетические алгоритмы (ГА) – это стохастические, эвристические оптимизационные методы, впервые предложенные Холландом (Holland) в 1975 году [30]. Они основываются на идее эволюции с помощью естественного отбора.

ГА работают с совокупностью «особей» – популяцией, каждая из которых представляет возможное решение данной проблемы. Каждая особь оценивается мерой ее «приспособленности» согласно тому, насколько «хорошо» соответствующее ей решение задачи. В природе это эквивалентно оценке того, насколько эффективен организм при конкуренции за ресурсы. Наиболее приспособленные особи получают возможность «воспроизводить» потомство с помощью «перекрестного скрещивания» с другими особями популяции. Это приводит к появлению новых особей, которые сочетают в себе некоторые характеристики, наследуемые ими от родителей. Наименее приспособленные особи с меньшей вероятностью смогут воспроизвести потомков, так что те свойства, которыми они обладали, будут постепенно исчезать из популяции в процессе эволюции. Иногда происходят мутации, или спонтанные изменения в генах. Таким образом, из поколения в поколение, хорошие характеристики распространяются по всей популяции. Скрещивание наиболее приспособленных особей приводит к тому, что исследуются наиболее перспективные участки пространства поиска. В конечном итоге популяция будет сходиться к оптимальному решению задачи. Преимущество ГА состоит в том, что он находит приблизительные оптимальные решения за относительно короткое время.

Чтобы применять ГА к задаче, сначала выбирается метод кодирование решений в виде строки. Фиксированная длина (l-бит) двоичной кодировки означает, что любая из 2^l возможных бинарных строк представляет возможное решение задачи. По существу, такая кодировка соответствует разбиению пространства параметров на

гиперкубы, которым соответствуют уникальные комбинации битов в строке – хромосоме. Для установления соответствия между гиперкубами разбиения области и бинарными строками, описывающими номера таких гиперкубов, кроме обычной двоичной кодировки использовался рефлексивный код Грея. Код Грея предпочтительнее обычного двоичного тем, что обладает свойством непрерывности бинарной комбинации: изменение кодируемого числа на единицу соответствует изменению кодовой комбинации только в одном разряде.

Стандартные операторы для всех типов генетических алгоритмов это: селекция, скрещивание и мутация.

Оператор селекции (reproduction, selection) осуществляет отбор хромосом в соответствии со значениями их функции приспособленности. Существуют как минимум два популярных типа оператора селекции: рулетка и турнир.

Метод рулетки (roulette-wheel selection) отбирает особей с помощью *n* «запусков» рулетки. Колесо рулетки содержит по одному сектору для каждого члена популяции. Размер *i*-ого сектора пропорционален соответствующей величине целевой функции особи. При таком отборе члены популяции с более высокой приспособленностью с большей вероятностью будут чаще выбираться, чем особи с низкой приспособленностью.

Турнирный отбор (tournament selection) реализует *n* турниров, чтобы выбрать *n* особей. Каждый турнир построен на выборке *k* элементов из популяции, и выбора лучшей особи среди них. Наиболее распространен турнирный отбор с *k=2*.

Оператор скрещивания (crossover) осуществляет обмен частями хромосом между двумя (может быть и больше) хромосомами в популяции. Может быть одноточечным или многоточечным. Одноточечный оператор скрещивания работает следующим образом. Сначала, случайным образом выбирается одна из *l-1* точек разрыва. Точка разрыва – участок между соседними битами в строке. Обе родительские структуры разрываются на два сегмента по этой точке. Затем, соответствующие сегменты различных родителей склеиваются и получаются два генотипа потомков.

Мутация (mutation) – стохастическое изменение части хромосом. Каждый ген строки, которая подвергается мутации, с некоторой, достаточно малой вероятностью меняется на другой ген.

Работа ГА представляет собой итерационный процесс, который продолжается до тех пор, пока не выполнятся заданное число поколений или какой-либо иной критерий останова. На каждом поколении ГА реализуется отбор пропорционально приспособленности, кроссинговер и мутация.

Алгоритм работы простого ГА выглядит так как показано на рис. 7.

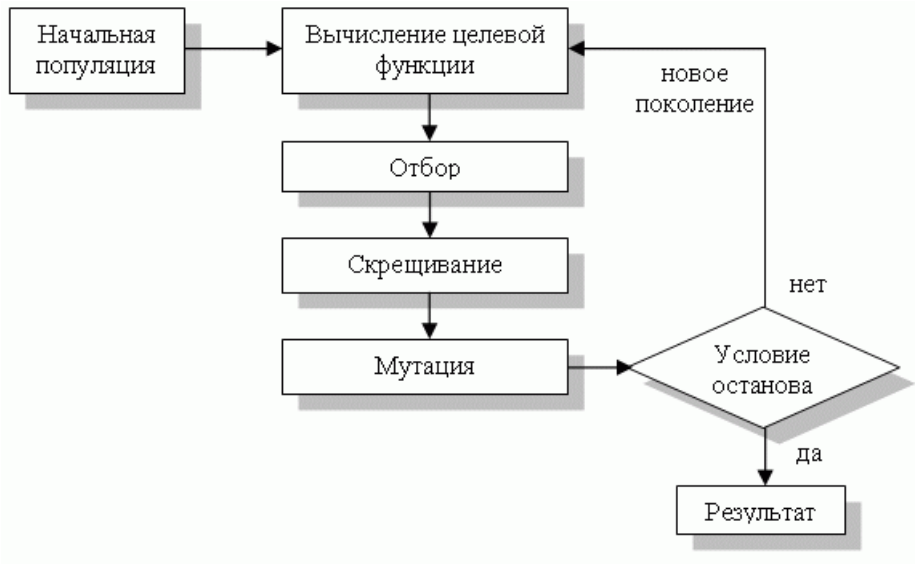


Рис. 7 Алгоритм работы классического ГА

3.6.2. Методы распараллеливания генетических алгоритмов

Генетические алгоритмы изначально создавались для применения в многопроцессорных системах, поэтому не составляет труда построить параллельную реализацию генетического алгоритма. Нами рассматривалось две модели распараллеливания ГА, представленные на рис. 8.

Первая схема хорошо зарекомендовала себя при распараллеливании в системах с быстрым обменом данными (например, Linux-кластер). При этом клиенты только переводят генотип группы особей из популяции в фенотип и вычисляют для них функцию пригодности. Все генетические операции (скрещивание, отбор, мутации) выполняются сервером.

Вторая схема может применяться в том случае, если связь между вычислительными узлами достаточно медленная (например осуществляется через Internet). При этом клиенты полностью поддерживают изолированные популяции, а сервер осуществляет только общий контроль, корректировку (обмен особями между популяциями) и анализ оптимизации. Этот подход открывает интересные перспективы прежде всего при оптимизации многомодальных функций. При приоритете близкородственного скрещивания в конце работы метода каждому найденному оптимуму будет соответствовать одна или несколько популяций. В том случае, если предпочтение будет отдаваться дальне родственному скрещиванию с интенсивным обменом особями, значительно повышается вероятность нахождения глобального оптимума.

а

б

Рис. 8 Подходы к распараллеливанию генетических алгоритмов:

а– распараллеливание при большом времени обсчёта и быстрой передаче данных;

б –распараллеливание при малом времени обсчёта или медленной передаче данных

3.7. Выводы

В этом разделе были рассмотрены построенные алгоритмы, предназначенные для распараллеливания различных этапов идентификации параметров сложных процессов и систем. Наряду с такими подходами, как параллельное имитационное моделирование и методы параллельного генетического поиска был предложен принципиально новый метод нейросетевой аппроксимации, позволяющий значительно сократить время идентификации неизвестных параметров процессов и систем. Рассмотренные в этом разделе методы и алгоритмы использовались на практике. Результаты их применения представлены в разделе 4.

4. РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ ОБРАБОТКИ ДАННЫХ

4.1. Распределенное имитационное моделирование

4.1.1. Система имитационного моделирования PARSIM

Программа создана для организации параллельного ИМ на компьютерах под операционными системами Windows 9X/NT/XP, соединенных в локальную сеть.

Имитационная модель представляется файлом динамически компонуемой библиотеки (dynamic-link library). Помимо самой модели, этот файл содержит также различную служебную информацию: версию файла, описание модели и её параметров, пределы допустимых значений параметров модели и их количество.

Роли в системе распределены следующим образом: на одном из компьютеров запускают серверную часть программы (далее просто «сервер»), с которой будет работать пользователь, на остальных – клиентскую часть (клиенты), которая и выполняет вычисления.

4.1.2. Порядок работы с системой

После запуска сервера пользователь выбирает файл, содержащий нужную модель. Если загрузка файла модели прошла успешно, в окне программы отображается его версия, а также описание загруженной имитационной модели, представленное на рис. 9.

После успешной загрузки файла модели пользователь может задать параметры моделирования вручную, либо загрузить их из файла. Затем, нажатием кнопки Listen пользователь переводит сервер в режим ожидания соединений с клиентами.

Далее, на каждом из компьютеров, предназначенных для проведения распределенных вычислений, и соединенных с сервером локальной сетью, запускается клиентская часть программы. При запуске клиент пытается установить соединение с сервером. Это происходит до тех пор, пока соединение не будет установлено, либо пока клиент не будет закрыт вручную.

Когда соединение между клиентом и сервером установлено, сервер передаёт клиенту заданные пользователем параметры и нажатием кнопки Start simulation запускает моделирование.

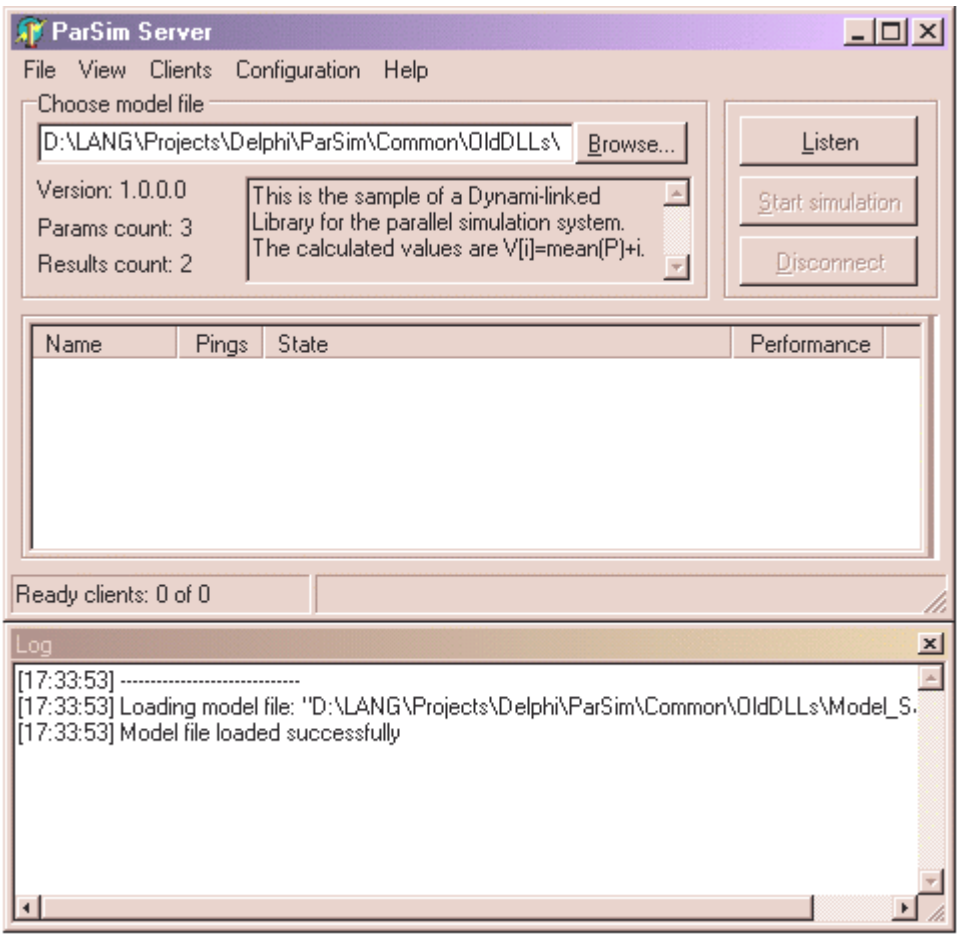


Рис. 9 Внешний вид серверной части системы ParSim

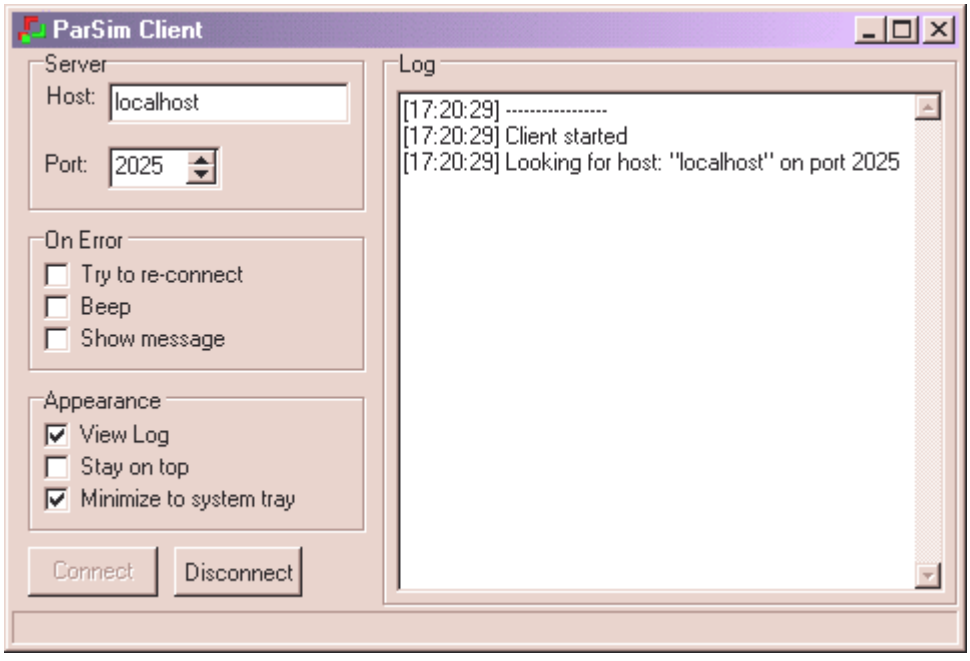


Рис. 10 Внешний вид клиентской части системы ParSim

В ходе работы пользователь имеет полную информацию о текущем состоянии каждого клиента. Если произошло зависание клиента или сеть сильно загружена, пользователь получит сообщение о том, что клиент в течение долгого времени (порядка нескольких секунд) не подаёт «признаков жизни». Пользователь может дать серверу указание либо некоторое время подождать отзыва клиента, либо разорвать соединение с этим клиентом.

Если в процессе вычислений происходит критическая ошибка в работе клиента, пользователь получает исчерпывающую информацию о произошедшей ошибке, а клиент пытается повторить вычисления. Если ошибка повторяется, пользователь может разорвать соединение с данным клиентом.

Если по каким-либо причинам пользователь пожелал прервать процесс моделирования, он нажимает кнопку Abort simulation, сервер уведомляет об этом всех клиентов, вычисления прекращаются, и клиенты переходят в режим ожидания команд с сервера.

После выполнения каждого одиночного моделирования клиент отсылает полученные результаты на сервер и, если моделирование было произведено нужное количество раз, переходит в режим ожидания команд с сервера. Такому клиенту можно передать новый набор параметров и снова запустить процесс моделирования.

Когда все клиенты завершили вычисления, пользователь может сохранить полученные результаты в файл или скопировать их в буфер обмена (clipboard), после чего процесс моделирования может быть повторен для другой модели, либо с иными параметрами.

Все изменения состояния системы, а также процесс передачи данных подробно протоколируется. Пользователь может просмотреть протокол работы, а также сохранить его в файл, либо скопировать в буфер обмена.

4.1.3. Программная реализация

Пакет ParSim реализован в среде разработки Borland Delphi 6. Для работы с сетью была использована технология Windows Sockets.

Ядро системы составляют компоненты TParSimServerSocket и TParSimClientSocket, являющиеся расширением стандартных компонент TServerSocket и TClientSocket, входящих в набор Borland Standard Components.

Двусторонний обмен информацией основан на системе сообщений. Сообщение представляет собой пакет данных, в котором находится идентификатор сообщения, сопровождаемый соответствующими данными.

Вся работа по приёму, передаче и обработке сообщений возложена компоненты ядра. Внутренняя работа компонент реализована с расчётом на большую универсальность, чем на данный момент используется. В частности, TParSimServerSocket работает с каждым клиентом независимо, что позволяет в случае необходимости легко расширить функциональность системы, например, передавать клиентам различные модели и различные параметры моделирования, вести протокол работы с каждым клиентом в отдельности и т.п.

Программы пакета ParSim являются многопоточными, т.е. обмен данными, обсчёт модели и обработка пользовательского интерфейса ведётся в нескольких параллельно выполняющихся потоках (execution thread).

Сервер создает такой поток для каждого клиентского соединения. Это позволяет сделать обмен данными с каждым клиентом независимым от остальных.

В течение работы системы сервер поддерживает список всех клиентов (в том числе и уже отсоединённых). В этом списке для каждого клиента хранятся:

- уникальный идентификационный номер клиента;
- имя машины, на которой он запущен;
- производительность клиентской машины;
- информация о текущем состоянии клиента (принимает параметры, производит вычисления, передает результаты, отсоединён);
- код произошедшей ошибки и её описание;
- причина отсоединения (по указанию с сервера, из-за ошибки, из-за выключения клиентской машины);
- переданные клиенту параметры;
- полученные от него результаты;
- количество полученных от клиента «still-alive» сообщений и время получения последнего из них.

Доступ ко всем полям синхронизирован посредством критических секций.

Таким образом, пользователь имеет всю необходимую информацию о каждом клиенте и может контролировать работу системы.

Поскольку процесс моделирования занимает значительное время, каждый клиент периодически (раз в 5 секунд) отсылает на сервер «still-alive» сообщение. Это позволяет пользователю контролировать активность клиентов. Если клиент в течение долгого времени не передает никаких сообщений, это свидетельствует либо о перегруженности сети, либо о неполадках на клиентской машине.

Разработка интерфейса клиента велась с учётом требования малой заметности для пользователя клиентской машины. Таким образом, на клиентской машине может вестись обычная работа. О том, что машина используется клиентом системы ParSim, будет свидетельствовать только иконка в области системного трэя (рис. 11).

Рис. 11 Клиент ParSim в свёрнутом виде

Конфигурация клиента и сервера (адреса, порты, положение окон и т.д.) сохраняется в ini-файлах, а не системном реестре, что упрощает перенос и инсталляцию программ пакета ParSim на нескольких компьютерах.

Информация о файле модели (версия, описание модели, авторские права) записана в области VersionInfo DLL-файла, благодаря чему эту информацию можно просмотреть стандартными средствами Windows.

4.1.4. Динамические библиотеки

Как уже упоминалось, имитационная модель представляется файлом динамически компонуемой библиотеки. Каждая библиотека создаётся по определённому шаблону и содержит следующие функции:

- int **GetParamNum** (void), int **GetValueNum** (void) – функции возвращают количество входных параметров и выходных значений описываемой модели;
- double **GetMaxAllowedParam** (int ip), int **GetMinAllowedParam** (int ip) – функции возвращают минимально и максимально допустимые значения параметра под номером *ip*;
- int **CheckParamRange** (double *P) – проверяет, попадают ли все параметры из вектора *P* в свою область определения;
- int **CheckParamSet** (double *P) – проверяет, является ли набор входных параметров совместимым. Эта функция является более жёстким вариантом предыдущей;
- int **GetParamName** (int ip, char *lpName) – функция передаёт в строку lpName название входного параметра ip. Возвращает длину строки;
- int **GetDefaultParam** (double *P) – передает в вектор P набор параметров «по умолчанию»;

- int **Simulate** (double *P, double *V) – непосредственно функция моделирования. Принимает входные параметры из вектора P, производит моделирование и выдает результат по адресу вектора V. В случае сбоя возвращает ненулевое значение.

Кроме этих функций в каждой библиотеке должна содержаться информация о числе входных параметров и выходных значений, область определения для каждого параметра, набор параметров «по умолчанию» и строки с их названиями.

Используя представленный выше шаблон, пользователь может самостоятельно создавать библиотеки моделей и подключать их в систему ParSim. В нашей работе библиотеки создавались в среде разработки MS Visual C++ 6.0.

4.1.5. Результаты работы системы ParSim

Для оценки эффективности распараллеливания ИМ был проведён следующий вычислительный эксперимент. В эксперименте использовались 17 ПК класса Pentium III (16 клиентов и сервер), объединённых в локально-вычислительные сети (ЛВС) под управлением WindowsNT. Скорость передачи данных в сети составляла 100 Мбит/с.

Для тестирования была выбрана имитационная модель переноса энергии в системах мембранных протеинов. Среднее время одного обсчёта модели составляло порядка 10 с. Для получения статистически справедливого результата требовалось произвести не менее 100 пусков ИМ.

В качестве параметра оценки параллельно моделирования было выбрано ускорение, т.е. отношение времени необходимого для получения результата на одном компьютере к времени получения результата при вычислениях в сети. Число пусков моделирования в эксперименте варьировалось от 1 до 1000. Результаты эксперимента представлены на рис. 12.

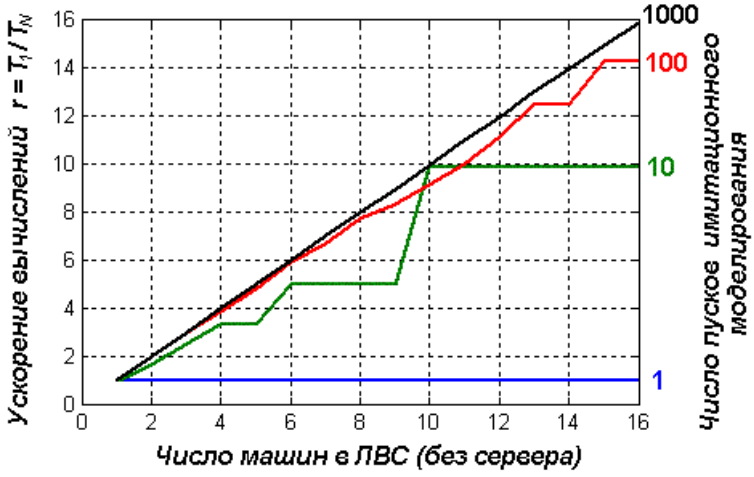


Рис. 12 Ускорение обсчёта имитационной модели в ЛВС

При больших значениях числа пусков ИМ поведение ускорения удовлетворяет закону Амдала. Отклонение при малых значениях связано с высоким порядком дискретности задачи. Например, время, необходимое для выполнения на 10 компьютерах 11 пусков будет равно времени, необходимого для выполнения 20 пусков ИМ.

4.2. Результаты применения нейросетевой аппроксимации

4.2.1. Моделирование процессов релаксации флуоресценции

Работа ИНС-аппроксиматора предварительно тестировалась на аналоге имитационной модели релаксации энергии молекулами различных типов. В том случае, когда в такой системе пренебрежимо малы перенос и безызлучательная релаксация, результирующая кривая флуоресценции может быть представлена много экспоненциальной функцией вида:

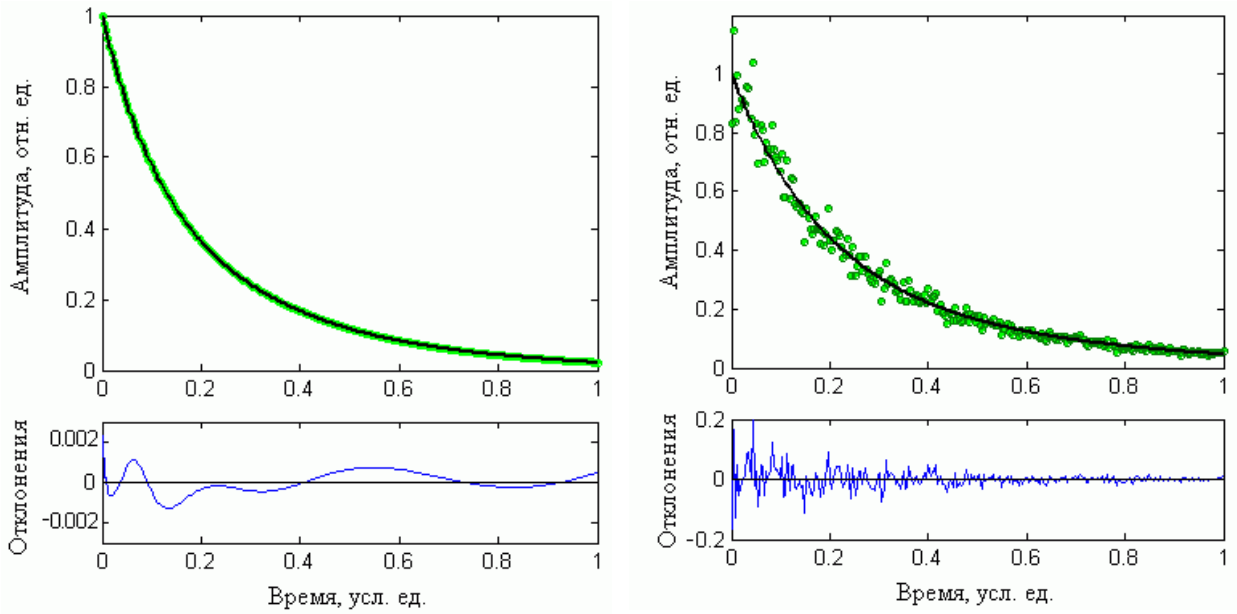
$$f(t,\mathbf{p}) = \sum_{i=0}^{n} p_{2i} \cdot e^{-t \cdot p_{2i+1}} + noise, \qquad (6)$$

где t – время от начала флуоресценции, p – параметры флуоресценции, причём число экспонент n совпадает с числом типов излучающих молекул. Именно функция (6) выступала в роли имитационной модели в наших экспериментах. Для аппроксимации (при n=2) был использован трехслойный персептрон с 10 нейронами в каждом слое. В ходе проведённых вычислительных экспериментов было установлено, что:

1. Оптимальный выбор узловых точек аппроксимации позволяет значительно (на порядок) снизить ошибку аппроксимации при постоянном размере обучающей выборки.
2. В результате итеративного подбора начальных весов сети ошибка обучения снижалась в сравнении со средним значением в ~2 раза (и до 4.4 раз по отношению к наиболее неудачному выбору).

Результаты аппроксимации представлены на рис. 13. Видно, что при аппроксимации зашумлённых функций коррелированные ошибки составляют незначительную величину (на рисунке ~1%) в сравнении со стохастическими ошибками.

Таким образом, результатом работы ИНС является гладкая функция, аппроксимирующая стохастический набор экспериментальных данных, либо данных имитационного моделирования.



а *б*

Рис. 13 Нейросетевая аппроксимация двухэкспоненциального затухания:

а – при отсутствии; *б* – при наличии шума

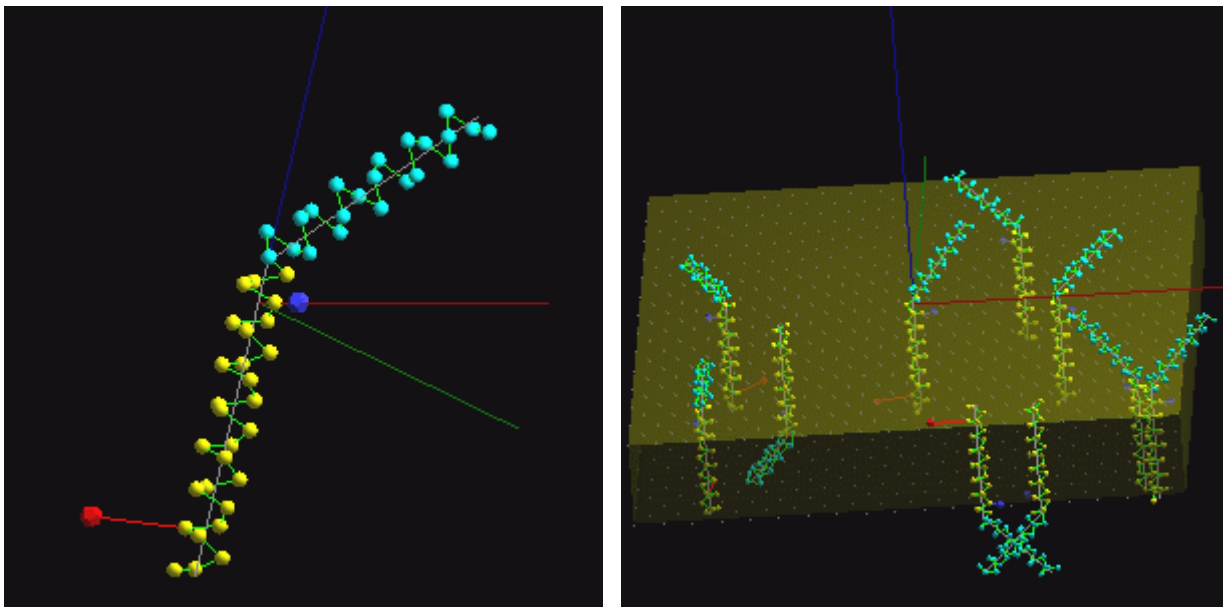
4.2.1. Моделирование процессов переноса энергии в системах мембранных протеинов

Метод нейросетевой аппроксимации в данной работе использован для моделирования и анализа процессов диполь-дипольного переноса энергии между флуоресцентными метками мембранных протеинов [31].

При построении формализованной модели каждый протеин заменялся моделью представленной на рис. 14, *а*. Липидная мембрана аппроксимировалась структурой, приведённой на рис. 14, *б*.

В биоорганической химии для определения расстояний внутри и между различными макромолекулами часто используется метод так называемой спектроскопии резонансного переноса энергии. Суть метода заключается в следующем. В исследуемые макромолекулы (в данном случае – мембранные протеины) внедряют флуоресцентные зонды двух типов: доноры (D), со сравнительно большим временем жизни в возбуждённом состоянии и акцепторы (A). Доноры возбуждаются внешним источником освещения. Часть из них передают свою энергию акцепторам, которые ее излучают с большей длиной волны.

Вероятность переноса энергии в системе донор-акцептор обратно пропорциональна 6-ой степени расстояния между ними и может быть вычислена по следующим формулам.



а *б*

Рис. 14 Модель мембранного протеина:

а – мембрана с флуоресцентными зондами; *б* – липидная мембрана

В случае одного акцептора

$$p_{DA} = \frac{1}{1 + \left(r_{DA} / R_0\right)^6} \tag{7}$$

где P_{DA} – вероятность переноса энергии в случае одного акцептора, r_{DA} – расстояние между донором и акцептором, R_0 – константа (расстояние Форстера), характеризующая перенос энергии для данных флуоресцентных зондов.

В случае ансамбля акцепторов:

$$p = \frac{\sum_i \left(R_0 / r_i\right)^6}{1 + \sum_i \left(R_0 / r_i\right)^6}, \tag{8}$$

где r_i – расстояния до донора от i-го акцептора.

В имитационной модели, после создания 3-х мерной модели мембраны рассчитывались вероятности переноса для всех доноров (согласно (8)), после чего вычислялось среднее этих значений. Такая интегральная величина получила название *эффективности переноса энергии*.

Для замены имитационной модели мы использовали трёхслойный персептрон. Необходимое число нейронов в слоях устанавливалось экспериментальным путём. ИНС обучалась методом обратного распространения ошибки. Для избежания переобучения каждые 10 итераций работа сети тестировалась на контрольном

множестве (200 элементов). Если ошибка при этом не уменьшалась в течение некоторого времени, обучение останавливалось. В результате погрешность аппроксимации составила порядка 2%, что с учётом зашумленности данных является весьма хорошим показателем.

Генерация обучающих пар и поиск оптимальных начальных весов ИНС производились в ЛВС под управлением Windows NT.

4.2.3. Оценка эффективности метода

Применение нейросетевой аппроксимации вместе с параллельным моделированием на этапе обучения ИНС позволило ускорить получение результата при идентификации экспериментальных систем в 10^5. Следует отметить, что результатом работы ИНС является гладкая функция, аппроксимирующая стохастические результаты имитационного моделирования. Это позволяет использовать градиентные методы оптимизации при идентификации системы.

К недостаткам метода можно отнести то, что количество варьируемых параметров ограничено. Максимальное число параметров, при котором метод работал на нашей задаче, равнялось шести (лучший результат получен при 8-ми варьируемых параметрах). Это ограничение связано с так называемым «проклятием размерности» когда число узлов, необходимых для обучения сети, экспоненциально растёт с ростом числа параметров.

Тем не менее, метод всегда может быть использован для быстрого получения хороших результатов при оптимизации и изучения функциональных зависимостей систем с небольшим числом варьируемых входов.

4.3. Реализация кластера в среде ASPLinux

С использованием пакета MPICH в среде ASPLinux построен учебный кластер на базе двух компьютеров Pentium, соединенных через сетевые карты. Особенности кластера заключались в использовании компьютеров с различной производительностью и недорогих сетевых карт Ethernet со скоростью передачи 10 Мб/с, широко применяемых в компьютерных классах учебных заведений.

В качестве тестовой задачи использовалась программа **cpi** для расчета числа «пи» [3]. Вычисления производились с точностью до 16 знаков после десятичной точки. Время выполнения тестовой задачи в различных условиях приведено в табл. 3.

Таблица 3

№ п. п.	Количество процессов	Время выполнения, мкс	Примечания
1	1	110.22	P1-100
2	1	100.29	P2-450
3	2	414.82	P2+P2
4	2	2263.76	P1+ P1
5	4	1148.86	P2+ P2
6	2	1705.44	P1+ P2
7	4	7647.86	P1+P1
8	4	4696.97	P1+P2

Полученные результаты, представленные в табл. 3, указывают на большие потери времени на обмен данными между компьютерами Pentium I (P1) и Pentium II (P2),, входящими в кластер.

В итоге относительно высокие затраты ресурсов компьютеров на организацию параллельных процессов не позволяет получить ожидаемого ускорения вычислительного процесса.

4.4. Выводы

В разделе представлен созданный в рамках проекта программный продукт ParSim, позволяющий распараллеливать имитационное моделирование в ЛВС под управлением Windows NT/XP. При этом имитационная модель представляется в виде функции динамически подключаемой библиотеки и может быть модифицирована пользователем. [32]

Также были рассмотрены результаты нейросетевой аппроксимации, когда при идентификации системы ее имитационная модель заменяется нейросетевой моделью. При этом самый ресурсоемкий этап – генерация обучающих пар производился параллельно в ЛВС. В результате применения нейросетевой аппроксимации для моделирования процессов переноса энергии в сложных биомолекулярных комплексах (мембранные протеины) удалось снизить время идентификации параметров в 10^3-10^4 раз [31].

В результате выполнения работы на кафедре системного анализа БГУ создан учебный Linux-кластер, состоящий из 2-х машин классов Pentium / Pentium II**.**

ЗАКЛЮЧЕНИЕ

Результаты проведенных исследований показывают целесообразность использования параллельных вычислений на всех этапах идентификации параметров сложных процессов и систем.

Были разработаны методы анализа и построены алгоритмы обработки данных в вычислительных системах с распределёнными ресурсами. Построено программное обеспечение, позволяющее распараллеливать имитационное моделирование в вычислительных сетях. Разработаны алгоритмы идентификации параметров сложных стохастических процессов и систем в вычислительных системах с распределёнными ресурсами. В рамках проекта был разработан метод нейросетевой аппроксимации имитационных моделей сложных стохастических систем. При применении этого метода на практике ускорение идентификации параметров доходило до 10^4.

Построенные алгоритмы параллельного моделирования и оптимизации будут применяться для анализа фотофизических процессов в сложных биомолекулярных образованиях (мембранные протеины) на кафедре системного анализа БГУ и в лаборатории биофизики Вагенингенского университета, Нидерланды (Laboratory of Boiphysics, Wageningen University, The Netherlands).

Главный результат проведенных исследований состоит:

- в разработке метода идентификации параметров процессов и систем с помощью нейросетевой аппроксимации, обеспечивающего значительное ускорение вычислений;
- в построении системы параллельного имитационного моделирования, позволяющей распараллеливать вычисления в ЛВС под управлением Windows NT/XP.
- в исследовании ограничений на применение гетерогенных кластеров с малым количеством компьютеров.

Результаты работы показывают также, что ускорение вычислений при использовании кластера можно достичь только при использовании быстродействующих каналов связи между компьютерами и достаточно большом количестве машин в кластере. Однако дальнейшее увеличение числа компьютеров (ориентировочно от 50 до 100) уже не будет экономически выгодным. Если же такие мощности действительно необходимы, выгоднее приобрести суперкомпьютер.

Таким образом при относительно небольшом количестве компьютеров в кластере с различным быстродействием предложенные методы организации распределенных вычислений более эффективны.

Полученные результаты докладывались на трех международных, а также 59-й научной конференции студентов и аспирантов Белорусского государственного университета.

Разработанные алгоритмы апробированы в лаборатории биофизики Вагенингенского университета, Нидерланды и на кафедре системного анализа ФРЭ БГУ.

Основные результаты проведенных исследований опубликованы в работах.

- Nazarov, P.V., Apanasovich, V.V., Lutkovski, V.M., Hemminga, M.A., Koehorst R.B.M., Neural Network Simulation of Energy Transfer Processes in a Membrane Protein System, Advances in Soft Computing: Neural Networks and Soft Computing, Springer, (2002), pp. 873–888 (в печати, дата опубл. – январь 2003).
- Назаров П.В., Лутковский В.М., Поплетеев А.М., Идентификация процессов и систем с использованием параллельного имитационного моделирования и нейросетевой аппроксимации, материалы I Международной конференции «Информационные системы и технологии», (IST-2002), Минск, 2002. В 2-х частях. Ч. II. С. 142 – 146.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы, математическое обеспечение и архитектура многопроцессорных вычислительных систем / Под ред. А. П. Ершова. М.: Наука, 1982. 336 с.
2. Антонов А. Компьютерра, 2002, №5 (430), с. 24–27.
3. Богданович П., Попов М. Компьютерра, 2002, №5 (430), с. 31–33.
4. Буза М. К. Введение в архитектуру компьютеров. Минск: БГУ, 2000 г.
5. Воеводин В. В. Математические модели и методы в параллельных процессах. М.: Наука,1986. 296с.
6. Воеводин В. В., Воеводин В. В. Параллельные вычисления. М.: 2002.
7. Воеводин В. В. Суперкомпьютерная грань компьютерного мира // Байт. 2000, №4.
8. Воеводин В. В. Компьютерра, 2002, № 5 (430). С. 26–27.
9. Хокни З., Джессхоуп К. Параллельные ЭВМ. М.: Радио и связь, 1986. 389 с.
10. Зимянин Л.Ф. Разработка и исследование системы параллельного программирования для многомодульных конфигураций. Диссертация к.т.н. Мн.: БГУ, 1994. 156 с.
11. Михайленко К. Компьютерра, 2002, №5 (430), с. 28–30.
12. Шпаковский Г. И. Параллельные микропроцессоры для цифровой обработки сигналов и медиа данных. Мн.: БГУ, 2000. 196 с.
13. Шпаковский Г. И. Организация параллельных ЭВМ и суперскалярных процессоров: Учеб. пособие. Мн.: Белгосуниверситет. 1996. 284 с.
14. Программирование для многопроцессорных систем в стандарте MPI / Г. И. Шпаковский, Н. В. Серикова. Мн.: БГУ, 2002. 323 с.
15. Галушкин А. И. Теория нейронных сетей. Кн. 1: Учебное пособие для вузов. Общ. ред. А. И. Галушкина. М: ИПРЖР, 2000. 416 с.
16. Subrahmanyam Allamaraju. Approaches for e-commerce architectures. 2001. Website: www.javable.com
17. Интероперабельные информационные системы / Д. О. Брюхов, В. И. Задорожный, Л.А. Калиниченко, М.Ю. Курошев, С.С. Шумилов// СУБД. 1995, № 4.

18. Ладыженский Г. Middleware: модель сервисов распределенных систем. 2001. Website: www.interface.ru

19. Ладыженский Г. Системы управления базами данных – коротко о главном. 1998. Website: www.interface.ru

20. Пушников А.Ю. Введение в системы управления базами данных. Часть 1. Реляционная модель данных: Учебное пособие. Уфа: Изд. Башкирского ун-та. 1999.

21. Alex Chaffee. One, two, three, or n tiers? // JavaWorld. 2000, №1.

22. Семихатов С. А. Технологии WWW, Corba и Java в построении распределенных объектных систем. 1998. Website: www.javapower.ru

23. Чертков Б. З., Сапцин Н.В. Технологии «клиент-сервер». Website: www.uic.ssu.samara.ru

24. Зеленков Ю.А. Введение в базы данных, 1997. Website: www.yars.free.net

25. Loura L.M.S., et al., Biophysical Journal, 80 (2001), 776–788.

26. Andrews L., Demidov A., ets. Resonance Energy Transfer, John Wiley & Sons Ltd Inc: New York, 1999.

27. Flecher R. Practical Methods of Optimization John Willey & Sons Inc., New York, 1987

28. Hornik K., Stinchcombe M., White H., Multilayer feedforward networks are universal approximators, Neural Networks 2, (1989) 359–366.

29. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика. Пер. с англ. Ю. А. Зуева и В. А. Точенова. М.: Мир, 1992. 184 с.

30. Davis L. The handbook of Genetic Algorithms. New York, 1991.

31. Nazarov, P.V., Apanasovich, V.V., Lutkovski, V.M., Hemminga, M.A., Koehorst R.B.M., Neural Network Simulation of Energy Transfer Processes in a Membrane Protein System, Advances in Soft Computing: Neural Networks and Soft Computing, Springer, (2003), pp. 873-888.

32. Назаров П.В., Лутковский В.М., Поплетеев А.М., Идентификация процессов и систем с использованием параллельного имитационного моделирования и нейросетевой аппроксимации, материалы I Международной конференции "Информационные системы и технологии", (IST-2002), Минск, 2002. В 2-х частях. Ч. II. С. 142 – 146.