МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Направление подготовки	**09.04.02**
Направленность (профиль)	**Информационные системы и технологии «Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2023

СОДЕРЖАНИЕ

1. ЦЕЛЬ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Целью освоения дисциплины «Вычислительные системы» является получение магистрантами знаний о принципах организации современных вычислительных систем, их структуре и функционировании.

Основные задачи дисциплины:

- ознакомление с важнейшими этапами и тенденциями в развитии вычислительных систем;
- ознакомление с методами оценки параметров компонент и систем в целом;
- приобретение теоретических знаний и практических навыков выбора и использования вычислительных систем для обработки информации.

2. КОМПЕТЕНЦИИ ОБУЧАЮЩЕГОСЯ, ФОРМИРУЕМЫЕ В РЕЗУЛЬТАТЕ ИЗУЧЕНИЯ ДИСЦИПЛИНЫ

Индекс	Формулировка:
ПК-1	способность осуществлять управление, развитием баз данных, включая развертывание, сопровождение, оптимизацию функционирования баз данных, являющихся частью различных информационных систем
ПК-4	способность выполнять разработку систем управления базами данных, операционных систем, организацию разработки системного программного обеспечения, интеграция разработанного системного программного обеспечения

В результате освоения дисциплины обучающийся должен:

ЗНАТЬ	- фундаментальные основы вычислительных систем и технологий, необходимых для успешного освоения новых инструментов в рамках профессиональной деятельности. - методы и средства получения, хранения, переработки и трансляции информации.
УМЕТЬ	- использовать современные компьютерные технологии как средство получения, хранения, переработки и трансляции информации. - использовать в практической деятельности новые знания и умения, как в рамках профессиональной деятельности, так и в новых областях знаний, непосредственно не связанных со сферой деятельности.
ВЛАДЕТЬ	- практическими навыками использования информационных технологий в профессиональной деятельности. - методами и средствами получения, хранения, переработки и трансляции информации посредством современных компьютерных технологий, в том числе, в глобальных компьютерных сетях.

3. СОДЕРЖАНИЕ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа 1.

Оценка производительности ВС с учетом возможных отказов отдельных подсистем.

Цель и содержание: Проанализировать существующие методы оценки ВС. Выявить показатели производительности ВС. Провести оценку компьютерных систем. Подвести итоги проведенного исследования.

Организационная форма занятий: лабораторное занятие (исследование).

Вопросы для обсуждения на лабораторном занятии: Анализ текущего состояния исследований в области оценки производительности параллельных и распределенных вычислительных комплексов. Характеристики и критерии для оценки производительности вычислительной системы.

Теоретическое обоснование

Оценка производительности сложная, но необходимая задача как при исследовании работающих, так и при проектировании новых вычислительных систем. Для работающих систем она позволяет выявить и устранить ”узкие” места (подсистемы или компоненты системы, на которые приходится максимальная нагрузка), тем самым, повысив надежность работы. Для вновь проектируемых, она помогает выбрать необходимое оборудование, соответствующее планируемым задачам, рассчитать рабочую нагрузку, при которой система будет работать наиболее эффективно. Оценка производительности особенно актуальна для высокопроизводительных (MPI,GRID) и кластерных систем высокой готовности со сложной архитектурой и топологией, то есть для систем, где приложения могут распределяться между узлами или мигрировать с одного узла на другой.

В 80-ые годы для оценки производительности вычислительных систем использовали метод Гибсона, но он отражает только быстродействие оборудования и неприменим к оценке качества выполнения задач и, кроме того, оставляет открытым выбор коэффициентов для ”смесей”. В настоящее время используется несколько методов оценки производительности систем. Наилучшим из них является испытание системы при реальной рабочей нагрузке, однако в случаях, когда этот подход оказывается неприемлемым, прибегают к эталонному тестированию — методу определения эффективности системы при определенных (эталонных) нагрузках. Эталонные тесты (benchmarks) можно разделить на двекатегории: широкого (оценивают эффективность системы в целом по ряду критериев: например SPEC, TPC) и узкого применения (оценивают эффективность компьютерных сервисов или системных компонентов: например Linpack, I/Ozone, Webstone). Из-за многообразия компьютерных архитектур, операционных систем и различия решаемых задач, на сегодняшний день отсутствуют универсальные методы оценки производительности. В любом случае, она опирается на специфику архитектуры компьютера, где каждый системный компонент (процессор, память, шины и.т.д.) имеет свое выражение в переменных и структурах ядра операционной системы. Набор значений этих переменных, каждая из которых отражает состояние одной из подсистем, характеризует состояние системы в определенный момент времени. Исследуя изменение этих характеристик, можно определить ее поведение на любом временном интервале. Ниже предлагается метод оценки производительности вычислительной системы, основанный на выборе этих переменных, их предельных значений и весовых коэффициентов с последующей сверткой этих значений в интегральный показатель.

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер с характеристиками, позволяющими установить операционную систему ОС UNIX.

Указания по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения, ремонт персонального компьютера. Соблюдать правила технической безопасности при работе с электрооборудованием.

Методика и порядок выполнения работы

Выберем необходимые характеристики и критерии для оценки производительности вычислительной системы, работающей под управлением ОС UNIX SysV.

Процессорная подсистема.

Процессорная подсистема, прежде всего, характеризуется показателями загруженности (процентом утилизации) центрального процессора (ЦП): %usr (процент времени на выполнение пользовательских задач), %sys (процент системного времени), %wio (процент времени на ожидание ввода-вывода блочных устройств), %idle (процент времени простоя ЦП). Очевидно, что суммарная утилизация ЦП не превышает 100%: *%usr%wio%sys%idle*≤100%. Для нормально функционирующей системы, как правило, должны выполняться условия: *%wio* <=30%, *%usr %wio %sys* <=70%,*%sys* <=30% .

Показатель %sys может меняться широких пределах, от типовых 10%-30%, до 50% и более. Это связано с тем, что некоторые сетевые сервисы (например, NFS) являются частью ядра, и любая активность NFS относится не к показателю пользовательской загрузки, а к показателю %sys, следовательно, его нельзя использовать для оценки производительности. Показатель загрузки %wio не должен превышать 30%, в противном случае можно с уверенностью говорить о низкой пропускной способности (а следовательно и производительности) системы ввода-вывода. Суммарный процент утилизации в нормально загруженной системе не должен превышать 70%-80%, так как при увеличении загрузки ЦП увеличивается очередь процессов, ждущих обслуживания, а значит и время их выполнения. Длина очереди процессов определяется характеристикой runq-sz и является более точным показателем загруженности ЦП, чем суммарная утилизация. Большое значение runq-sz указывают на то, что процессор не успевает обслуживать прикладные процессы, то есть для нормальной работы системы необходимо выполнение условия: runq -sz <=4.

Дополнительно можно упомянуть параметры, показывающие частоту системных вызовов, выполняемых ЦП. В UNIX-системах есть возможность разделения общей статистики системных вызовов (scall/s) на вызовы чтения/записи (sread/s, swrit/s), отражающие активность ввода-вывода, и fork/exec-вызовы (fork/s, exec/s), характерные для командных интерпретаторов. По количеству системных вызовов сложно судить о производительности, но можно получить общее представление об уровне активности и типе исследуемой системы.

Память.

При оценке работы памяти, казалось бы, лучше всего ориентироваться на значения свободной и занятой (используемой) памяти. Однако, это невозможно, потому что практически вся память при загрузке операционной системы разбивается на страницы и, далее, попадает в свободный список, отводится под файловый КЭШ или резервируется ядром. Поэтому значения параметров свободной и используемой памяти не отражает реальные процессы и при исследовании производительности они бесполезны.

Для определения характеристик памяти, действительно важных при оценке ее производительности, необходимо рассмотреть механизмы ее работы. В случае, когда все приложения (включая и ядро) одновременно запрашивают объем памяти, который превышает доступный, системе приходится прибегать к механизму подкачки (записи неиспользуемых страниц памяти на диск). В случае, если ситуация с нехваткой памяти усугубляется, на диск записываются все участки памяти, используемые процессом или

процессами. Этот механизм называется свопингом. В современных UNIX-системах механизм свопинга начинается только в случае критической не хватки памяти, поэтому все параметры, показывающие его активность важны при оценке производительности системы. Достаточно часто встречаются ситуации, при которых производительность резко падает, но свопинг при этом не начинается. Такое “пограничное” поведение системы возможно при определенной, но некритичной нехватке памяти. В подобных случаях ядро включает механизм страничного сканирования, который пытается освободить страницы памяти с наибольшим временем простоя. Этому механизму в UNIX соответствует параметр частоты сканирования страниц. Активность механизма страничного сканирования может значительно снижать производительность системы.

Загрузку памяти представляют следующие характеристики: ppgin/s, ppgout/s, pgfree/s – соответственно число загруженных, выгруженных и освобожденных страниц в секунду; pgscan/s – частота сканирования страниц для возможного их освобождения; swpot/s, swpin/s – соответственно число процессов в секунду выгруженных и загруженных в своп; swp-sz — длина очереди процессов, выгруженных в своп. Характеристики ppgin/s, ppgout/s, pgfree/s не говорят о качественном состоянии системы, по их изменению можно судить только об уровне активности. Фактически, высокие значения этих характеристик могут говорить об активном вводе-выводе, который реализован через страничную подкачку. В основном, нехватку или достаточность памяти можно определить, основываясь на значениях частоты сканирования (pgscan/s) и активности свопинга (swpot/s, swpin/s, swp-sz). При достаточном количестве памяти в ОС Solaris 8 все эти параметры всегда должны иметь нулевые значения. В ОС Solaris 7 реализован несколько другой механизм сканирования страниц, поэтому значение pgscan/s может быть ненулевым (но не превышать 250). Кроме того, необходимо добавить ряд важных параметров, показывающих производительность буферного КЭШа, которые тоже должны удовлетворять определенным требованиям: %rcache (частота попадания пользовательских запросов на чтение в КЭШ), %wcache (частота попадания пользовательских запросов на запись в КЭШ). Сказанное позволяет сделать предварительные выводы о том, что для нормальной работы подсистемы памяти необходимо выполнение следующих условий: *pgscan / s* ~ 0, *swp -sz* ~ 0 , *%rchache* ≥ 90%, *%wcache* ≥ 60%

Система ввода–вывода.

Система ввода-вывода, прежде всего, отождествляется с активностью блочных устройств: дисками, массивами, лентами и.т.д. Частично к ней можно отнести параметры, характеризующие состояние файловой системы и КЭШа.

Каждому блочному устройству соответствует ряд параметров, отражающих его работу: r+w/s — число операций чтения или записи, выполняемых в секунду; %busy — доля времени в процентах, когда устройство было чем-то занято; avwait, avserv — соответственно среднее время ожидания и выполнения запроса, затраченное устройством, в миллисекундах; %ufsipf — процент времени, на которое из списка свободной памяти брался индексный дескриптор с ассоциированными повторно используемыми страницами.

Отдельно необходимо описать параметр %ufsipf, который можно отнести также и к подсистеме памяти. Он отображает весьма важную информацию, которая показывает, насколько часто ядру приходилось повторно обращаться к неактивному индексному дескриптору, у которого были ассоциированные с ним страницу в буферном КЭШе. Ненулевое значение параметра, как правило, является признаком того, что таблица inode слишком мала, и поэтому полезные данные из буферного КЭШа приходится выбрасывать.

Из перечисленных характеристик только число операций чтения/записи (r+w/s) сложно интерпретируемо и отражает лишь активность устройства подсистемы, остальные параметры должны удовлетворять следующим требованиям: *%busy* <=3%, *avwait avserv* <=50*ms* , *%ufsipf* ~ 0.

Интегральная оценка состояния системы.

Для оценки производительности вычислительной системы при ее эксплуатации целесообразно пользоваться методом сводных показателей, суть которого состоит в “свертке” многих оценок сложного объекта в единую, интегрированную оценку. При построении сводного показателя учитываются как оценки отдельных характеристик объектов, так и информация об относительной весомости отдельных характеристик, то есть степени влияния оценок по скалярным критериям на сводную оценку в целом.

Сформируем вектор исходных характеристик, достаточных для оценивания производительности:

$$X = \langle runq\text{-}sz, \%wio, pgscan/s, swp\text{-}sz, \%ufsipf, \%busy, avwait, avserv, \%rcache, \%wcache \rangle$$

Сгруппировав все требования, предъявляемые к характеристикам, получим следующую систему оценочных критериев, разделенных на классы соответствия подсистемам:

$$\begin{cases} runq - sz \le 4 \\ \%wio < 30\% \end{cases} \quad \begin{cases} pgscan/s \sim 0 \\ swp - sz \sim 0 \\ \%rchache > 90\% \\ \%wcache > 60\% \end{cases} \quad \begin{cases} \%busy \le 3 \\ avwait + avserv \le 50ms \\ \%ufsipf \sim 0 \end{cases}$$

В качестве примера, рассмотрим применение критериев для четырехпроцессорной системы под управлением ОС SOLARIS 8, предназначенной для обработки запросов к базе данных (активный узел кластера высокой готовности), работающей в крупном банке и обслуживающей сотни клиентов. Сбор статистических данных о заданных характеристиках проводился в течение одного рабочего дня с периодичностью 20 секунд средствами анализатора системной активности SAR. Далее, исходя из предварительной оценки измерений, выберем два временных интервала: интервал, в течение которого система работала устойчиво, и интервал пиковой активности с нестабильными характеристиками рабочей нагрузки. Для каждого были рассчитаны текущие значения интегрального показателя производительности $I(t)$, представленные в виде графиков на рисунке 1. Для расчета $I(t)$ использовалась Оболочка Системы Поддержки Принятия Решений (ОСППР) АСПИД-3W, предназначенная для всестороннего оценивания сложных объектов в условиях неопределенности.

На первом этапе весовые коэффициенты характеристик считались равными. Из графиков (рис. 1) на временном интервале с пиковой нагрузкой (15:05-15:25) хорошо виден провал, соответствующий снижению производительности (пик нагрузки). В реальности, этот временной промежуток соответствует увеличению и изменению типа нагрузки (послеобеденная активность пользователей системы). На временном интервале с равномерной нагрузкой (14:15-14:35) во время обеденного перерыва, напротив, наблюдается достаточно стабильное поведение кривой с некоторыми пиками в сторону повышения интегрального показателя производительности.

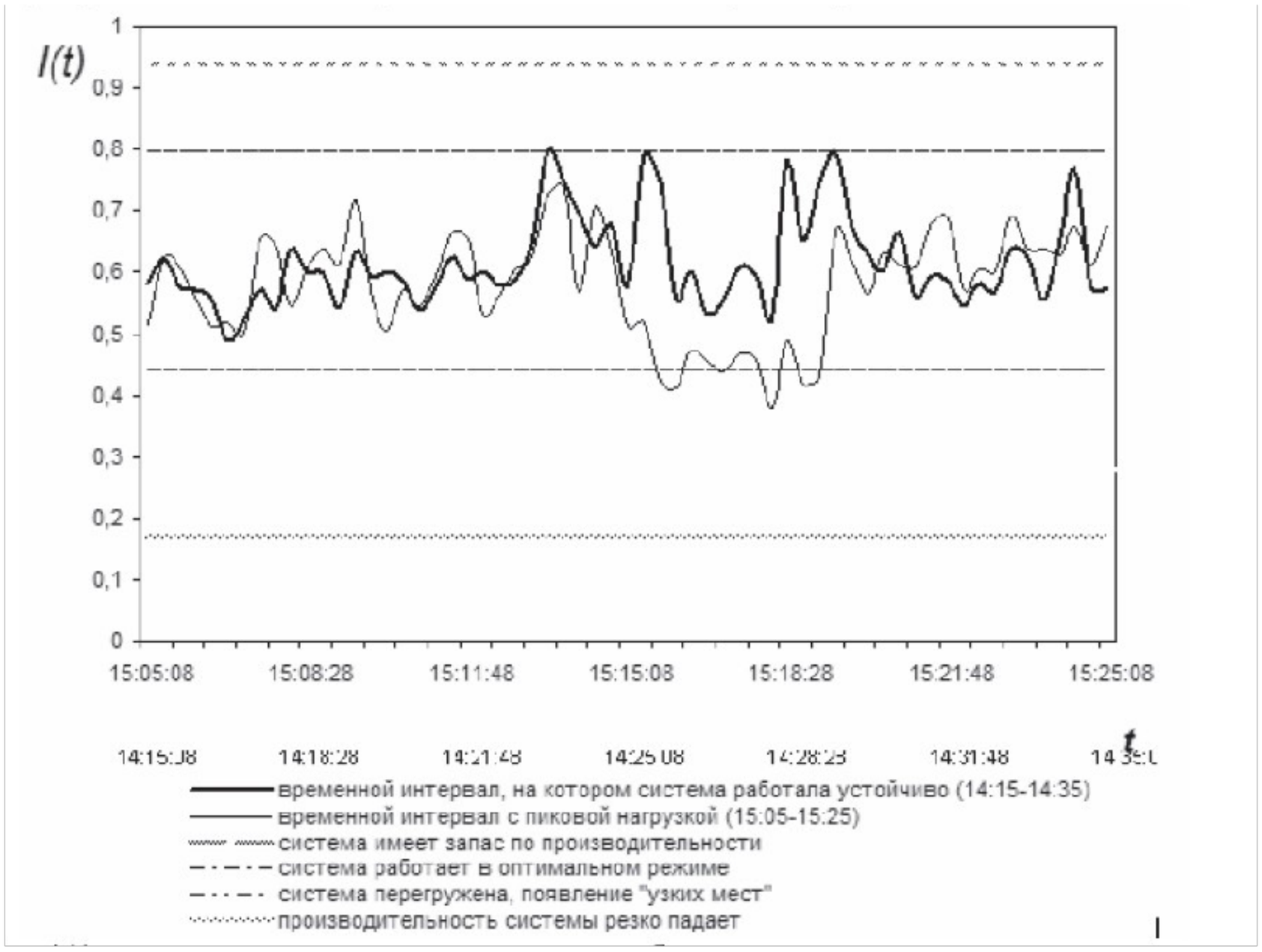


Рисунок 1 - Интегральная оценка производительности работы системы для различных временных интервалов.

На следующем этапе для каждого из временных интервалов были проведены расчеты $I(t)$ с различными порядковыми отношениями между весовыми коэффициентами характеристик. При этом учитывались классы характеристик, то есть их принадлежность одной из трех базовых подсистем. При оценке системы ввода-вывода “веса” характеристик этого класса считались более значимыми, чем ”веса” характеристик других подсистем. Такой же подход применялся и в отношении процессорной подсистемы и подсистемы памяти. Векторы весовых коэффициентов, соответствующие подсистемам ввода-вывода, процессорной и памяти соответственно, имеют вид:

$$W_{I/O} = (w_{\%busy}, w_{avserv+avwait}, w_{\%ufsipf});$$

$$W_{CPU} = (w_{runq-sz}, w_{\%wio});$$

$$W_{MEMORY} = (w_{pgscan/s}, w_{swp-sz}, w_{\%rcache}, w_{\%wcache});$$

Результаты расчетов $I(t)$ производительности подсистем ввода-вывода $W_{I/O} \text{ аА } W_{CPU}; W_{I/O} \text{ аА } W_{MEMORY}$, процессора $W_{CPU} \text{ аА } W_{I/O}; W_{CPU} \text{ аА } W_{MEMORY}$ и памяти $W_{MEMORY} \text{ аА } W_{CPU}; W_{MEMORY} \text{ аА } W_{I/O}$ для каждого из временных интервалов представлены в виде графиков на рисунках. 2 и 3.

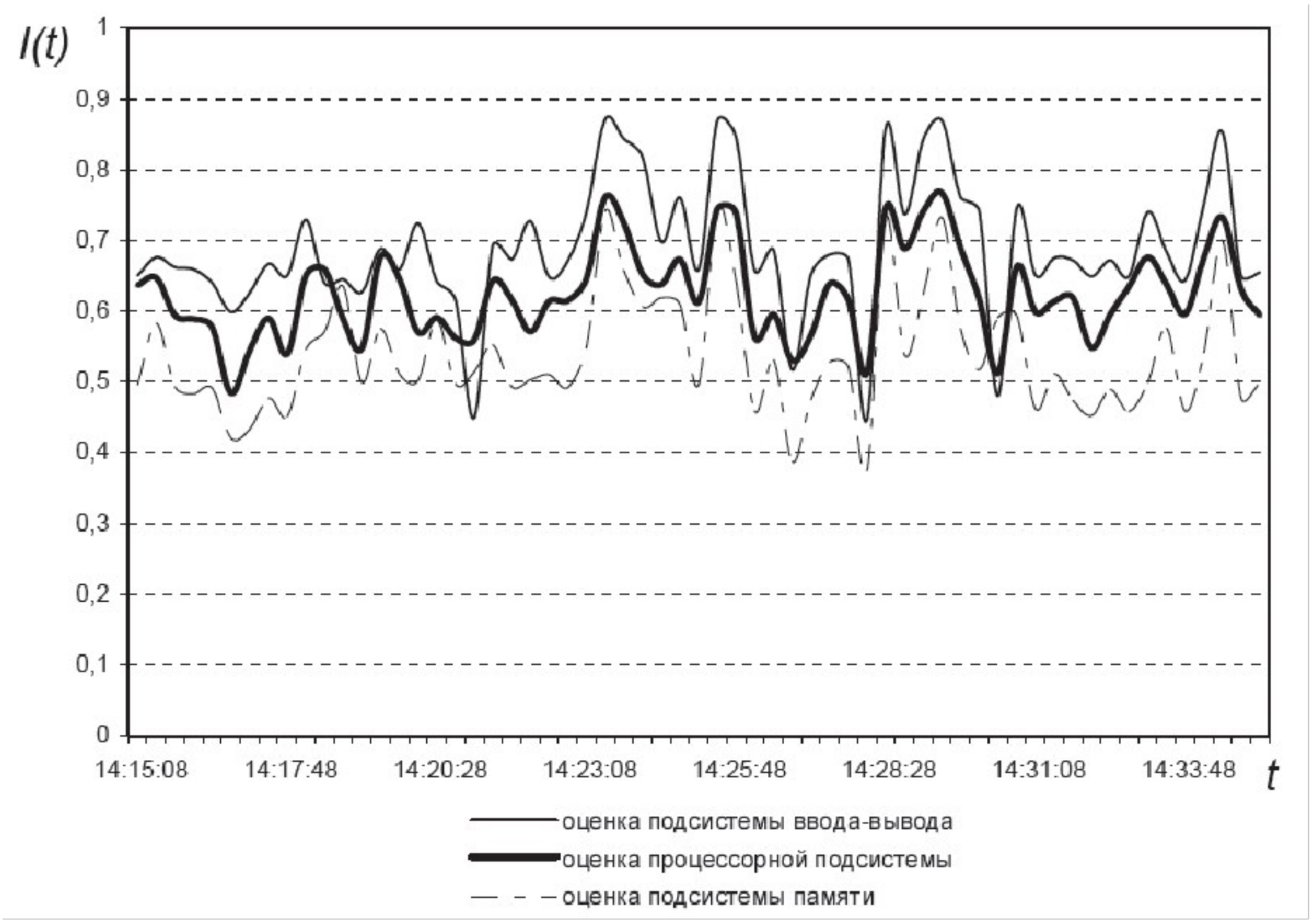


Рисунок 2 - Интегральная оценка производительности подсистем на временном интервале, в течение которого нагрузка была равномерной.

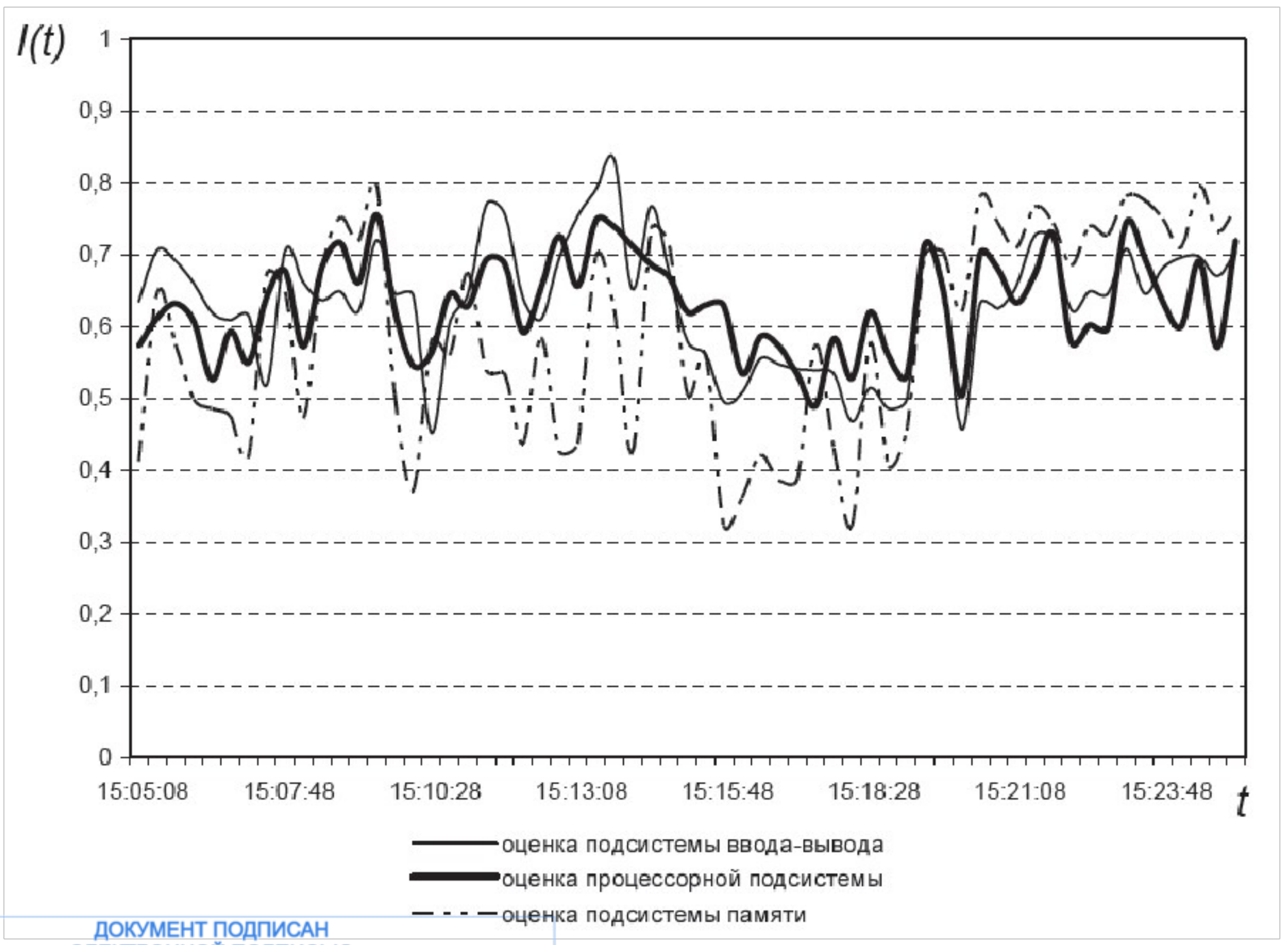


Рисунок 3 - Интегральная оценка производительности подсистем на временном интервале с пиковой и неравномерной нагрузкой.

Степень загрузки системы в каждый момент времени может быть определена путем соотношения между текущим значением показателя $I(t)$ и значениями i I типовых режимов работы системы. Расчет значений $_i I$ для четырех режимов работы (слабая, нормальная, сильная загрузка и перегрузка системы) был проведен методом свертки с использованием экспертных оценок значений характеристик и дал следующие результаты: $_1 I$ ⁖ 0,937, $_2 I$ ⁖ 0,799, $_3 I$ ⁖ 0, 442, $_4 I$ ⁖ 0,171.

Таким образом, можно утверждать, что система слабо загружена и имеет запас по производительности при $_{2\,1} I(t) \in [I; I]$, имеет оптимальную загрузку при $_{3\,2} I(t) \in [I; I]$ и перегружена при $_{4\,3} I(t) \in [I; I]$. При $_1 I(t)$ ▲ I И $_4 I(t)$ ■ I можно говорить о неоправданной недогрузке (или отсутствии прикладных задач) и критической перегруженности. Так как I (t) определяется рядом не зависящих друг от друга и часто неучтенных факторов, изменяющихся во времени, то можно говорить, что I (t) носит псевдослучайный характер. Это позволяет оперировать такими характеристиками, как математическое ожидание $_0 I(T)$ и дисперсия $d(T)$ на конечном временном интервале T. Для заданного интервала значение $_0 I(T)$ характеризует эффективность, а $d(T)$ — стабильность (устойчивость) работы системы. Отсюда можно сделать важный вывод: для $I(t)$ увеличение $_0 I(T)$ в интервале $_{4\,2} [I; I]$ и уменьшение $d(T)$ свидетельствует об увеличении эффективности и стабильности.

Дополнительную информацию о поведении системы дают графики оценки производительности подсистем (см. рис. 2, 3). На рисунке 2. кривые $I(t)$ всех подсистем подобны (изменяются одинаково), что говорит о равномерной их загрузке.

На рисунке 3, напротив, кривые расходятся, в некоторых точках наблюдается увеличение производительности одной подсистемы в ущерб другой, что указывает на появление ”узких мест, характерное при $_{4\,3} I(t) \in [I; I]$ или $_4 I(t)$ ■ I.

Таким образом, графики $I(t)$, полученные для подсистем, позволяют сделать выводы о доминировании загрузки той или иной подсистемы, то есть выявить причину наблюдаемой нестабильности режима работы системы в целом.

Содержание отчета и его форма

Подготовьте отчет, в котором подтвердите эффективность предлагаемого метода оценки производительности системы по интегральному показателю $I(t)$.

Отчет по лабораторной работе должен содержать:

- название работы;
- цель лабораторной работы;
- формулировку задания и технологию его выполнения;
- ответы на контрольные вопросы.

Контрольные вопросы:

1. Характеристики поведения системы направленные на повышение производительности.
2. Применение метода при рассмотрении более сложных систем, таких как кластеры высокой готовности и распределенные сети GRID.

Защита лабораторной работы

По результатам отчета, представленного в письменной форме, проводится собеседование, которое имеет контролирующую и учебную функции.

Лабораторная работа 2.

Параллелизм как основа высокопроизводительных вычислительных систем.

Цель и содержание: Определить цели распараллеливания. Проанализировать способы использования параллелизма. Рассмотреть выполнение параллельных алгоритмов на ЭВМ различных типов.

Организационная форма занятий: лабораторное занятие.

Вопросы для обсуждения на лабораторном занятии: Типы параллелизма. Классификация Флинна. Закон Амдала. Конвейерность и параллельность.

Теоретическое обоснование

Развитие компьютерных систем происходило и происходит под девизом «Скорость и быстрота вычислений». Если быстродействие первой вычислительной машины ENIAC составляло всего несколько тысяч операций в секунду, то самый быстрый на данный момент времени суперкомпьютер RoadRunner может выполнять уже квадриллионы (1015) команд.

Темп развития вычислительной техники просто впечатляет – увеличение скорости вычислений в триллионы (1012) раз не многим более чем за 60 лет. Для лучшего понимания необычности столь стремительного развития средств ВТ часто приводят яркие аналогии – например, что если бы автомобильная промышленность развивалась с такой же динамикой, то сейчас бы автомобили весили бы порядка 200 грамм и тратили бы несколько литров бензина на миллионы километров!

История развития вычислительной техники представляет увлекательное описание замечательных научно-технических решений, радости побед и горечи поражений.

Проблема создания высокопроизводительных вычислительных систем относится к числу наиболее сложных научно-технических задач современности и ее разрешение возможно только при всемерной концентрации усилий многих талантливых ученых и конструкторов, предполагает использование всех последних достижений науки и техники и требует значительных финансовых инвестиций. Важно отметить при этом, что при общем росте скорости вычислений в 1012 раз, быстродействие самих технических средств вычислений увеличилось всего в несколько миллионов раз. И дополнительный эффект достигнут за счет введения параллелизма буквально на всех стадиях и этапах вычислений.

Типы параллелизма.

- Параллелизм на уровне битов (bitlevel parallelism)

8-, 16, 32-, 64-, 128-разрядные процессоры;

- Параллелизм на уровне инструкций (Instruction-level parallelism)

Конвейеризация

Суперскалярность

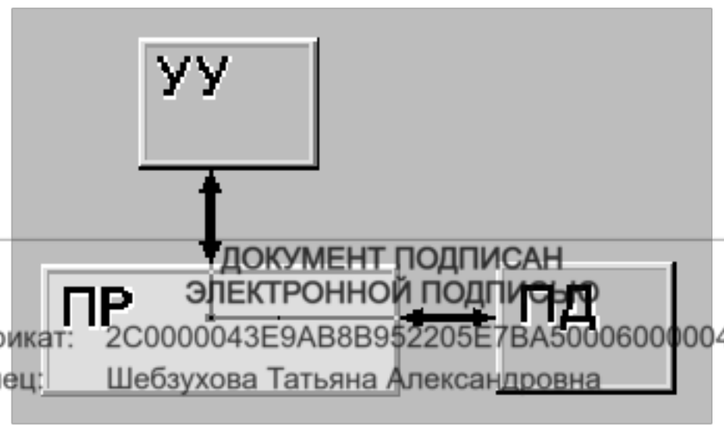
Явный параллелизм команд (VLIW и EPIC)

- Параллелизм задач (task parallelism)
- Параллелизм данных (data parallelism)

Классификация Флинна.

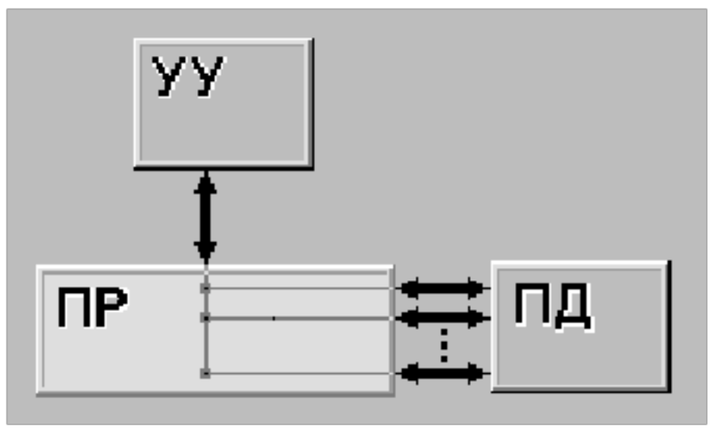
Самой ранней и наиболее известной является классификация архитектур вычислительных систем, предложенная в 1966 году М.Флинном.

Классификация базируется на понятии потока, под которым понимается последовательность элементов, команд или данных, обрабатываемая процессором. На основе числа потоков команд и потоков данных Флинн выделяет четыре класса архитектур: SISD,MISD,SIMD,MIMD.

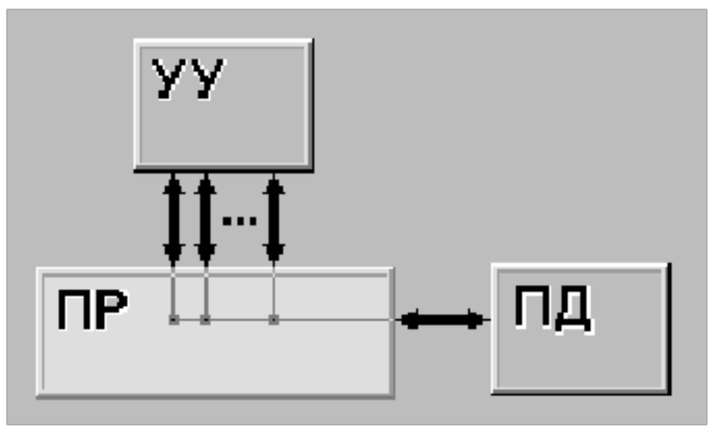


SISD (single instruction stream / single data stream) - одиночный поток команд и одиночный поток данных. К этому классу относятся, прежде всего, классические последовательные машины, или иначе, машины фон-неймановского типа, например, PDP-11 или VAX 11/780. В таких машинах есть только один поток команд, все команды обрабатываются последовательно друг за другом и каждая команда

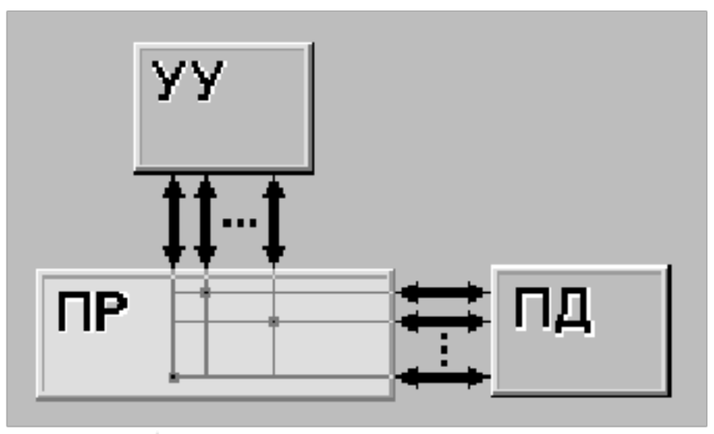
инициирует одну операцию с одним потоком данных. Не имеет значения тот факт, что для увеличения скорости обработки команд и скорости выполнения арифметических операций может применяться конвейерная обработка - как машина CDC 6600 со скалярными функциональными устройствами, так и CDC 7600 с конвейерными попадают в этот класс.



SIMD (single instruction stream / multiple data stream) - одиночный поток команд и множественный поток данных. В архитектурах подобного рода сохраняется один поток команд, включающий, в отличие от предыдущего класса, векторные команды. Это позволяет выполнять одну арифметическую операцию сразу над многими данными - элементами вектора. Способ выполнения векторных операций не оговаривается, поэтому обработка элементов вектора может производится либо процессорной матрицей, как в ILLIAC IV, либо с помощью конвейера, как, например, в машине CRAY-1.



MISD (multiple instruction stream / single data stream) - множественный поток команд и одиночный поток данных. Определение подразумевает наличие в архитектуре многих процессоров, обрабатывающих один и тот же поток данных. Однако ни Флинн, ни другие специалисты в области архитектуры компьютеров до сих пор не смогли представить убедительный пример реально существующей вычислительной системы, построенной на данном принципе. Ряд исследователей [3,4,5] относят конвейерные машины к данному классу, однако это не нашло окончательного признания в научном сообществе. Будем считать, что пока данный класс пуст.



MIMD (multiple instruction stream / multiple data stream) - множественный поток команд и множественный поток данных. Этот класс предполагает, что в вычислительной системе есть несколько устройств обработки команд, объединенных в единый комплекс и работающих каждое со своим потоком команд и данных.

Итак, что же собой представляет каждый класс? В SISD, как уже говорилось, входят однопроцессорные последовательные компьютеры типа VAX 11/780. Однако, многими критиками подмечено, что в этот класс можно включить и векторно-конвейерные машины, если рассматривать вектор как одно неделимое данное для соответствующей команды. В таком случае в этот класс попадут и такие системы, как CRAY-1, CYBER 205, машины семейства FACOM VP и многие другие.

Бесспорными представителями класса SIMD считаются матрицы процессоров: ILLIAC IV, ICL DAP, Goodyear Aerospace MPP, Connection Machine 1 и т.п. В таких

системах единое управляющее устройство контролирует множество процессорных элементов. Каждый процессорный элемент получает от устройства управления в каждый фиксированный момент времени одинаковую команду и выполняет ее над своими локальными данными. Для классических процессорных матриц никаких вопросов не возникает, однако в этот же класс можно включить и векторно-конвейерные машины, например, CRAY-1. В этом случае каждый элемент вектора надо рассматривать как отдельный элемент потока данных.

Класс MIMD чрезвычайно широк, поскольку включает в себя всевозможные мультипроцессорные системы: Cm*, C.mmp, CRAY Y-MP, Denelcor HEP,BBN Butterfly, Intel Paragon, CRAY T3D и многие другие. Интересно то, что если конвейерную обработку рассматривать как выполнение множества команд (операций ступеней конвейера) не над одиночным векторным потоком данных, а над множественным скалярным потоком, то все рассмотренные выше векторно-конвейерные компьютеры можно расположить и в данном классе.

Предложенная схема классификации вплоть до настоящего времени является самой применяемой при начальной характеристике того или иного компьютера. Если говорится, что компьютер принадлежит классу SIMD или MIMD, то сразу становится понятным базовый принцип его работы, и в некоторых случаях этого бывает достаточно. Однако видны и явные недостатки. В частности, некоторые заслуживающие внимания архитектуры, например dataflow и векторно--конвейерные машины, четко не вписываются в данную классификацию. Другой недостаток - это чрезмерная заполненность класса MIMD. Необходимо средство, более избирательно систематизирующее архитектуры, которые по Флинну попадают в один класс, но совершенно различны по числу процессоров, природе и топологии связи между ними, по способу организации памяти и, конечно же, по технологии программирования.

Наличие пустого класса (MISD) не стоит считать недостатком схемы. Такие классы, по мнению некоторых исследователей в области классификации архитектур [6,7], могут стать чрезвычайно полезными для разработки принципиально новых концепций в теории и практике построения вычислительных систем.

Закон Амдала.

Параллельность, это хороший способ обойти ограничение роста тактовой частоты, но у него есть собственные ограничения. Прежде всего, это закон Амдала, в соответствии с которым ускорение процесса вычислений при использовании p процессоров ограничивается величиной, где f есть доля последовательных вычислений в применяемом алгоритме обработки даннях

Закон Амдала. Достижению максимального ускорения может препятствовать существование в выполняемых вычислениях последовательных расчетов, которые не могут быть распараллелены. Пусть f есть *доля последовательных вычислений* в применяемом алгоритме обработки данных, тогда в соответствии с *законом Амдала ускорение* процесса вычислений при использовании p процессоров ограничивается величиной

$$S_p \le \frac{1}{f + (1-f)/p} \le S^* = \frac{1}{f}.$$

Ускорение кода зависит от числа процессоров и параллельности кода согласно формуле. Действительно, с помощью параллельного выполнения мы можем ускорить время выполнения только параллельного кода.

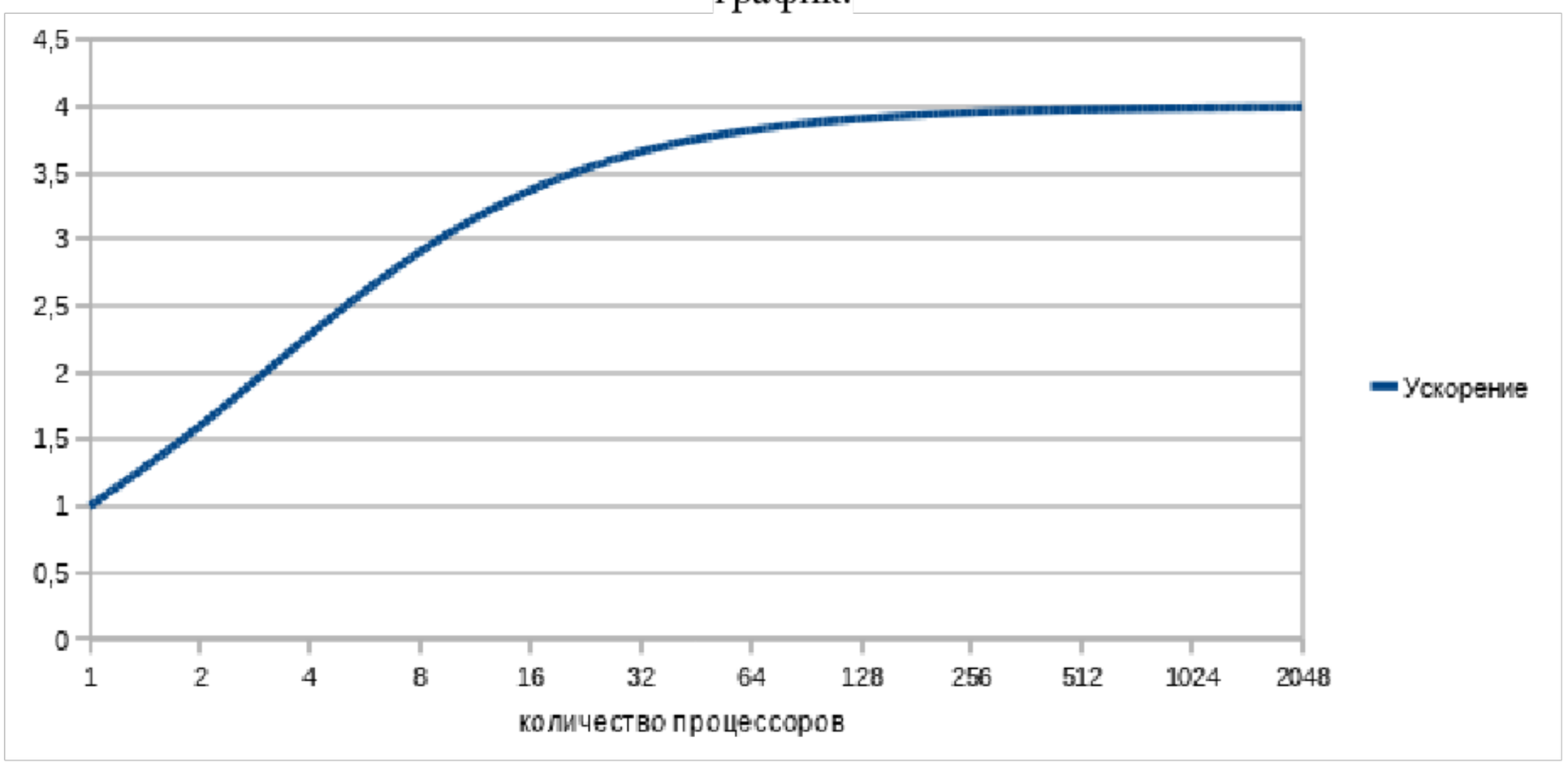
$$Ускорение = \frac{1}{Время\,выполнения\,последовательного\,кода + \frac{время\,выполнения\,параллельного\,кода}{количество\,процессоров}}$$

В любой же программе кроме параллельного кода есть и последовательные участки и ускорить их с помощью увеличения количества процессоров не получится, над ними будет работать только один процессор. *Например, если выполнение последовательного кода занимает всего 25% от времени выполнения всей программы, то ускорить эту программу более чем в 4 раза не получится никак.*

Построим график зависимости ускорения нашей программы от количества параллельно работающих вычислителей-процессоров.

В реальном мире затраты обеспечение параллельности никогда не равны нулю и потому при добавлении все новых и новых процессоров производительность, начиная с некоторого момента, начнет падать. Но как используется мощь современных многоядерных суперкомпьютеров? Во многих алгоритмах время исполнения параллельного кода сильно зависит от количества обрабатываемых данных, а время исполнения последовательного кода — нет. Чем больше данных требуется обработать, тем больше выигрыш от параллельности их обработки. Потому «загоняя» на суперкомпьютере большие объемы данных получаем хорошее ускорение. Например, перемножая матрицы 3*3 на суперкомпьютере мы вряд ли заметим разницу с обычным однопроцессорным вариантом, а вот умножение матриц, размером 1000*1000 уже будет вполне оправдано на многоядерной машине.

Подставив в формулу 1/4 последовательного кода и 3/4 параллельного, получим график.



Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер с выходом в сеть интернет.

Указания по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения, ремонт персонального компьютера. Соблюдать правила технической безопасности при работе с электрооборудованием.

Методика и порядок выполнения работы

Рассмотрим особенности выполнения задачи умножения матриц на конвейерных ЭВМ, процессорных матрицах и многопроцессорных ЭВМ типа МКМД.

Конвейерные ЭВМ. В качестве образца конвейерной ЭВМ для экспериментов над параллельными алгоритмами возьмем упрощенный вариант ЭВМ CRAY-1.

Напомним ее основные характеристики: такт синхронизации 12,5 нс; память состоит из 16 блоков, обращение к каждому блоку занимает 4 такта; имеется ряд функциональных устройств (векторные АЛУ для чисел с плавающей запятой и длиной конвейера 6 и 7 тактов; временем подключения этих устройств к векторным регистрам для простоты будем пренебрегать) и восемь векторных регистров, каждый по 64 слова.

Приведем примеры стандартного размещения вектора длиной 16 элементов и матрицы размерностью16X16 (рис. 1).

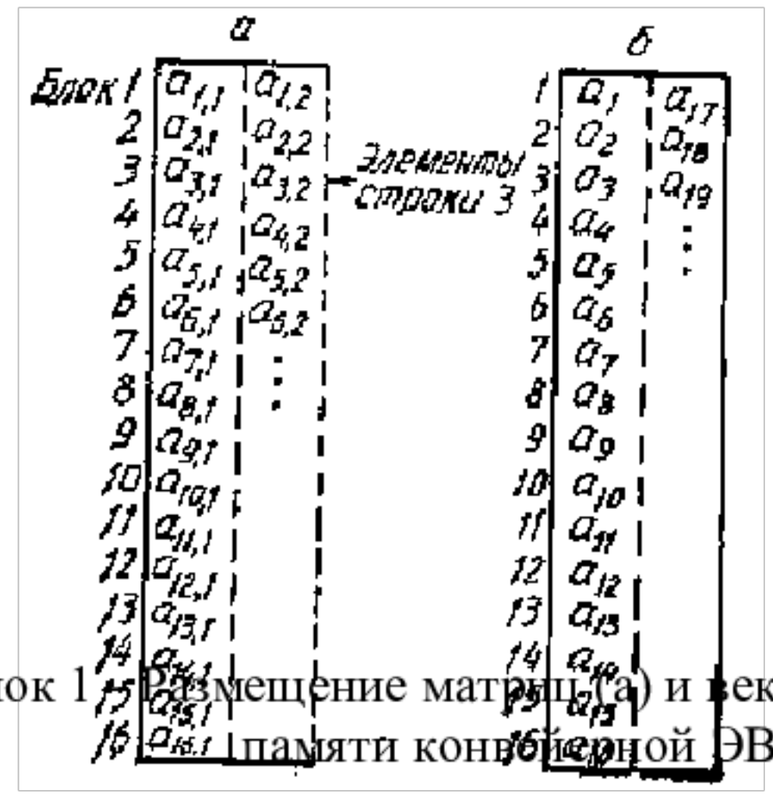


Рисунок 1 Размещение матриц (а) и векторов (б) в многоблочной памяти конвейерной ЭВМ

Стандартным для расположения матриц является размещение по столбцам. Если размер матрицы кратен числу блоков памяти, то стандартное размещение матрицы приводит к тому, что элементы столбцов расположены в разных блоках памяти и их можно считывать по очереди с максимальной скоростью: один элемент за один такт. Но каждая строка оказывается полностью размещенной в одном блоке памяти, поэтому последовательные элементы строк считываются с минимальной скоростью: один элемент за четыре такта (см. рис. 1, а).

Определенные проблемы существуют и в размещении векторов (см. рис. 1, *б)*. Последовательные элементы вектора a_1, a_2, a_3, ... и a_1, a_5, a_9 ... с кратностью 4 считываются с максимальной скоростью, а элементы a_1, a_9, a_{17}, ... с кратностью 8 — с половинной скоростью, так как за время цикла памяти (четыре такта) в каждый блок поступает два запроса; элементы a_1, a_{17} ... с кратностью 16, расположенные в одном блоке, считываются со скоростью в четыре раза ниже максимальной. Эти ситуации могут встречаться при использовании методов каскадных сумм или циклической редукции.

В конвейерной ЭВМ выделяют три уровня производительности:

1) скалярная производительность достигается только при использовании скалярных команд и регистров;

2) векторная производительность достигается при использовании программ с векторными командами, скорость выполнения которых ограничивается скоростью обмена с памятью;

3) сверхвекторная производительность достигается при использовании программы с векторными командами, скорость выполнения которых ограничивается готовностью векторных регистров или функциональных устройств.

В CRAY-1 скорости выполнения программ векторных команд заметно отличаются и равны 12 и 153 Мфлоп/с соответственно.

При умножении матриц нас будут интересовать только векторная и сверхвекторная производительности, причем основными средствами ускорения вычислений являются:

- обеспечение максимальной скорости обращения к памяти;

- уменьшение объема обращений к памяти и соответственно увеличение объема работы с векторными регистрами;
- увеличение длины векторов, чтобы уменьшить влияние времени «разгона» (определяется длиной конвейера функциональных устройств);
- максимальное использование зацепления операций.

Для простоты дальнейшего анализа рассмотрим матрицы размерностью 64Х64. При использовании метода внутреннего произведения необходимо выполнить следующий алгоритм для вычисления одного элемента матрицы С (рис. 2, *а).*

Из памяти в векторный регистр выбирается строка А (I, *) матрицы *А.* При стандартном размещении матриц все элементы строки оказываются в одном блоке памяти, поэтому на выборку строки затрачивается время (в тактах) $t_i=l_1+4n$, где l_1 — время «разгона» (длина конвейера) памяти (для CRAY-1 $l_i=11$). Затем производится поэлементное умножение векторов в АЛУ (*). Суммирование получаемых произведений в АЛУ (+) выполняется в зацеплении с предыдущей операцией. Результат *С(I, J)* заносится в один из скалярных регистров S. Это потребует времени $t_2=l_2+l_3+n$, где $l_2=7$ и $l_3=6$ — время «разгона» конвейерных АЛУ и умножения соответственно.

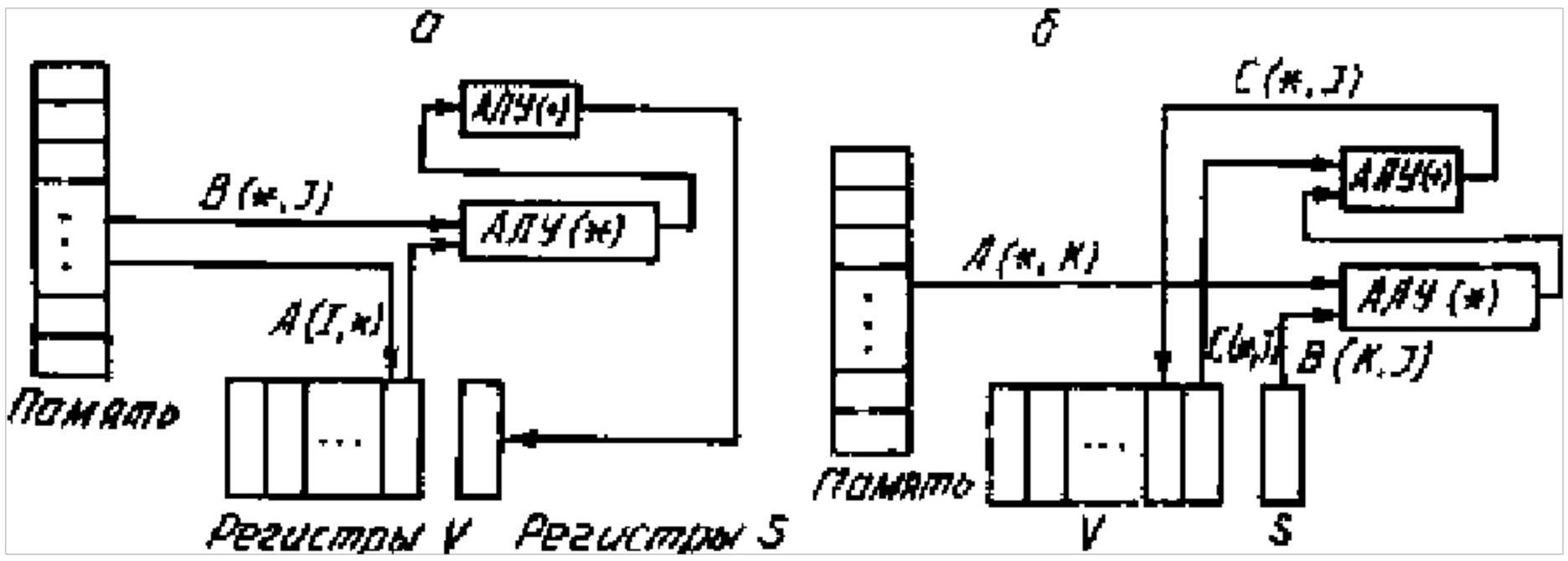


Рисунок 2 - Схема соединений конвейерной ЭВМ для выполнения операций умножения матриц: *а* — метод внутреннего произведения; б — метод среднего произведения

Тогда время вычисления всей матрицы методом внутреннего произведения

$$t_{вн}=(t_1+t_2)n^2=(l_1+l_2+l_3+5n)n^2=5{,}3n^3,$$

$$l_1+l_2+l_3=22\approx 0{,}3n.$$

если предположить, что при использовании описанного метода большое замедление вызывает выборка строки матрицы *А.* Если применим специальное (нестандартное) размещение матрицы *А* таким образом, чтобы смежные элементы строки были расположены в соседних блоках памяти, то обеспечим считывание каждого элемента строки за один такт, т. е. с максимальной скоростью. Тогда

$$t_{вн}=2{,}3n^3, \qquad (2)$$

Но при этом требуется изменение стандартных трансляторов.

Более перспективными являются методы среднего произведения. Рассмотрим реализацию такой программы. Соответствующая схема для вычисления одного столбца

матрицы приведена на рис. 2, б. При использовании метода среднего произведения столбец С(*, /) постоянно располагается в регистрах V, а из блоков памяти выбираются только столбцы А (*, К). Поскольку при стандартном расположении матриц элементы столбца располагаются в различных блоках памяти, то элементы А(*, К} будут выбираться с максимальной скоростью. При вычислении элементов столбца матрицы С методом среднего произведения совмещают три операции: считывание А(*, К} из блоков памяти; умножение этого вектора на константу В (К, J), получаемую из скалярного регистра 5; сложение столбца С(*, /) с результирующим вектором из АЛУ (*). Поэтому среднее время вычисления всей матрицы С:

$$t_{ср} = n^2(l_1 + l_2 + l_3 + n) = 1{,}3n^3. \quad (3)$$

При определении времени умножения матриц опущены различного рода вспомогательные операции. В частности, в последнем случае не учтены операции считывания из памяти и запись в память столбцов матрицы С. Таким образом, полученные оценки времени умножения следует считать приблизительными.

Оценим быстродействие, получаемое по формуле (3). При вычислении произведения матриц выполняется n^3 операций умножения и n^3 операций сложения. Для конвейерных ЭВМ длительности тактов сложения и умножения равны, поэтому будем считать, что выполняется $2n^3$ операций, и, следовательно, на одну арифметико-логическую операцию затрачивается

$$\frac{t_{ср}}{2n^3} = \frac{1{,}3}{2} = 0{,}65 \text{ такта}.$$

Для CRAY-1 при длительности такта 12,5 нс это соответствует быстродействию

$$V = \frac{10^9}{0{,}65 \cdot 12{,}5} = 120 \text{ Мфлоп/с}.$$

Таким образом, метод среднего произведения обеспечивает сверхвекторную производительность и считается, будучи запрограммированным на Ассемблере, наилучшим методом для ЭВМ типа CRAY-1. Его преимущества обеспечиваются большой глубиной зацепления и тем, что один из векторов (столбец матрицы С) постоянно расположен в векторных регистрах.

Метод внутреннего произведения обеспечивает векторную (см. (1)) и промежуточную между векторной и сверхвекторной (см. (2)) производительности.

Метод внешнего произведения не имеет преимуществ перед методом среднего произведения. Более того, возможность работы с векторными регистрами и глубокое зацепление делают метод среднего произведения предпочтительным.

Процессорные матрицы.

При выборе наилучшего алгоритма для ПМ необходимо учитывать следующие структурные отличия процессорных матриц от конвейерных ЭВМ.

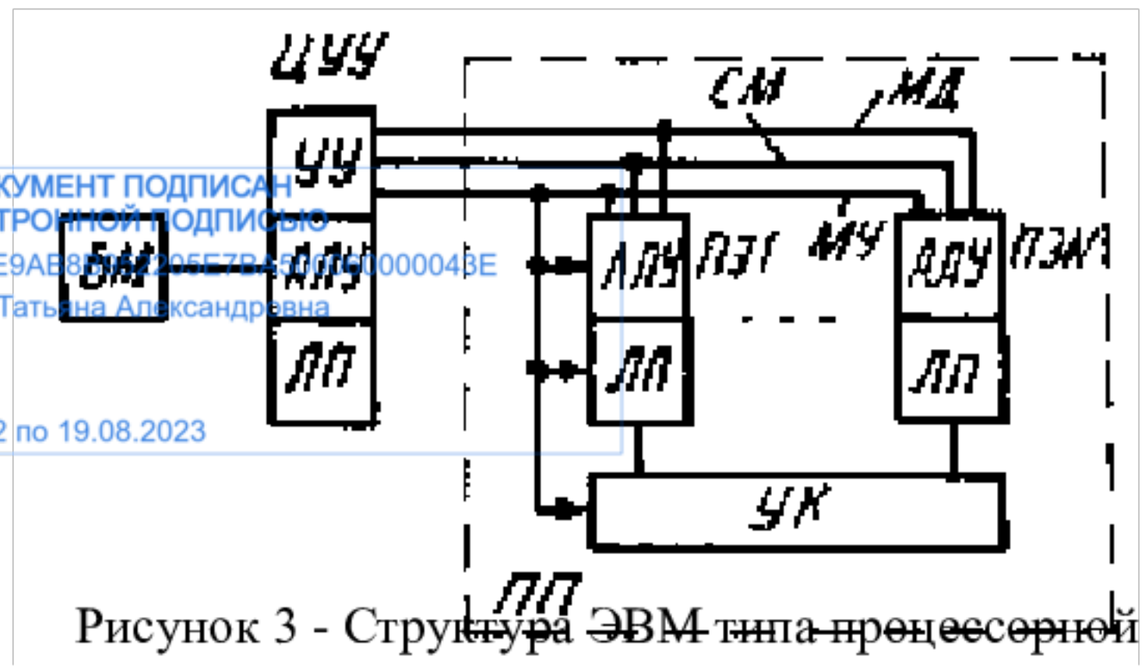


Рисунок 3 - Структура ~~ЭВМ типа процессорной~~ матрицы

1. Быстродействие в ПМ достигается за счет числа процессоров *N,* а не за счет глубины конвейера.

2. Основными факторами, увеличивающими время выполнения алгоритма, является конфликты в памяти и затраты на коммутацию, поэтому размещение данных в памяти является одним из наиболее существенных моментов для процессорных матриц.

3. Для дальнейшего использования в этом параграфе принята структура процессорной матрицы (рис. 3), для которой:

— каждый ПЭ имеет локальную память (ЛП) данных достаточно большого размера (сотни килобайт);

— процессоры обмениваются данными через универсальный коммутатор (УК), который может за время исполнения одной команды коммутации выполнить любую операцию коммутации: сдвиг или обмен по произвольному вектору адресов. Поэтому метод каскадных сумм может быть в такой структуре реализован двояко: либо с помощью сдвигов, либо прямой реализацией каскадного дерева.

4. Программа исполняется в ЦУУ и может выключать любую группу ПЭ на время исполнения одной или нескольких команд, посылаемых через СМ или размещаемых заранее в массиве ПЭ. Процессорные элементы могут также выключаться в зависимости от результата выполнения операций.

5. В системе команд процессорной матрицы имеются команды для выполнения каскадной суммы типа *SUM N, A,* где *N* — число включенных процессоров;

Л — адрес размещения результата суммирования.

Если для конвейерных ЭВМ характерна длительность такта синхронизации 10...15 нс, то для процессорных матриц — 100...150 нс.

Примем для дальнейших расчетов следующее время выполнения отдельных операций в ПЭ: сложение двух чисел — 1 такт; выборка числа из памяти — 1 такт; умножение двух чисел — 2 такта; операция коммутации для N процессоров ($N \le 2^5$) — 1 такт; операция типа SUM — $2log_2N$ тактов.

Далее будет рассмотрена операция умножения матриц для следующих вариантов в предположении, что n — размер стороны матрицы данных; N — общее число процессоров в ПМ:

1) N=n. Этот вариант позволяет рассмотреть некоторые специальные методы размещения данных;

2) $N=n^2$, $N=n^3$. В большинстве случаев данные варианты имеют теоретический интерес, так как при n=100 (это реальное число) требуется 104 или 106 процессоров, что пока еще нельзя осуществить;

3) N<.n. Этот вариант является наиболее интересным с точки зрения практики.

Очевидно, что для любого N следует выбирать методы умножения матриц с наибольшим потенциальным параллелизмом.

Вариант *N=n*.

Для этого варианта следует применять алгоритмы с уровнем параллелизма O(n) и выше. Рассмотрим метод внутренних произведений. В основе метода лежит параллельная выборка строк матрицы Л и столбцов матрицы *B*. Тогда алгоритм вычисления одного элемента *C(I, J)* может быть таким:

1) параллельное считывание из памяти в регистры всех ПЭ 1-й строки матрицы Л — 1 такт;

2) считывание 7-го столбца матрицы *B* в регистры ПЭ — 1 такт;

3) поэлементное перемножение выбранных строки и столбца — 2 такта;

4) суммирование n частных произведений — $2\log_2 n$ тактов.

Следовательно, вычисление всех элементов матрицы С займет время (в тактах)

$$t_{вн} = n^2(4 + \log_2 n).$$

При пользовании методом среднего произведения матрицы *A и C* располагаются горизонтально по столбцам, а матрица *B* — в памяти ЦУУ. Поэтому алгоритм вычисления одного столбца матрицы *C* будет следующим:

1) запись очередного элемента *B (K, f)* во все ПЭ — 1 такт;

2) считывание из памяти в ПЭ *K-го* столбца матрицы Л — 1 такт;

3) умножение Л (* , *K) B (K, J)* — 2 такта;

4) суммирование полученных частных произведений с соответствующими частными суммами элементов /-го столбца матрицы С, который постоянно располагается в регистрах ПЭ,— 1 такт.

Таким образом, одна итерация вычисления C(*, *J)* занимает 5 тактов; полное вычисление столбца C(*, /) займет 5 тактов, а вычисление всей матрицы *C* $t_{ср}=5n^2$ тактов.

Очевидно, что $t_{ср}<t_{вн}$ для всех $n\geq 2$. Преимущество метода средних произведений — в отсутствии операции вычисления каскадной суммы.

Вариант $N=n^3$.

Этот вариант имеет больше теоретическое, чем прикладное, значение. В частности, он может использоваться при умножении матриц размерностью 16X16 на ЭВМ ICL DAP (размерность ПП 64X64). В этом случае все размещение матриц в ПП аналогично размещению этих матриц в описанном выше методе внешнего произведения, но поскольку вместо одной матрицы процессоров будет 16 слоев ПМ, то все n^3 операции умножения могут быть выполнены одновременно, а затем все частные произведения суммируются за $O(\log_2 n)$ тактов. Следовательно, общее время выполнения операции умножения матриц составит $t=O(1+\log_2 n)$. Этот метод требует такого же избыточного резервирования данных, как и метод среднего произведения при $N=n^2$.

Вариант N<n.

Следует отметить, что это наиболее реалистичная ситуация, так как обычно N колеблется от 8...16 до 100, в то время как размер матрицы n может колебаться от 100 до 1000.

Оценим ситуацию, когда n кратно числу процессоров N. Тогда можно использовать метод клеточного умножения матриц. Рассмотрим для простоты умножение матриц AxB=C размерностью 4x4:

$$\left[\begin{array}{cc:cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hdashline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] \times \left[\begin{array}{cc:cc} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \hdashline b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{array}\right] = \left[\begin{array}{cc:cc} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ \hdashline c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{array}\right]. \quad (4)$$

Вычисление любого элемента матрицы С производится по известной формуле

$$C(I, J) = C(I, J) + A(I, K) * B(K, J).$$

В частности, для C_{11} получаем

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41}. \quad (5)$$

Разобьем исходные матрицы на клетки (как показано в штриховыми линиями), которые обозначим a'_{ij}, где i, j — индексы, указывающие положение клеток в соответствии

с обычными обозначениями для матриц. Тогда выражение для матриц A, B, C заменяется следующим выражением для матриц A', B', C':

$$\begin{vmatrix} a'_{11} a'_{12} \\ a_{21} a_{22} \end{vmatrix} \times \begin{vmatrix} b'_{11} b'_{12} \\ b_{21} b_{22} \end{vmatrix} = \begin{vmatrix} c'_{11} c'_{12} \\ c_{21} c_{22} \end{vmatrix}. \quad (6)$$

Для этого выражения по обычным правилам выполнения матричных операций имеем:

$$c'_{11} = a'_{11} b'_{11} + a'_{12} b'_{21}, \quad c'_{12} = a'_{11} b'_{12} + a'_{12} b'_{22},$$
$$c'_{21} = a'_{21} b'_{11} + a'_{22} b'_{21}, \quad c'_{22} = a'_{21} b'_{12} + a'_{22} b'_{22}. \quad (7)$$

Эти выражения позволяют вычислить все элементы исходной матрицы C на основе матриц меньшего размера, в частности для Сц можно записать:

$$c'_{11} = \begin{vmatrix} c_{11} c_{12} \\ c_{21} c_{22} \end{vmatrix} = \begin{vmatrix} a_{11} a_{12} \\ a_{21} a_{22} \end{vmatrix} \times \begin{vmatrix} b_{11} b_{12} \\ b_{21} b_{22} \end{vmatrix} + \begin{vmatrix} a_{13} a_{14} \\ a_{23} a_{24} \end{vmatrix} \times \begin{vmatrix} b_{31} b_{32} \\ b_{41} b_{42} \end{vmatrix}, \quad (8)$$

откуда, например, по обычным правилам для умножения матриц следует:

$$c_{11} = a_{11} b_{11} + a_{12} b_{21} + a_{13} b_{31} + a_{14} b_{41}. \quad (9)$$

Выражения (5) и (9) идентичны, что подтверждает правильность приведенных выкладок.

Таким образом, при использовании метода клеток для умножения матриц необходимо в общем случае выполнить перечисленные ниже действия.

1. Исходные матрицы большого размера, когда сторона матрицы n >> N, разбиваются на клетки размерностью NxN. Число таких клеток в каждой матрице будет $(n/N)^2$. Эти клетки размещаются в памяти и могут в дальнейшем обрабатываться с параллелизмом N.

2. Для полученного разбиения составляются системы выражений типа (7). Каждое выражение будет содержать *nfN* слагаемых. Алгоритмы вычислений по системе выражений типа (7) закладываются в программу пользователя или автоматически генерируются транслятором с языка ЯВУ.

3. В процессе выполнения программы в соответствии с выражениями типа (8) из памяти выбираются и обрабатываются с параллелизмом N клетки, т. е. матрицы значительно меньшего размера, чем исходные, что и обеспечивает эффективность обработки. При этом умножение матриц в выражении (8) может производиться любым подходящим методом, например методом среднего произведения.

Для некоторых методов вычислений метод клеточных матриц не может быть использован, в подобном случае применяется размещение данного вектора в памяти процессорного поля по слоям.

Обобщая полученные результаты, сравним время умножения матриц для конвейерных ЭВМ $t_к$ и матриц процессоров $t_м$. Предположим, что для каждого типа ЭВМ используются лучшие методы (см. (3) и (4) соответственно) и время одного такта равно 10

нс для конвейерной ЭВМ и 100 нс для ПМ. Вопрос о требуемых объемах оборудования при этом не рассматривается. Соотношение времени умножения матриц

$$r = \frac{t_м}{t_к} = \frac{5 \cdot n^2 \cdot 100}{1,3 \cdot n^3 \cdot 10} \approx \frac{39}{n}. \quad (10)$$

Из формулы (10) следует, что при n≥39 процессорные матрицы предпочтительнее.

Необходимо еще раз подчеркнуть, что подобные расчеты носят качественный характер.

Задания:

1. Приведите дополнительные примеры параллельных вычислительных систем.
2. Выполните рассмотрение дополнительных способов классификации компьютерных систем.
3. Изучите и дайте общую характеристику способов выполнения задачи умножения матриц на конвейерных ЭВМ, процессорных матрицах и многопроцессорных ЭВМ типа МКМД.

Содержание отчета и его форма

Подготовьте отчет, в котором опишите выполнение заданий.

Отчет по лабораторной работе должен содержать:

- название работы;
- цель лабораторной работы;
- формулировку задания и технологию его выполнения;
- ответы на контрольные вопросы.

Контрольные вопросы:

1. В чем заключаются основные способы достижения параллелизма?
2. Что положено в основу классификация Флинна?
3. Закон Амдала.
4. Основные способы организации многопроцессорности.
5. Способы организации параллельных вычислений.
6. Современные параллельные архитектуры.
7. Чем вызвана необходимость разработки ускорителей вычислений общего назначения?

Защита лабораторной работы

По результатам отчета, представленного в письменной форме, проводится собеседование, которое имеет контролирующую и учебную функции.

Лабораторная работа 3.

Ознакомление с системой параллельного программирования MPI.

Цель и содержание: Изучение общих понятий MPI. Создание параллельного варианта программы, реализующей подсчет экспоненты. Исследование эффективности работающего варианта программы. Предложение вариантов возможных улучшений.

Организационная форма занятий: лабораторное занятие.

Вопросы для обсуждения на лабораторном занятии: Общие функции MPI. Прием/передача между отдельными прцессами. Коллективные взаимодействия процессов. Синхронизация процессов. Работа с группами процессов.

Теоретическое обоснование

MPI - это библиотека передачи сообщений, собрание функций на C/C++ (или подпрограмм в Фортране), облегчающих коммуникацию (обмен данными и синхронизацию задач) между процессами параллельной программы с распределенной памятью. Акроним MPI установлен для Message Passing Interface (интерфейс передачи сообщений).

Коммуникационная библиотека MPI стала общепризнанным стандартом в параллельном программировании с использованием механизма передачи сообщений.

MPI . Терминология и обозначения.

MPI - message passing interface - библиотека функций, предназначенная для поддержки работы параллельных процессов в терминах передачи сообщений.

Номер процесса - целое неотрицательное число, являющееся уникальным атрибутом каждого процесса.

Атрибуты сообщения - номер процесса-отправителя, номер процесса-получателя и идентификатор сообщения. Для них заведена структура *MPI_Status*, содержащая три поля: *MPI_Source* (номер процесса отправителя), *MPI_Tag* (идентификатор сообщения), *MPI_Error* (код ошибки); могут быть и добавочные поля.

Идентификатор сообщения (msgtag) - атрибут сообщения, являющийся целым неотрицательным числом, лежащим в диапазоне от 0 до 32767. Процессы объединяются в *группы*, могут быть вложенные группы. Внутри группы все процессы перенумерованы. С каждой группой ассоциирован свой*коммуникатор*. Поэтому при осуществлении пересылки необходимо указать идентификатор группы, внутри которой производится эта пересылка. Все процессы содержатся в группе с предопределенным идентификатором *MPI_COMM_WORLD*.

При описании процедур MPI будем пользоваться словом OUT для обозначения "выходных" параметров, т.е. таких параметров, через которые процедура возвращает результаты.

Общие процедуры MPI.

int MPI_Init(int* argc, char* argv)**

MPI_Init - инициализация параллельной части приложения. Реальная инициализация для каждого приложения выполняется не более одного раза, а если MPI уже был инициализирован, то никакие действия не выполняются и происходит немедленный возврат из подпрограммы. Все оставшиеся MPI-процедуры могут быть вызваны только после вызова *MPI_Init*.

Возвращает: в случае успешного выполнения - *MPI_SUCCESS*, иначе - код ошибки. (То же самое возвращают и все остальные функции, рассматриваемые в данном руководстве.)

int MPI_Finalize(void)

MPI_Finalize - завершение параллельной части приложения. Все последующие обращения к любым MPI-процедурам, в том числе к *MPI_Init*, запрещены. К моменту вызова *MPI_Finalize* некоторым процессом все действия, требующие его участия в обмене сообщениями, должны быть завершены. Сложный тип аргументов *MPI_Init* предусмотрен для того, чтобы передавать всем процессам аргументы *main*:

```
int main(int argc, char** argv)
{
    MPI_Init(&argc, &argv);
        ...
    MPI_Finalize();
}
```


int MPI_Comm_size(MPI_Comm comm, int* size)

Определение общего числа параллельных процессов в группе *comm*.

- *comm* - идентификатор группы
- OUT *size* - размер группы

int MPI_Comm_rank(MPI_Comm comm, int* rank)
Определение номера процесса в группе *comm*. Значение, возвращаемое по адресу *&rank*, лежит в диапазоне от 0 до *size_of_group-1*.

- *comm* - идентификатор группы
- OUT *rank* - номер вызывающего процесса в группе *comm*

double MPI_Wtime(void)
Функция возвращает астрономическое время в секундах (вещественное число), прошедшее с некоторого момента в прошлом. Гарантируется, что этот момент не будет изменен за время существования процесса.

Прием/передача сообщений между отдельными процессами.
Прием/передача сообщений с блокировкой
int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int msgtag, MPI_Comm comm)

- *buf* - адрес начала буфера посылки сообщения
- *count* - число передаваемых элементов в сообщении
- *datatype* - тип передаваемых элементов
- *dest* - номер процесса-получателя
- *msgtag* - идентификатор сообщения
- *comm* - идентификатор группы

Блокирующая посылка сообщения с идентификатором *msgtag*, состоящего из *count* элементов типа *datatype*, процессу с номером *dest*. Все элементы сообщения расположены подряд в буфере *buf*. Значение *count* может быть нулем. Тип передаваемых элементов *datatype* должен указываться с помощью предопределенных констант типа. Разрешается передавать сообщение самому себе.
Блокировка гарантирует корректность повторного использования всех параметров после возврата из подпрограммы. Выбор способа осуществления этой гарантии: копирование в промежуточный буфер или непосредственная передача процессу *dest*, остается за MPI. Следует специально отметить, что возврат из подпрограммы *MPI_Send* не означает ни того, что сообщение уже передано процессу *dest*, ни того, что сообщение покинуло процессорный элемент, на котором выполняется процесс, выполнивший *MPI_Send*.

int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int msgtag, MPI_Comm comm, MPI_Status *status)

- OUT *buf* - адрес начала буфера приема сообщения
- *count* - максимальное число элементов в принимаемом сообщении
- *datatype* - тип элементов принимаемого сообщения
- *source* - номер процесса-отправителя
- *msgtag* - идентификатор принимаемого сообщения
- *comm* - идентификатор группы
- OUT *status* - параметры принятого сообщения

Прием сообщения с идентификатором *msgtag* от процесса *source* с блокировкой. Число элементов в принимаемом сообщении не должно превосходить значения *count*. Если число принятых элементов меньше значения *count*, то гарантируется, что в буфере *buf* изменятся только элементы, соответствующие элементам принятого сообщения. Если нужно узнать точное число элементов в сообщении, то можно воспользоваться подпрограммой *MPI_Probe*.

Блокировка гарантирует, что после возврата из подпрограммы все элементы сообщения приняты и расположены в буфере *buf*.

В качестве номера процесса-отправителя можно указать предопределенную константу *MPI_ANY_SOURCE* - признак того, что подходит сообщение от любого процесса. В качестве идентификатора принимаемого сообщения можно указать константу *MPI_ANY_TAG* - признак того, что подходит сообщение с любым идентификатором.

Если процесс посылает два сообщения другому процессу и оба эти сообщения соответствуют одному и тому же вызову *MPI_Recv*, то первым будет принято то сообщение, которое было отправлено раньше.

int MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count)

- *status* - параметры принятого сообщения
- *datatype* - тип элементов принятого сообщения
- OUT *count* - число элементов сообщения

По значению параметра *status* данная подпрограмма определяет число уже принятых (после обращения к *MPI_Recv*) или принимаемых (после обращения к *MPI_Probe* или *MPI_Iprobe*) элементов сообщения типа *datatype*.

int MPI_Probe(int source, int msgtag, MPI_Comm comm, MPI_Status *status)

- *source* - номер процесса-отправителя или *MPI_ANY_SOURCE*
- *msgtag* - идентификатор ожидаемого сообщения или *MPI_ANY_TAG*
- *comm* - идентификатор группы
- OUT *status* - параметры обнаруженного сообщения

Получение информации о структуре ожидаемого сообщения с блокировкой. Возврата из подпрограммы не произойдет до тех пор, пока сообщение с подходящим идентификатором и номером процесса-отправителя не будет доступно для получения. Атрибуты доступного сообщения можно определить обычным образом с помощью параметра *status*. Следует обратить внимание, что подпрограмма определяет только факт прихода сообщения, но реально его не принимает.

Прием/передача сообщений без блокировки.

int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int msgtag, MPI_Comm comm, MPI_Request *request)

- *buf* - адрес начала буфера посылки сообщения
- *count* - число передаваемых элементов в сообщении
- *datatype* - тип передаваемых элементов
- *dest* - номер процесса-получателя
- *msgtag* - идентификатор сообщения
- *comm* - идентификатор группы
- OUT *request* - идентификатор асинхронной передачи

Передача сообщения, аналогичная *MPI_Send*, однако возврат из подпрограммы происходит сразу после инициализации процесса передачи без ожидания обработки всего сообщения, находящегося в буфере *buf*. Это означает, что нельзя повторно использовать данный буфер для других целей без получения дополнительной информации о завершении данной посылки. Окончание процесса передачи (т.е. того момента, когда можно переиспользовать буфер *buf* без опасения испортить передаваемое сообщение) можно определить с помощью параметра *request* и процедур *MPI_Wait* и *MPI_Test*.

Сообщение, отправленное любой из процедур *MPI_Send* и *MPI_Isend*, может быть принято любой из процедур *MPI_Recv* и *MPI_Irecv*.

int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int msgtag, MPI_Comm comm, MPI_Request *request)

- OUT *buf* - адрес начала буфера приема сообщения
- *count* - максимальное число элементов в принимаемом сообщении
- *datatype* - тип элементов принимаемого сообщения
- *source* - номер процесса-отправителя
- *msgtag* - идентификатор принимаемого сообщения
- *comm* - идентификатор группы
- OUT *request* - идентификатор асинхронного приема сообщения

Прием сообщения, аналогичный *MPI_Recv*, однако возврат из подпрограммы происходит сразу после инициализации процесса приема без ожидания получения сообщения в буфере *buf*. Окончание процесса приема можно определить с помощью параметра *request* и процедур *MPI_Wait* и *MPI_Test*.

int MPI_Wait(MPI_Request *request, MPI_Status *status)

- *request* - идентификатор асинхронного приема или передачи
- OUT *status* - параметры сообщения

Ожидание завершения асинхронных процедур *MPI_Isend* или *MPI_Irecv*, ассоциированных с идентификатором *request*. В случае приема, атрибуты и длину полученного сообщения можно определить обычным образом с помощью параметра *status*.

int MPI_Waitall(int count, MPI_Request *requests, MPI_Status *statuses)

- *count* - число идентификаторов
- *requests* - массив идентификаторов асинхронного приема или передачи
- OUT *statuses* - параметры сообщений

Выполнение процесса блокируется до тех пор, пока все операции обмена, ассоциированные с указанными идентификаторами, не будут завершены. Если во время одной или нескольких операций обмена возникли ошибки, то поле ошибки в элементах массива *statuses* будет установлено в соответствующее значение.

int MPI_Waitany(int count, MPI_Request *requests, int *index, MPI_Status *status)

- *count* - число идентификаторов
- *requests* - массив идентификаторов асинхронного приема или передачи
- OUT *index* - номер завершенной операции обмена
- OUT *status* - параметры сообщений

Выполнение процесса блокируется до тех пор, пока какая-либо операция обмена, ассоциированная с указанными идентификаторами, не будет завершена. Если несколько операций могут быть завершены, то случайным образом выбирается одна из них. Параметр *index* содержит номер элемента в массиве *requests*, содержащего идентификатор завершенной операции.

int MPI_Waitsome(int incount, MPI_Request *requests, int *outcount, int *indexes, MPI_Status *statuses)

- *incount* - число идентификаторов
- *requests* - массив идентификаторов асинхронного приема или передачи
- OUT *outcount* - число идентификаторов завершившихся операций обмена
- OUT *indexes* - массив номеров завершившихся операции обмена
- OUT *statuses* - параметры завершившихся сообщений

Выполнение процесса блокируется до тех пор, пока по крайней мере одна из операций обмена, ассоциированных с указанными идентификаторами, не будет завершена. Параметр *outcount* содержит число завершенных операций, а первые *outcount* элементов

массива *indexes* содержат номера элементов массива *requests* с их идентификаторами. Первые *outcount* элементов массива *statuses* содержат параметры завершенных операций.

int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)

- *request* - идентификатор асинхронного приема или передачи
- OUT *flag* - признак завершенности операции обмена
- OUT *status* - параметры сообщения

Проверка завершенности асинхронных процедур *MPI_Isend* или *MPI_Irecv*, ассоциированных с идентификатором *request*. В параметре *flag* возвращает значение 1, если соответствующая операция завершена, и значение 0 в противном случае. Если завершена процедура приема, то атрибуты и длину полученного сообщения можно определить обычным образом с помощью параметра *status*.

int MPI_Testall(int count, MPI_Request *requests, int *flag, MPI_Status *statuses)

- *count* - число идентификаторов
- *requests* - массив идентификаторов асинхронного приема или передачи
- OUT *flag* - признак завершенности операций обмена
- OUT *statuses* - параметры сообщений

В параметре *flag* возвращает значение *1*, если все операции, ассоциированные с указанными идентификаторами, завершены (с указанием параметров сообщений в массиве *statuses*). В противном случае возвращается *0*, а элементы массива *statuses* неопределены.

int MPI_Testany(int count, MPI_Request *requests, int *index, int *flag, MPI_Status *status)

- *count* - число идентификаторов
- *requests* - массив идентификаторов асинхронного приема или передачи
- OUT *index* - номер завершенной операции обмена
- OUT *flag* - признак завершенности операции обмена
- OUT *status* - параметры сообщения

Если к моменту вызова подпрограммы хотя бы одна из операций обмена завершилась, то в параметре *flag* возвращается значение *1*, *index* содержит номер соответствующего элемента в массиве *requests*, а *status* - параметры сообщения.

int MPI_Testsome(int incount, MPI_Request *requests, int *outcount, int *indexes, MPI_Status *statuses)

- *incount* - число идентификаторов
- *requests* - массив идентификаторов асинхронного приема или передачи
- OUT *outcount* - число идентификаторов завершившихся операций обмена
- OUT *indexes* - массив номеров завершившихся операции обмена
- OUT *statuses* - параметры завершившихся операций

Данная подпрограмма работает так же, как и *MPI_Waitsome*, за исключением того, что возврат происходит немедленно. Если ни одна из указанных операций не завершилась, то значение *outcount* будет равно нулю.

int MPI_Iprobe(int source, int msgtag, MPI_Comm comm, int *flag, MPI_Status *status)

- *source* - номер процесса-отправителя или *MPI_ANY_SOURCE*
- *msgtag* - идентификатор ожидаемого сообщения или *MPI_ANY_TAG*
- *comm* - идентификатор группы
- OUT *flag* - признак завершенности операции обмена
- OUT *status* - параметры обнаруженного сообщения

Получение информации о поступлении и структуре ожидаемого сообщения без блокировки. В параметре *flag* возвращает значение *1*, если сообщение с подходящими атрибутами уже может быть принято (в этом случае ее действие полностью аналогично *MPI_Probe*), и значение *0*, если сообщения с указанными атрибутами еще нет.

Объединение запросов на взаимодействие.
Процедуры данной группы позволяют снизить накладные расходы, возникающие в рамках одного процессора при обработке приема/передачи и перемещении необходимой информации между процессом и сетевым контроллером. Несколько запросов на прием и/или передачу могут объединяться вместе для того, чтобы далее их можно было бы запустить одной командой. Способ приема сообщения никак не зависит от способа его посылки: сообщение, отправленное с помощью объединения запросов либо обычным способом, может быть принято как обычным способом, так и с помощью объединения запросов.

int MPI_Send_init(void *buf, int count, MPI_Datatype datatype, int dest, int msgtag, MPI_Comm comm, MPI_Request *request)

- *buf* - адрес начала буфера посылки сообщения
- *count* - число передаваемых элементов в сообщении
- *datatype* - тип передаваемых элементов
- *dest* - номер процесса-получателя
- *msgtag* - идентификатор сообщения
- *comm* - идентификатор группы
- OUT *request* - идентификатор асинхронной передачи

Формирование запроса на выполнение пересылки данных. Все параметры точно такие же, как и у подпрограммы *MPI_Isend*, однако в отличие от нее пересылка не начинается до вызова подпрограммы *MPI_Startall*.

int MPI_Recv_init(void *buf, int count, MPI_Datatype datatype, int source, int msgtag, MPI_Comm comm, MPI_Request *request)

- OUT *buf* - адрес начала буфера приема сообщения
- *count* - число принимаемых элементов в сообщении
- *datatype* - тип принимаемых элементов
- *source* - номер процесса-отправителя
- *msgtag* - идентификатор сообщения
- *comm* - идентификатор группы
- OUT *request* - идентификатор асинхронного приема

Формирование запроса на выполнение приема данных. Все параметры точно такие же, как и у подпрограммы *MPI_Irecv*, однако в отличие от нее реальный прием не начинается до вызова подпрограммы *MPI_Startall*.

MPI_Startall(int count, MPI_Request *requests)

- *count* - число запросов на взаимодействие
- OUT *requests* - массив идентификаторов приема/передачи

Запуск всех отложенных взаимодействий, ассоциированных вызовами подпрограмм *MPI_Send_init* и *MPI_Recv_init* с элементами массива запросов *requests*. Все взаимодействия запускаются в режиме без блокировки, а их завершение можно определить обычным образом с помощью процедур *MPI_Wait* и *MPI_Test*.

Совмещенные прием/передача сообщений

int MPI_Sendrecv(void *sbuf, int scount, MPI_Datatype stype, int dest, int stag, void *rbuf, int rcount, MPI_Datatype rtype, int source, MPI_Datatype rtag, MPI_Comm comm, MPI_Status *status)

- *sbuf* - адрес начала буфера посылки сообщения
- *scount* - число передаваемых элементов в сообщении
- *stype* - тип передаваемых элементов
- *dest* - номер процесса-получателя
- *stag* - идентификатор посылаемого сообщения
- OUT *rbuf* - адрес начала буфера приема сообщения
- *rcount* - число принимаемых элементов сообщения
- *rtype* - тип принимаемых элементов
- *source* - номер процесса-отправителя
- *rtag* - идентификатор принимаемого сообщения
- *comm* - идентификатор группы
- OUT *status* - параметры принятого сообщения

Данная операция объединяет в едином запросе посылку и прием сообщений. Принимающий и отправляющий процессы могут являться одним и тем же процессом. Сообщение, отправленное операцией *MPI_Sendrecv*, может быть принято обычным образом, и точно также операция *MPI_Sendrecv* может принять сообщение, отправленное обычной операцией *MPI_Send*. Буфера приема и посылки обязательно должны быть различными.

Коллективные взаимодействия процессов.

В операциях коллективного взаимодействия процессов участвуют все процессы коммуникатора. Соответствующая процедура должна быть вызвана каждым процессом, быть может, со своим набором параметров. Возврат из процедуры коллективного взаимодействия может произойти в тот момент, когда участие процесса в данной операции уже закончено. Как и для блокирующих процедур, возврат означает то, что разрешен свободный доступ к буферу приема или посылки, но не означает ни того, что операция завершена другими процессами, ни даже того, что она ими начата (если это возможно по смыслу операции).

int MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int source, MPI_Comm comm)

- OUT *buf* - адрес начала буфера посылки сообщения
- *count* - число передаваемых элементов в сообщении
- *datatype* - тип передаваемых элементов
- *source* - номер рассылающего процесса
- *comm* - идентификатор группы

Рассылка сообщения от процесса *source* всем процессам, включая рассылающий процесс. При возврате из процедуры содержимое буфера *buf* процесса *source*будет скопировано в локальный буфер процесса. Значения параметров *count*, *datatype* и *source* должны быть одинаковыми у всех процессов.

int MPI_Gather(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, int dest, MPI_Comm comm)

- *sbuf* - адрес начала буфера посылки
- *scount* - число элементов в посылаемом сообщении
- *stype* - тип элементов отсылаемого сообщения
- OUT *rbuf* - адрес начала буфера сборки данных
- *rcount* - число элементов в принимаемом сообщении
- *rtype* - тип элементов принимаемого сообщения
- *dest* - номер процесса, на котором происходит сборка данных
- *comm* - идентификатор группы
- OUT *ierror* - код ошибки

Сборка данных со всех процессов в буфере *rbuf* процесса *dest*. Каждый процесс, включая *dest*, посылает содержимое своего буфера *sbuf* процессу *dest*. Собирающий процесс сохраняет данные в буфере *rbuf*, располагая их в порядке возрастания номеров процессов. Параметр *rbuf* имеет значение только на собирающем процессе и на остальных игнорируется, значения параметров *count*, *datatype* и *dest* должны быть одинаковыми у всех процессов.

int MPI_Allreduce(void *sbuf, void *rbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

- *sbuf* - адрес начала буфера для аргументов
- OUT *rbuf* - адрес начала буфера для результата
- *count* - число аргументов у каждого процесса
- *datatype* - тип аргументов
- *op* - идентификатор глобальной операции
- *comm* - идентификатор группы

Выполнение *count* глобальных операций *op* с возвратом *count* результатов во всех процессах в буфере *rbuf*. Операция выполняется независимо над соответствующими аргументами всех процессов. Значения параметров *count* и *datatype* у всех процессов должны быть одинаковыми. Из соображений эффективности реализации предполагается, что операция *op* обладает свойствами ассоциативности и коммутативности.

int MPI_Reduce(void *sbuf, void *rbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

- *sbuf* - адрес начала буфера для аргументов
- OUT *rbuf* - адрес начала буфера для результата
- *count* - число аргументов у каждого процесса
- *datatype* - тип аргументов
- *op* - идентификатор глобальной операции
- *root* - процесс-получатель результата
- *comm* - идентификатор группы

Функция аналогична предыдущей, но результат будет записан в буфер *rbuf* только у процесса *root*.

Синхронизация процессов

int MPI_Barrier(MPI_Comm comm)

- *comm* - идентификатор группы

Блокирует работу процессов, вызвавших данную процедуру, до тех пор, пока все оставшиеся процессы группы *comm* также не выполнят эту процедуру.

Работа с группами процессов

int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm *newcomm)

- *comm* - идентификатор группы
- *color* - признак разделения на группы
- *key* - параметр, определяющий нумерацию в новых группах
- OUT *newcomm* - идентификатор новой группы

Данная процедура разбивает все множество процессов, входящих в группу *comm*, на непересекающиеся подгруппы - одну подгруппу на каждое значение параметра *color* (неотрицательное число). Каждая новая подгруппа содержит все процессы одного цвета. Если в качестве *color* указано значение *MPI_UNDEFINED*, то в *newcomm* будет возвращено значение *MPI_COMM_NULL*.

int MPI_Comm_free(MPI_Comm comm)

- OUT *comm* - идентификатор группы

Уничтожает группу, ассоциированную с идентификатором *comm*, который после возвращения устанавливается в *MPI_COMM_NULL*.

Предопределенные константы.
Предопределенные константы типа элементов сообщений.

Константы MPI	Тип в C
MPI_CHAR	signed char
MPI_SHORT	signed int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

Другие предопределенные типы
MPI_Status - структура; атрибуты сообщений; содержит три обязательных поля:
- *MPI_Source* (номер процесса отправителя)
- *MPI_Tag* (идентификатор сообщения)
- *MPI_Error* (код ошибки)

MPI_Request - системный тип; идентификатор операции посылки-приема сообщения

MPI_Comm - системный тип; идентификатор группы (коммуникатора)
- *MPI_COMM_WORLD* - зарезервированный идентификатор группы, состоящей их всех процессов приложения

Константы-пустышки
- *MPI_COMM_NULL*
- *MPI_DATATYPE_NULL*
- *MPI_REQUEST_NULL*

Константа неопределенного значения
- *MPI_UNDEFINED*

Глобальные операции
MPI_MAX
MPI_MIN
MPI_SUM
MPI_PROD

Любой процесс/идентификатор
MPI_ANY_SOURCE
MPI_ANY_TAG

Код успешного завершения процедуры *MPI_SUCCESS*

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер с характеристиками, позволяющими установить Microsoft Visual Studio 2010, MPI.

Указания по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения, ремонт персонального компьютера. Соблюдать правила технической безопасности при работе с электрооборудованием.

Методика и порядок выполнения работы

Рассмотрение начнем на примере демонстрационной программы - "Hello world".

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char* argv[])
{
int rank, numprocs;
// инициализация библиотеки MPI
MPI_Init(&argc, &argv);
// получение количества процессов в программе
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
// получение ранга процесса
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
printf ("\n Hello from process %3d", rank);
// завершение библиотеки MPI
MPI_Finalize();
return 0;
}
```

В качестве следующего примера рассмотрим расчет экспоненты (e).

Один из вариантов ее нахождения — ряд Тейлора: **e^x=∑((x^n)/n!)**, где суммирование происходит от n=0 до бесконечности.

Данная формула легко поддается распараллеливанию, так как искомое число является суммой отдельных слагаемых и благодаря этому каждый отдельный процессор может заняться вычислением отдельных слагаемых.

Количество слагаемых, которое будет рассчитываться в каждом отдельно взятом процессоре, зависит как и от длины интервала n, так и от имеющегося количества процессоров k, которые смогут участвовать в процессе вычисления. Так, например, если длина интервала n=4, а в вычислениях участвуют пять процессоров (k=5), то с первого по четвертый процессоры получат по одному слагаемому, а пятый будет не задействован. В случае же если n=10, а k=5, каждому процессору достанется по два слагаемых для вычисления.

Изначально, первый процессор с помощью функции широковещательной рассылки MPI_Bcast отправляет остальным значение заданной пользователями переменной n. В общем случае функция MPI_Bcast имеет следующий формат: int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm), где buffer – это адрес буфера с элементом, count – количество элементов, datatype – соответствующий тип данных в MPI, root – ранг главного процессора, который занимается пересылкой, а comm- имя коммуникатора. В нашем случае в роли главного процессора, как уже говорилось, будет выступать первый процессор с рангом 0.

После того число n будет успешно отправлено, каждый процессор займется вычислением своих слагаемых. Для этого в каждом шаге цикла к числу i, которое

изначально равно рангу процессора, будет прибавляться число, равное количеству процессоров участвующих в вычислениях. Если число в ходе следующих действий число i превысит заданное пользователем число n, выполнение цикла для данного процессора остановится. В ходе выполнения цикла слагаемые будут прибавляться в отдельную переменную и, после его завершения, полученная сумма отправится в главный процессор. Для этого будет использоваться функция операции приведения MPI_Reduce. В общем виде она выглядит следующим образом: int MPI_Reduce(void *buf, void *result, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm) Она объединяет элементы входного буфера каждого процесса в группе, используя операцию op, и возвращает объединенное значение в выходной буфер процесса с номером root. Результатом такой операции будет единственное значение, благодаря чему функция приведения и получила свое название.

После выполнения программы на всех процессорах, первый процессор получит общую сумму слагаемых, которая и будет являться нужным нам значение экспоненты.

Первостепенной задачей все же является нахождение значения экспоненты и если процессоры начнут вычислять каждый факториал каждого слагаемого раздельным образом, это может привести к прямо обратно эффекту, а именно значительной потери в производительности и скорости вычисления.

Объясняется это тем, что в данном случае начнется весьма большая нагрузка на коммуникационную среду, которая и без того зачастую является слабым звеном в системах параллельных вычислений. Если же вычисление факториала будет происходить на каждом процессоре частным образом, нагрузка на линии коммуникаций будет минимальна. Данный случай можно назвать хорошим примером того, что и задача распараллеливания тоже должна порой иметь свои границы.

Алгоритм выполнения кода.

1. Из визуальной оболочки в программу передается значение числа n, которое затем с помощью функции широковещательной рассылки отправляется по всем процессорам.

2. При инициализации первого главного процессора, запускается таймер.

3. Каждый процессор выполняет цикл, где значением приращения является количество процессоров в системе. В каждой итерации цикла вычисляется слагаемое и сумма таких слагаемых сохраняется в переменную drobSum.

4. После завершения цикла каждый процессор суммирует свое значение drobSum к переменной Result, используя для этого функцию приведения MPI_Reduce.

5. После завершения расчетов на всех процессорах, первый главный процессор останавливает таймер и отправляет в поток вывода получившееся значение переменной Result.

6. В поток вывода отправляется также и отмеренное нашим таймером значение времени в милисекундах.

Листинг кода.

Программа написана на C++, аргументы для выполнения передаются из внешней оболочки. Код выглядит следующим образом:

```
#include "mpi.h"
#include <iostream>
#include <windows.h>
using namespace std;

double Fact(int n)
{
  if (n==0)
    return 1;
  else
```

```
  return n*Fact(n-1);
}

int main(int argc, char *argv[])
{
 SetConsoleOutputCP(1251);
 int n;
 int myid;
 int numprocs;
 int i;
 int rc;
 long double drob,drobSum=0,Result, sum;
 double startwtime = 0.0;
 double endwtime;

 n = atoi(argv[1]);

 if (rc= MPI_Init(&argc, &argv))
 {
  cout << "Ошибка запуска, выполнение остановлено " << endl;
  MPI_Abort(MPI_COMM_WORLD, rc);
 }

MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);

 if (myid == 0)
 {

  startwtime = MPI_Wtime();
 }
 MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
 for (i = myid; i <= n; i += numprocs)
 {
  drob = 1/Fact(i);
  drobSum += drob;
 }

MPI_Reduce(&drobSum, &Result, 1, MPI_LONG_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
cout.precision(20);
 if (myid == 0)
 {
  cout << Result << endl;
  endwtime = MPI_Wtime();
  cout << (endwtime-startwtime)*1000 << endl;
 }
 MPI_Finalize();
 return 0;
}
```

Получена программа для подсчета экспоненты с использованием сразу нескольких процессоров.

Задания:

1. Реализуйте и усовершенствуйте программу подсчета экспоненты более рациональным решением записи результата в файл.
2. В исходном тексте программы на языке С пропущены вызовы процедур подключения к MPI, определения количества процессов и ранга процесса. Добавьте эти вызовы, откомпилируйте и запустите программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[ ])
{
int myid, numprocs;
....
fprintf(stdout,"Process %d of %d\n", myid, numprocs);
MPI_Finalize();
return 0;
}
```

Содержание отчета и его форма

Подготовьте отчет, в котором опишите выполнение заданий.

Отчет по лабораторной работе должен содержать:

- название работы;
- цель лабораторной работы;
- формулировку задания и технологию его выполнения;
- ответы на контрольные вопросы.

Контрольные вопросы:

1. Каковы преимущества программирования на MPI?
2. Какой минимальный набор средств является достаточным для организации параллельных вычислений в системах с распределенной памятью?
3. В чем различие понятий процесса и процессора?
4. Как описываются в MPI передаваемые сообщения?
5. В чем различие парных и коллективных операций передачи данных?
6. Какая функция MPI обеспечивает передачу данных от одного процесса всем процессам?
7. Какие режимы передачи данных поддерживаются в MPI?

Защита лабораторной работы

По результатам отчета, представленного в письменной форме, проводится собеседование, которое имеет контролирующую и учебную функции.

Лабораторная работа 4.

Ознакомление с системой параллельного программирования OpenMP.

Цель и содержание: Создание параллельного варианта программы, реализующей вычисления числа Пи. Исследование эффективности работающих вариантов программ реализуемых с помощью OpenMP и MPI.

Организационная форма занятий: лабораторное занятие (компьютерные симуляции).

Вопросы для обсуждения на лабораторном занятии: Основные идеи, причина возникновения, модель программирования. Стандарт OpenMP. Сравнение с MPI и другими средствами. Механизм работы.

Теоретическое обоснование

Одним из наиболее популярных средств программирования для компьютеров с общей памятью, базирующихся на традиционных языках программирования и использовании специальных комментариев, в настоящее время является технология OpenMP. За основу берётся последовательная программа, а для создания её параллельной версии пользователю предоставляется набор директив, функций и переменных окружения. Предполагается, что создаваемая параллельная программа будет переносимой между различными компьютерами с разделяемой памятью, поддерживающими OpenMP API.

Для использования механизмов OpenMP нужно скомпилировать программу компилятором, поддерживающим OpenMP, с указанием соответствующего ключа (например, в icc/ifort используется ключ компилятора –openmp, в gcc /gfortran –fopenmp, Sun Studio -xopenmp, в Visual C++ - /openmp, в PGI -mp). Компилятор интерпретирует директивы OpenMP и создаёт параллельный код. При использовании компиляторов, не поддерживающих OpenMP, директивы OpenMP игнорируются без дополнительных сообщений.

Компилятор с поддержкой OpenMP определяет макрос _OPENMP, который может использоваться для условной компиляции отдельных блоков, харак- терных для параллельной версии программы. Этот макрос определён в формате yyyymm, где yyyy и mm – цифры года и месяца, когда был принят поддерживаемый стандарт OpenMP. Например, компилятор, поддерживающий стандарт OpenMP 3.0, определяет _OPENMP в 200805.

Для проверки того, что компилятор поддерживает какую-либо версию OpenMP, достаточно написать директивы условной компиляции #ifdef или #ifndef. Простейший пример условной компиляции в программах на языке Си приведен ниже.

```
#include <stdio.h>
int main(){
#ifdef _OPENMP
   printf("OpenMP is supported!\n");
#endif
}
```

Модель параллельной программы.

Распараллеливание в OpenMP выполняется явно при помощи вставки в текст программы специальных директив, а также вызова вспомогательных функ- ций. При использовании OpenMP предполагается SPMD-модель (Single Program Multiple Data) параллельного программирования, в рамках которой для всех параллельных нитей используется один и тот же код.

Программа начинается с последовательной области – сначала работает один процесс (нить), при входе в параллельную область порождается ещё некоторое число процессов, между которыми в дальнейшем распределяются части кода. По завершении параллельной области все нити, кроме одной (нити-мастера), завершаются, и начинается последовательная область. В программе может быть любое количество параллельных и последовательных областей. Кроме того, параллельные области могут быть также вложенными друг в друга. В отличие от полноценных процессов, порождение нитей является относительно быстрой операцией, поэтому частые порождения и завершения нитей не так сильно влияют на время выполнения программы.

Для написания эффективной параллельной программы необходимо, чтобы все нити, участвующие в обработке программы, были равномерно загружены полезной работой. Это достигается тщательной балансировкой загрузки, для чего предназначены различные механизмы OpenMP.

Существенным моментом является также необходимость синхронизации доступа к общим данным. Само наличие данных, общих для нескольких нитей, приводит к конфликтам при одновременном несогласованном доступе. Поэтому значительная часть функциональности OpenMP предназначена для осуществления различного рода синхронизаций работающих нитей.

OpenMP не выполняет синхронизацию доступа различных нитей к одним и тем же файлам. Если это необходимо для корректности программы, пользователь должен явно использовать директивы синхронизации или соответствующие библиотечные функции. При доступе каждой нити к своему файлу никакая синхронизация не требуется.

Директивы и функции.

Значительная часть функциональности OpenMP реализуется при помощи директив компилятору. Они должны быть явно вставлены пользователем, что позволит выполнять программу в параллельном режиме. Директивы OpenMP в программах на языке Си начинаются указаниями препроцессору, начинающимися с #pragma omp. Ключевое слово omp используется для того, чтобы исключить случайные совпадения имён директив OpenMP с другими именами.

Формат директивы на Си/Си++: #pragma omp directive-name [опция[[,] опция]...].

Объектом действия большинства директив является один оператор или блок, перед которым расположена директива в исходном тексте программы. В OpenMP такие операторы или блоки называются ассоциированными с директивой. Ассоциированный блок должен иметь одну точку входа в начале и одну точку выхода в конце. Порядок опций в описании директивы несуществе- нен, в одной директиве большинство опций может встречаться несколько раз. После некоторых опций может следовать список переменных, разделяемых запятыми.

Все директивы OpenMP можно разделить на 3 категории: определение параллельной области, распределение работы, синхронизация. Каждая директива может иметь несколько дополнительных атрибутов – опций (clause). Отдельно специфицируются опции для назначения классов переменных, которые могут быть атрибутами различных директив.

Чтобы задействовать функции библиотеки OpenMP периода выполнения (исполняющей среды), в программу нужно включить заголовочный файл omp.h. Если вы используете в приложении только OpenMP-директивы, включать этот файл не требуется. Функции назначения параметров имеют приоритет над соответствующими переменными окружения.

Все функции, используемые в OpenMP, начинаются с префикса omp_. Если пользователь не будет использовать в программе имён, начинающихся с такого префикса, то конфликтов с объектами OpenMP заведомо не будет. В языке Си, кроме того, является существенным регистр символов в названиях функций. Названия функций OpenMP записываются строчными буквами.

Для того чтобы программа, использующая функции OpenMP, могла оставаться корректной для обычного компилятора, можно прилинковать специальную библиотеку, которая определит для каждой функции соответствующую «заглушку» (stub). Например, в компиляторе Intel соответствующая библиотека подключается заданием ключа –openmp-stubs.

Выполнение программы.

После получения выполняемого файла необходимо запустить его на требуемом количестве процессоров. Для этого обычно нужно задать количество нитей, выполняющих параллельные области программы, определив значение переменной среды OMP_NUM_THREADS. Например, в Linux в командной оболочке bash это можно сделать при помощи следующей команды:

```
export OMP_NUM_THREADS=n
```

После запуска начинает работать одна нить, а внутри параллельных областей одна и та же программа будет выполняться всем набором нитей. Стандарт- ный вывод программы в зависимости от системы будет выдаваться на терми- нал или записываться в файл с предопределенным именем.

Замер времени.

В OpenMP предусмотрены функции для работы с системным таймером.

Функция omp_get_wtime() возвращает в вызвавшей нити астрономическое время в секундах (вещественное число двойной точности), прошедшее с не- которого момента в прошлом.

Си:

double omp_get_wtime(void).

Если некоторый участок программы окружить вызовами данной функции, то разность возвращаемых значений покажет время работы данного участка. Гарантируется, что момент времени, используемый в качестве точки отсчета, 12 не будет изменён за время существования процесса. Таймеры разных нитей могут быть не синхронизированы и выдавать различные значения.

Функция omp_get_wtick() возвращает в вызвавшей нити разрешение таймера в секундах.

Это время можно рассматривать как меру точности таймера.

Си:

double omp_get_wtick(void);

Пример 2 иллюстрирует применение функций omp_get_wtime() и omp_get_wtick() для работы с таймерами в OpenMP. В данном примере производится замер начального времени, затем сразу замер конечного време- ни. Разность времён даёт время на замер времени. Кроме того, измеряется точность системного таймера.

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   double start_time, end_time, tick;
   start_time = omp_get_wtime();
   end_time = omp_get_wtime();
   tick = omp_get_wtick();
   printf("Время на замер времени %lf\n", end_time-start_time);
   printf("Точность таймера %lf\n", tick);
}
```

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер с характеристиками, позволяющими установить Microsoft Visual Studio 2010, OpenMP.

Указания по технике безопасности. Самостоятельно не производить: установку и удаление программного обеспечения, ремонт персонального компьютера. Соблюдать правила технической безопасности при работе с электрооборудованием.

Методика и порядок выполнения работы

Задания:

1. Определите, какую версию стандарта OpenMP поддерживает компилятор на доступной системе.
2. Реализуйте программу вычисления числа Пи. Определите процессорное время, потраченное на выполнение расчета.

```
#include <stdio.h>
double f(double y) {return(4.0/(1.0+y*y));}
int main()
{
   double w, x, sum, pi;
   int i;
   int n = 1000000;
   w = 1.0/n;
   sum = 0.0;
#pragma omp parallel for private(x) shared(w)\
            reduction(+:sum)
   for(i=0; i < n; i++)
   {
      x = w*(i-0.5);
      sum = sum + f(x);
   }
   pi = w*sum;
   printf("pi = %f\n", pi);
}
```

3. Распараллельте программу с помощью MPI. Определите процессорное время, потраченное на выполнение расчета. Сравните с результатами, полученными в задании 2.

Содержание отчета и его форма

Подготовьте отчет, в котором опишите выполнение заданий.

Отчет по лабораторной работе должен содержать:

- название работы;
- цель лабораторной работы;
- формулировку задания и технологию его выполнения;
- ответы на контрольные вопросы.

Контрольные вопросы:

1. Каковы преимущества программирования с использованием OpenMP?
2. Может ли программа на OpenMP состоять только из параллельных областей? Только из последовательных областей?
3. В чем состоит концепция нитей?
4. Чем отличается нить-мастер от всех остальных нитей?
5. Как достигается балансировка нагрузки в OpenMP?

Защита лабораторной работы

По результатам отчета, представленного в письменной форме, проводится собеседование, которое имеет контролирующую и учебную функции.

4. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Оценка «отлично» выставляется студенту, если он продемонстрировал глубокие, исчерпывающие знания и творческие способности в понимании, изложении и использовании учебно-программного материала; логически последовательные, содержательные, полные, правильные и конкретные ответы на все поставленные вопросы и дополнительные вопросы преподавателя; свободное владение основной и дополнительной литературой, рекомендованной учебной программой.

Оценка «хорошо» выставляется студенту, если он продемонстрировал твердые и достаточно полные знания всего программного материала, правильное понимание сущности и взаимосвязи рассматриваемых процессов и явлений; последовательные,

правильные, конкретные ответы на поставленные вопросы при свободном устранении замечаний по отдельным вопросам; достаточное владение литературой, рекомендованной учебной программой.

Оценка «удовлетворительно» выставляется студенту, если он продемонстрировал твердые знания и понимание основного программного материала; правильные, без грубых ошибок ответы на поставленные вопросы при устранении неточностей и несущественных ошибок в освещении отдельных положений при наводящих вопросах преподавателя; недостаточное владение литературой, рекомендованной учебной программой.

Оценка «неудовлетворительно» выставляется студенту, если он продемонстрировал неправильные ответы на основные вопросы, допущены грубые ошибки в ответах, непонимание сущности излагаемых вопросов; неуверенные и неточные ответы на дополнительные вопросы.

5. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ, ОПРЕДЕЛЯЮЩИЕ ПРОЦЕДУРЫ ОЦЕНИВАНИЯ ЗНАНИЙ, УМЕНИЙ, НАВЫКОВ И (ИЛИ) ОПЫТА ДЕЯТЕЛЬНОСТИ, ХАРАКТЕРИЗУЮЩИХ ЭТАПЫ ФОРМИРОВАНИЯ КОМПЕТЕНЦИЙ

Текущая аттестация студентов проводится преподавателями, ведущими лабораторные занятия по дисциплине, в следующих формах: отчет письменный, коллоквиум, контрольная работа.

Допуск к лабораторным работам происходит при наличии у студентов печатного варианта отчета. Защита отчета проходит в форме доклада студента по выполненной работе и ответов на вопросы преподавателя.

Оценку «отлично» студент получает, если оформление отчета соответсвует установленным требованиям, правильно отвечает на предложенные преподавателем контрольные вопросы, правильно отвечает на дополнительные вопросы по теме лабораторной работы.

Оценку «хорошо» студент получает, если оформление отчета соответсвует установленным требованиям, правильно отвечает на предложенные преподавателем контрольные вопросы.

Оценку «удовлетворительно» студент получаетбез беседы с преподавателем, если оформление отчета соответсвует установленным требованиям.

Отчет может быть отправлен на доработку в следующих случаях:

- польностью не соовествует установленным требованиям;
- не раскрыта суть работы.

6. . УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Рекомендуемая литература

Основная литература

1. Бройдо, В.Л. Вычислительные системы, сети и телекоммуникации: учебник/ В. Л. Бройдо, О. П. Ильина- СПб.: Питер, 2011.
2. Пятибратов, А.П. Вычислительные системы, сети и телекоммуникации: учебник/ А. П. Пятибратов, Л. П. Гудыно, А. А. Кириченко; ред. А. П. Пятибратов - М.: Финансы и статистика, 2011.

Дополнительная литература

1. Олифер, В. Г. Компьютерныесети. Принципы, технологии, протоколы: учеб. пособие / В. Г. Олифер, Н. А. Олифер. - 4-е изд. - СПб.: Питер, 2011.
2. Таненбаум, Э. С. Архитектуракомпьютера / Э. С. Таненбаум; пер.: Ю. Гороховский, Д. Шинтяков. - 5-е изд. - СПб.: Питер, 2009.

Интернет-ресурсы

1. http://www.intuit.ru – сайт дистанционного образования в области информационных технологий;
2. http://www.iqlib.ru - интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия;
3. http://www.biblioclub.ru - электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике, а так же содержит материалы по точным наукам;
4. http://window.edu.ru – образовательные ресурсы ведущих вузов;

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ СТУДЕНТОВ ПО ОРГАНИЗАЦИИ И ПРОВЕДЕНИЮ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Направление подготовки	**09.04.02**
Направленность (профиль)	**Информационные системы и технологии «Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2023

СОДЕРЖАНИЕ

1. ЦЕЛЬ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Целью освоения дисциплины «Вычислительные системы» является получение магистрантами знаний о принципах организации современных вычислительных систем, их структуре и функционировании.

Основные задачи дисциплины:

- **ОЗНАКОМЛЕНИЕ С ВАЖНЕЙШИМИ ЭТАПАМИ И ТЕНДЕНЦИЯМИ В РАЗВИТИИ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ;**
- **ОЗНАКОМЛЕНИЕ С МЕТОДАМИ ОЦЕНКИ ПАРАМЕТРОВ КОМПОНЕНТ И СИСТЕМ В ЦЕЛОМ;**
- **ПРИОБРЕТЕНИЕ ТЕОРЕТИЧЕСКИХ ЗНАНИЙ И ПРАКТИЧЕСКИХ НАВЫКОВ ВЫБОРА И ИСПОЛЬЗОВАНИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ ДЛЯ ОБРАБОТКИ ИНФОРМАЦИИ.**

2. КОМПЕТЕНЦИИ ОБУЧАЮЩЕГОСЯ, ФОРМИРУЕМЫЕ В РЕЗУЛЬТАТЕ ИЗУЧЕНИЯ ДИСЦИПЛИНЫ

Индекс	Формулировка:
ПК-1	способность осуществлять управление, развитием баз данных, включая развертывание, сопровождение, оптимизацию функционирования баз данных, являющихся частью различных информационных систем
ПК-4	способность выполнять разработку систем управления базами данных, операционных систем, организацию разработки системного программного обеспечения, интеграция разработанного системного программного обеспечения

3. ТЕХНОЛОГИЧЕСКАЯ КАРТА САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТА

Коды реализуемых компетенций, индикатора(ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателем	Всего
2 семестр					
ПК-1 (ИД-1 $_{ПК-1}$, ИД-2, $_{ПК-1}$, ИД-3 $_{ПК-1}$), ПК-4 (ИД-1 $_{ПК-4}$, ИД-2, $_{ПК-4}$, ИД-3 $_{ПК-4}$)	Подготовка к лекциям	Коллоквиум	0,405	0,045	0,45

ПК-1 (ИД -1 пк -1, ИД-2, пк -1, ИД-3 пк -1), ПК-4 (ИД -1 пк- 4, ИД-2, пк -4, ИД-3 пк-4)	Самостоятельное изучение литературы	Коллоквиум	48,105	5,345	53,45
ПК-1 (ИД -1 пк -1, ИД-2, пк -1, ИД-3 пк -1), ПК-4 (ИД -1 пк- 4, ИД-2, пк -4, ИД-3 пк-4)	Подготовка и выполнение лабораторных работ	Отчет письменный	1,215	0,135	1,35
ПК-1 (ИД -1 пк -1, ИД-2, пк -1, ИД-3 пк -1), ПК-4 (ИД -1 пк- 4, ИД-2, пк -4, ИД-3 пк-4)	Выполнение контрольной работы	Контрольная работа	9	1	10
Итого за 2 семестр			**58,725**	**6,525**	**65,25**
Итого			**58,725**	**6,525**	**65,25**

4. СОДЕРЖАНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

4.1. **ПОДГОТОВКА** К ЛЕКЦИЯМ. САМОСТОЯТЕЛЬНОЕ ИЗУЧЕНИЕ ЛИТЕРАТУРЫ

ТЕМА 2. ПАРАЛЛЕЛИЗМ КАК ОСНОВА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ.

Тема самостоятельного изучения. Параллельные вычисления в России

Вид деятельности студентов: самостоятельное изучение учебно-методической литературы
Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: собеседование
План конспекта:
Основные российские производители современных высокопроизводительных компьютеров. Ученые и специалисты из России и ближнего зарубежья, работающие в области параллельных вычислений. Российские исследования в области параллельных и высокопроизводительных вычислений. Примеры организации вычислительных кластерных установок.
Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы

1-2	1-2	1-2	1-5

Параллельные структуры вычислительных систем

Два уровня распараллеливания

Развитие вычислительной техники характеризуется тем, что на каждом этапе новых разработок требования к производительности значительно превышают возможности элементной базы.

Это обусловлено задачами сложных систем управления в реальном времени, централизованным решением задач в сетях, имитационным моделированием сложных процессов (например, в ядерной физике), оперативным планированием и управлением и решением других задач исследования операций, преодолевающих "проклятие размерности". Такие задачи требуют концентрации вычислительных мощностей, постоянно поддерживая высокую актуальность проблемы создания супер-ЭВМ.

Уже давно стало ясно, что только структурными методами можно уравнять возможности вычислительных средств и требуемые скорости решения на них задач. Под структурными понимают методы распараллеливания работ. К распараллеливанию прибегают при проектировании отдельных устройств ЭВМ — устройств управления, буферов команд, каналов обращения к памяти и модулей памяти, многофункциональных арифметическо-логических устройств (АЛУ), повсеместно применяемых конвейеров и т.д. Но к распараллеливанию же прибегают и в проектировании совместной работы многих процессоров при параллельной или распределенной обработке информации, вводя в обращение термин "вычислительная система (ВС)".

Технический прогресс, несомненно, сказывается на росте частоты работы элементной (элементно-конструкторской) базы, на повышении степени интеграции, но благодаря ему появляются все новые задачи, требующие еще более значительного роста производительности вычислительных средств. Это можно считать законом, приводящим к новым уловкам при совмещении работы устройств ВС, при увеличении их количества в системе, при увеличении их эффективности в процессе решения задач.

Под эффективностью работы устройства в составе ВС понимают степень его участия в общей работе ВС при решении конкретной задачи — коэффициент загрузки устройства. Распараллеливание работ оправдано, если приводит к существенному росту усредненного по всем устройствам коэффициента загрузки оборудования при решении задач. Это непосредственно сказывается на времени решения. Сегодня говорят не о специальном классе задач, а о задачах, ориентирующих ВС на универсальность, что обусловлено современными областями применения.

Важным революционизирующим моментом стал переход на микропроцессорную элементно-конструкторскую базу, обусловившую построение мультимикропроцессорных ВС.

Сложилось представление о двух основных уровнях, на которых в ВС применяются практические методы распараллеливания:

- на уровне программ, процессов, процедур (первый уровень распараллеливания);
- на уровне команд и операций (второй уровень распараллеливания).

Эти уровни обусловили уровни структуризации ВС на пути превращения ее в супер-ЭВМ. Современным практическим воплощением первого уровня структуризации являются однородные многопроцессорные ВС на общей (разделяемой) оперативной памяти. Они получили название симметричных ВС за обеспечение "равноправия" составляющих модулей. Окончательное признание симметричных ВС положило конец поиску "экзотических" архитектур, эффективных лишь при решении определенных классов задач. Универсальность симметричных ВС, возможность реализации на них

любых вычислительных процессов с высокой эффективностью оборудования иллюстрируются многими применениями и анализируются ниже.

Уровень команд и операций наиболее ярко представлен многофункциональными АЛУ и их обобщением — решающими полями, представляющими разделяемый вычислительный ресурс многих процессоров. Некоторые современные проекты ВС в разной степени предполагают такой ресурс. Здесь основная проблема — полная загрузка отдельных исполнительных устройств (ИУ) в процессе выполнения программы.

Различают два способа реализации такой загрузки: динамический и статический.

Динамическая загрузка осуществляется аппаратурой в процессе выполнения программы. Она использует упрощенные алгоритмы распараллеливания.

Статическая загрузка предусматривается при трансляции программы. Транслятор оптимизирует использование оборудования, также решая задачи распараллеливания. Это выражается в формировании "длинных" командных слов, задающих работы устройствам АЛУ в каждом машинном такте.

Основная сложность распараллеливания заключается в соблюдении частичной упорядоченности распределяемых работ. Поэтому решение задач синхронизации параллельного вычислительного процесса сопровождает все усилия по организации совместной работы устройств. Это сказывается и при решении всех задач эффективного использования расслоенной многоуровневой памяти ВС.

Отечественный опыт создания семейства МВК (многопроцессорных вычислительных комплексов) "Эльбрус", модели которого относятся к симметричным ВС, и последующее проектирование позволили проанализировать, разработать и применить ряд существенно новых, важных и перспективных решений в распараллеливании как самого вычислительного процесса, так и работы отдельных устройств. Разработка пронизана такими решениями, они ложатся в основу проектирования развития семейства, являются основой обобщений и дальнейшего исследования возможности применения.

Классификация параллельных ВС

Потоки команд и потоки данных. Общепринята удачная классификация ВС, которую предложил в 1966 г. М.Флинн (США). Основным определяющим архитектурным параметром он выбрал взаимодействие потока команд и потока данных (операндов и результатов).

В ЭВМ классической архитектуры ведется последовательная обработка команд и данных. Команды поступают одна за другой (за исключением точек ветвления программы), и для них из ОЗУ или регистров так же последовательно поступают операнды. Одной команде (операции) соответствует один необходимый ей набор операндов (как правило, два для бинарных операций). Этот тип архитектуры — " один поток команд — один поток данных", ОКОД (SISD - "Single Instruction, Single Data") условно изображен на рис. 1.1.

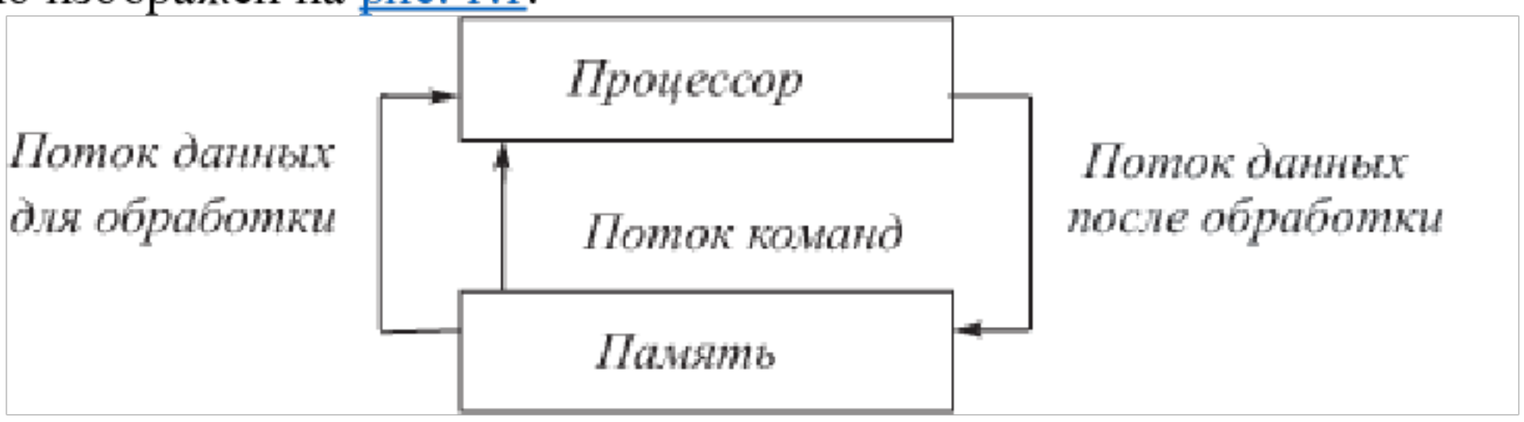


Рис. 1.1. ВС типа ОКОД (SISD)

Тип ОКМД — "один поток команд — много потоков данных" (SIMD — "Single Instruction — Multiple Data") охватывает ВС, в которых одной командой обрабатывается набор данных, множество данных, вектор, и вырабатывается множество результатов. Это векторные и матричные системы, в которых по одной команде выполняется одна и та же операция над всеми элементами массива — вектора или матрицы, распределенными

между процессорными (обрабатывающими) элементами ПЭ или процессорами. Принцип обработки показан на рис. 1.2.

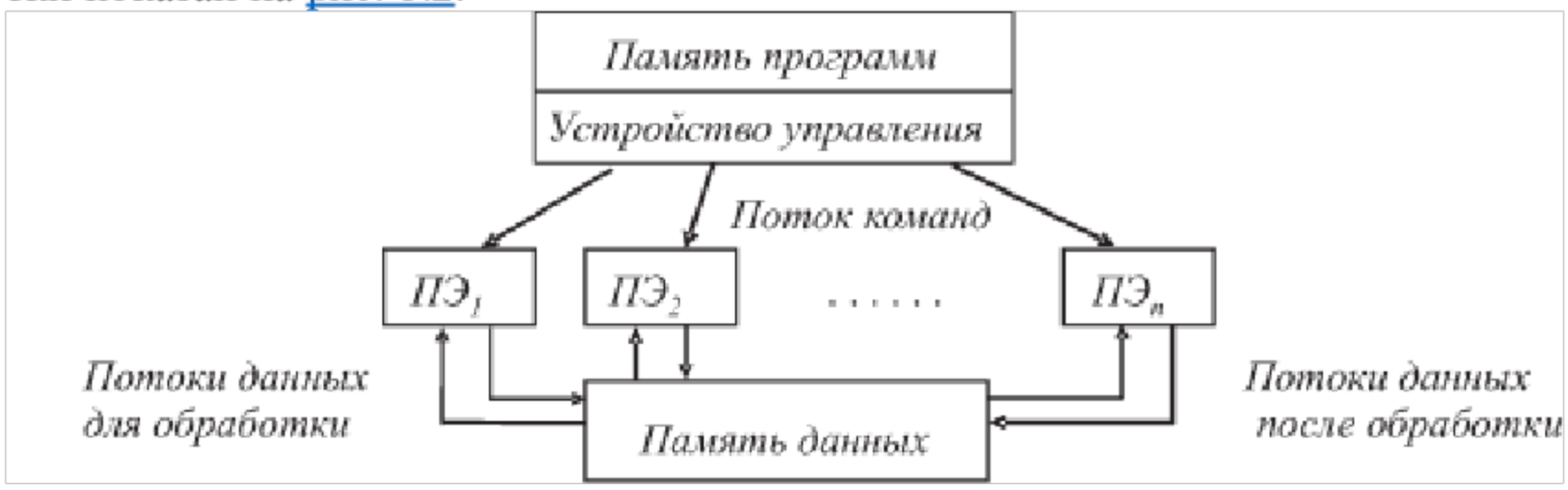


Рис. 1.2. ВС типа ОКМД (SIMD)

Отечественные векторные ВС — ПС-2000, ПС-2100. Допускают организацию матричной обработки. Классический пример матричной архитектуры — ILLIAC-1V (США).

К типу МКОД — "много потоков команд — один поток данных" (MISD — "Multiple Instruction — Single Data") принято относить векторный конвейер (обычно в составе ВС, чтобы подчеркнуть основной используемый принцип вычислений), например, в составе ВС Crey -1, "Электроника ССБИС". На векторном конвейере производится последовательная обработка одного потока данных многими обрабатывающими устройствами (ступенями, станциями) конвейера.

К такому же типу относится ВС, реализующая макроконвейер (ВС "Украина"). В ней задача, решаемая циклически, "разрезается" на последовательные этапы, закрепляемые за отдельными процессорами. Запускается конвейер многократного выполнения цикла, составляющего задачу.

Принцип обработки показан на рис. 1.3.

Рис. 1.3. ВС типа МКОД (MISD)

Тип МКМД — "много потоков команд — много потоков данных" (MIMD — "Multiple Instruction — Multiple Data") соответствует более полному и независимому распараллеливанию. К этому типу относятся, например, все многопроцессорные вычислительные комплексы (МВК) семейства "Эльбрус".

"Фон-Неймановские" и "не-Фон-Неймановские" архитектуры

Немного истории. Первую ЭВМ создал в 1939 г. в США профессор Джон Атанасов, болгарин, со своим аспирантом К.Берри. Две малые ЭВМ, созданные ими в период 1937 — 1942 гг., были прототипами большой ЭВМ АВС для решения систем линейных уравнений, которая в 1942 г. доводилась по устройствам ввода-вывода и должна была войти в строй в 1943 г., но призыв Атанасова в армию в 1942 г. воспрепятствовал этому. Проект электронной ЭВМ Эниак (Electronics Numerical Integrator and Computer) был сделан в 1942 г. Д.Моучли и Д.Эккертом и осуществлен в 1945 г. в Муровской электротехнической лаборатории Пенсильванского университета. В 1946 г. Эниак был публично продемонстрирован в работе. В нем впервые были применены

триггеры. Рождение Эниак считают началом компьютерной эры, посвящая ему научные симпозиумы и другие торжественные мероприятия. (Международный симпозиум, посвященный 50-летию первой ЭВМ, был проведен и в Москве в июне 1996 г.)

Однако еще в начале 40-х годов XX века Атанасов поделился с Моучли информацией о принципах, заложенных в ЭВМ ABC. Хотя Моучли впоследствии утверждал, что он не воспользовался этой информацией в патенте на Эниак, суд не согласился с этим. Вернувшись из армии после войны, Атанасов узнал, что более мощная ЭВМ Эниак уже создана, и потерял интерес к этой теме, не поинтересовавшись, насколько Эниак похож на его ЭВМ ABC.

Известный английский математик Алан Тьюринг был не только теоретиком по информации и теории алгоритмов, автором теоретического автомата "машины Тьюринга", но и талантливым инженером, создавшим в начале 1940-х годов первую работающую специализированную ЭВМ. Эта ЭВМ под названием "Колосс" была сконструирована и сделана им совместно с Х.А.Ньюменом в Блетчи (Англия) и начала работать в 1943 г. Сообщения о ней своевременно не публиковались, т.к. она использовалась для расшифровки секретных германских кодов во время войны.

Основные архитектурно-функциональные принципы построения ЭВМ были разработаны и опубликованы в 1946 г. венгерским математиком и физиком Джоном фон Нейманом и его коллегами Г.Голдстайном и А.Берксом в ставшем классическим отчете "Предварительное обсуждение логического конструирования электронного вычислительного устройства". Основополагающими принципами ЭВМ на основании этого отчета являются: 1) принцип программного управления выполнением программы, и 2) принцип хранимой в памяти программы. Они легли в основу понятия фон-Неймановской архитектуры, широко использующей счетчик команд .

Вернемся к настоящему. Счетчик команд отражает "узкое горло", которое ограничивает поток команд, поступающих на исполнение, их последовательным анализом.

Альтернативной архитектурой является "не-фон-Неймановская" архитектура, допускающая одновременный анализ более одной команды. Поиски ее обусловлены необходимостью распараллеливания выполнения программы между несколькими исполнительными устройствами — процессорами. Счетчик команд при этом не нужен. Порядок выполнения команд определяется наличием исходной информации для выполнения каждой из них. Если несколько команд готовы к выполнению, то принципиально возможно их назначение для выполнения таким же количеством свободных процессоров. Говорят, что такие ВС управляются потоком данных (data flow).

Общая схема потоковых ВС представлена на рис. 1.4.

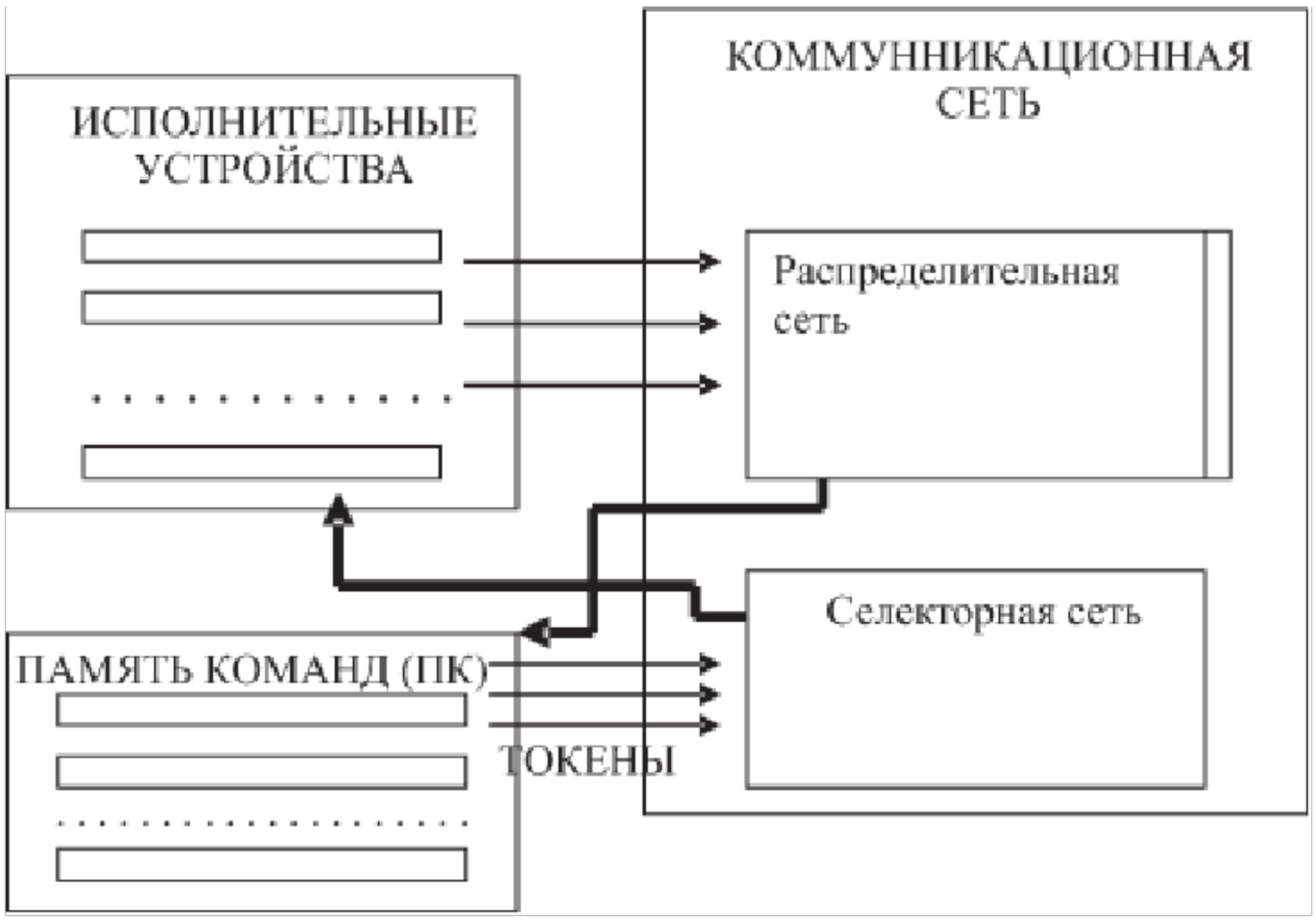


Рис. 1.4. "Идеальная" потоковая ВС

Программа или ее часть (сегмент) размещается в памяти команд ПК, состоящей из ячеек команд. Команды имеют структуру

{код операции, операнд 1, ..., операнд L,

адрес результата 1, ..., адрес результата M}

В командах проверки условия возможно альтернативное задание адреса результата (ИЛИ — ИЛИ). Адреса результатов являются адресами ПК, т.е. результаты выполнения одних команд в качестве операндов могут поступать в текст других команд. Команда не готова к выполнению, если в ее тексте отсутствует хотя бы один операнд. Ячейка, обладающая полным набором операндов, переходит в возбужденное состояние и передает в селекторную сеть информационный пакет (токен), содержащий код операции и необходимую числовую и связную информацию. Он поступает по сети на одно из исполнительных устройств. Там же операция выполняется, и в распределительную сеть выдается результирующий пакет, содержащий результат вычислений и адреса назначения в ПК (возможно, за счет выбора альтернативы, т.е. такой выбор — тоже результат). По этим адресам в ПК результат и поступает, создавая возможность активизации новых ячеек. После выдачи токена в селекторную сеть операнды в тексте команды уничтожаются, что обеспечивает повторное выполнение команды в цикле, если это необходимо.

Селекторная и распределительная сети образуют коммуникационную сеть ВС.

Ожидаемая сверхвысокая производительность такой системы может быть достигнута за счет одновременной и независимой активизации большого числа готовых команд, проблематичном допущении о бесконфликтной передаче пакетов по сетям и параллельной работы многих исполнительных устройств.

Существует ряд трудностей, в силу которых "не-фон-Неймановские" архитектуры не обрели технического воплощения для массового применения в "классическом", отраженном выше, исполнении. Однако многие устройства используют данный принцип, но чаще всего взаимодействие процессоров при совместном решении общей задачи и их синхронизация при использовании общих данных основаны на анализе готовности данных для их обработки. Это дает основание многим конструкторам заявлять, что в своих моделях они реализовали принцип data flow.

Системы с общей и распределенной памятью

Системы с общей (разделяемой) оперативной памятью образуют современный класс ВС — многопроцессорных супер-ЭВМ. Одинаковый доступ всех процессоров к программам и данным представляет широкие возможности организации параллельного вычислительного процесса (параллельных вычислений). Отсутствуют потери реальной производительности на межпроцессорный (между задачами, процессами и т.д.) обмен данными (рис. 1.5а).

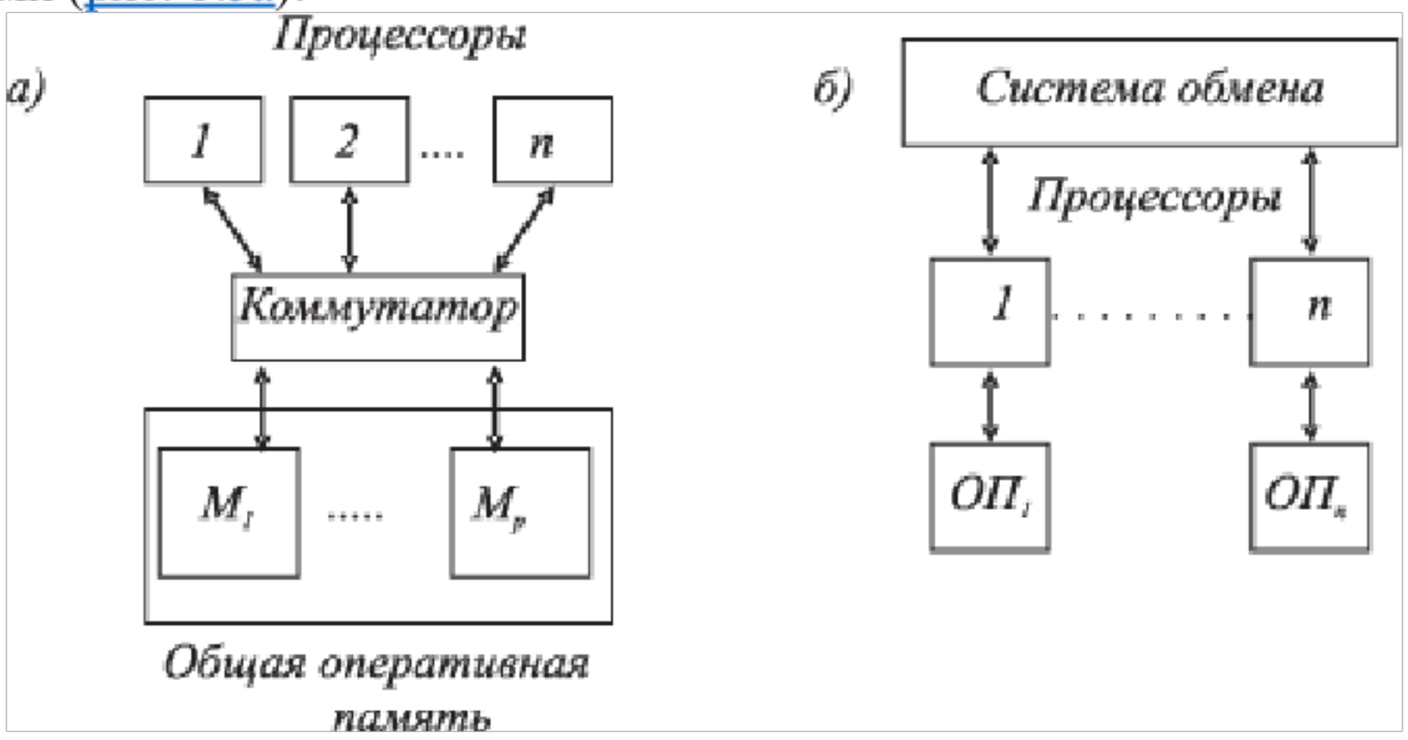


Рис. 1.5. ВС с общей (а) и распределённой (б) памятью

Системы с распределенной памятью образуют вычислительные комплексы (ВК) — коллективы ЭВМ с межмашинным обменом для совместного решения задач (рис. 1.5б). В ВК объединяются вычислительные средства систем управления, решающие специальные наборы задач, взаимосвязанных по данным. Принято говорить, что такие ВК выполняют распределенные вычисления, а сами ВК называют распределенными ВК.

Другое, противоположное воплощение принципа МИМД — масспроцессорные или высокопараллельные архитектуры, объединяющие сотни — тысячи — десятки тысяч процессоров.

В современных супер-ЭВМ наметилась тенденция объединения двух принципов: общей (распределяемой) и распределенной (локальной) оперативной памяти (ЛОП). Такая структура используется в проекте МВК "Эльбрус-3" и "Эльбрус-3М" (рис. 1.6).

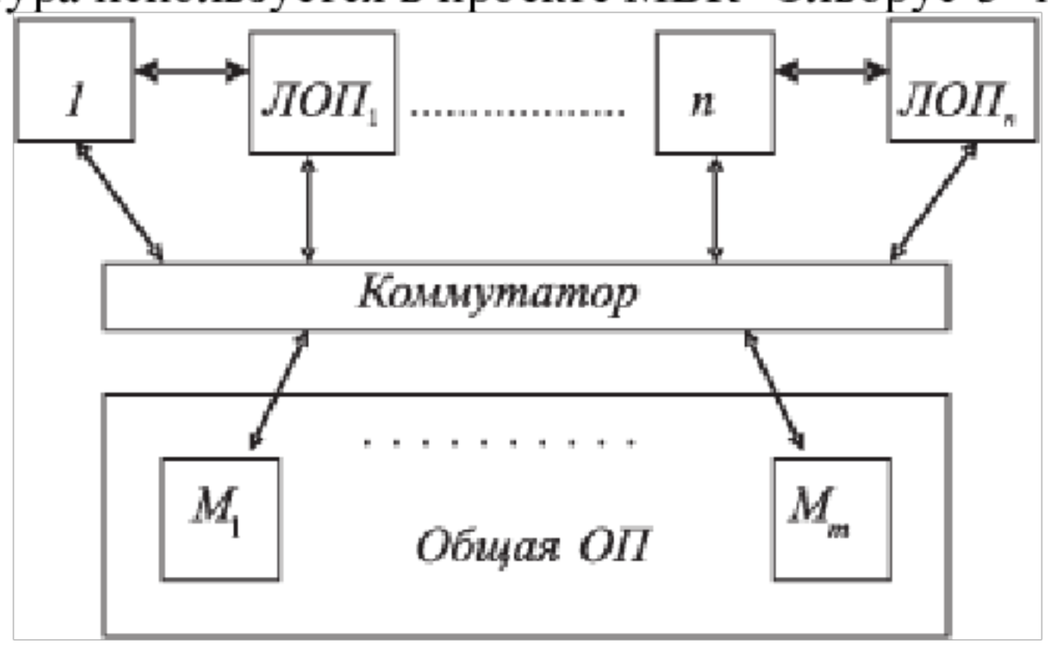


Рис. 1.6. Схема ВС с модулями локальной памяти

Способы межмодульного соединения (комплексирования)

Различают два противоположных способа комплексирования: с общей шиной (шинная архитектура) и с перекрестной (матричной) коммутацией модулей ВС (процессоров, модулей памяти, периферии).

На рис. 1.7 представлена система с общей шиной. Шина состоит из линий, по которым передаются информационные и управляющие сигналы.

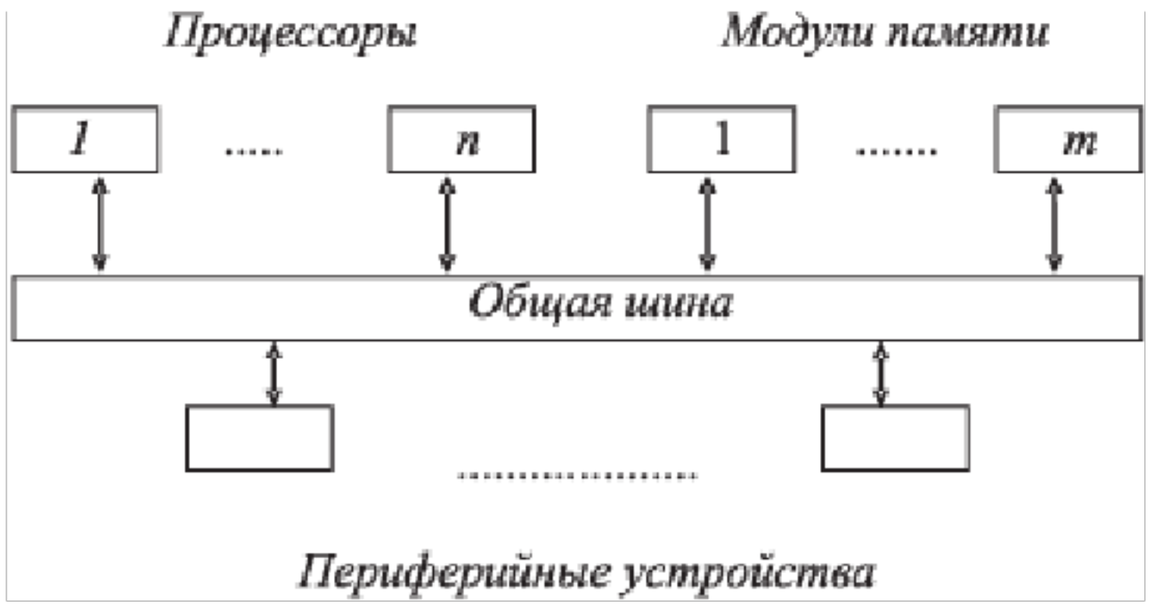


Рис. 1.7. Схема ВС с общей шиной

Шина используется в режиме разделения времени, при котором лишь один модуль в данный момент работает на передачу. Принимать принципиально могут все модули, хотя преимущественно информация при выдаче в нее адресуется. Применяется в микро- и мини-ЭВМ при сравнительно небольшом числе модулей. Практически производится разделение шины на управляющую, адресную и шину данных.

В высокопроизводительных ВС для возможности одновременного обмена многими парами абонентов используется перекрестная или матричная коммутация.

Матричный коммутатор можно представить (прямоугольной) сеткой шин. К одному концу каждой подсоединен источник-потребитель информации (рис. 1.8). Точки пересечения — узлы этой сетки — представляют собой управляющие ключи, которые соединяют или разъединяют соответствующие шины, устанавливая или прекращая связь между модулями. Реализуется связь "каждый с каждым". Одновременно могут связываться многие (до n/2) пары модулей.

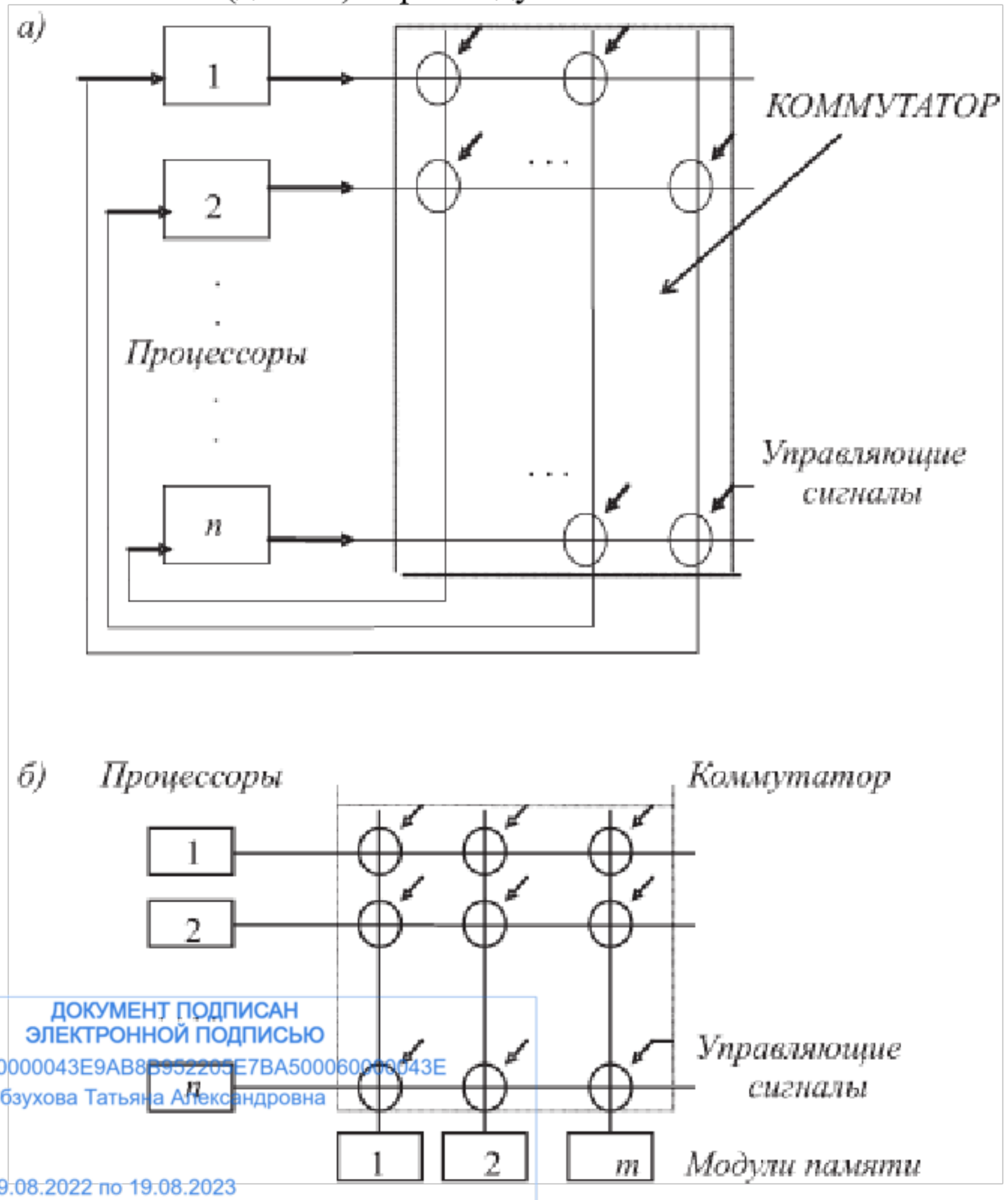


Рис. 1.8. Матричные коммутаторы: а) — перекрёстная коммутация процессоров, б) — коммутация процессоров и модулей памяти

На рис. 1.8а — перекрестная связь между процессорами в ВС с распределенной памятью, на рис. 1.8б — между n процессорами и m модулями ОП.

"Исторические" модели

Векторная ВС ПС-2000

Разрабатывалась как проблемно-ориентированная ВС для задач обработки геофизической информации, информации со спутников в интересах геологии, картографии, обработки изображений, моделирования поведения среды и т.д. Является типичной иллюстрацией типа ОКМД.

Схема ВС приведена на рис. 1.9.

Рис. 1.9. Схема ВС ПС-2000

Основу ПС-2000 составляет параллельный процессор с общим потоком команд ППС-2000, содержащий до 64 ПЭ.

Отмечая типичные архитектурные особенности моделей, иллюстрируя их на архитектурах ВС, ставших классическими, уделим второстепенное внимание быстро устаревающим характеристикам этих ВС. Однако отметим, что ПЭ ППС-2000 — 24-разрядный микропроцессор с производительностью 3 млн оп./с, а это было значительным достижением в начале 1980-х годов.

Соседние ПЭ связаны регулярными каналами РК для оперативного обмена данными. В связи с проблемной ориентированностью ВС это обеспечивает быстрый обмен при обработке "соседними" ПЭ "соседних" же элементов поверхностей, конечно же, информационно взаимосвязанных. Связь с помощью РК — циклическая, в частности, ПЭ1 с помощью РК связан с ПЭ64. Т.е. при "прокатывании" 64 ПЭ по длинной последовательности обрабатываемых элементов поверхности ПЭ1 и ПЭ64 являются "соседними": после элемента поверхности, обрабатываемого ПЭ64, следует элемент поверхности, обрабатываемый ПЭ1. Затем пошли дальше, превратив векторную ППС-2000 в матричную ВС.

Ввели сегментирование множества процессоров (процессоров решающего поля). 64 ПЭ разбиваются или на 8 сегментов по 8 ПЭ, или на 4 сегмента по 16 ПЭ, или на 2 сегмента по 32 ПЭ. Это осуществляется, прежде всего, за счет установления связи с помощью РК между первым и последним ПЭ каждого сегмента. Тогда при обработке поверхности ПЭ "прокатываются" циклически не только по "длинному" вектору, но и, при сегментировании, по матричным элементам поверхности, как будет показано далее.

Для общей связи всех элементов ППС-2000, включая УУ, используется магистральный канал — общая шина. По магистрали микрокоманд каждому ПЭ сообщаются команды, микропрограммы операций и микрокоманды из УУ.

ППС-2000 имеет распределенную память.

В ППС-2000 предусмотрена двухуровневая схема управления: на уровне микрокоманд и команд. Память для их хранения — в УУ. На микрокомандном уровне пишутся отдельные фрагменты задачи — подпрограммы процедур, на командном — последовательность обращения к этим подпрограммам.

Основным, задающим, является вычислительный процесс в мониторной подсистеме МПС. В ней в режиме разделения времени могут выполняться несколько задач. С ППС-2000 в данный момент может быть связана только одна задача. Т.е. ППС-2000 по отношению к малой машине (типа СМ-2), выполняющей функции МПС, является интеллектуальным терминалом с последовательным доступом.

В обычном режиме набор процедур, выполняемых в ППС-2000, по запросам со стороны МПС, определяется заранее. Микропрограммы, реализующие этот набор процедур, загружаются в память ППС-2000 заранее. Если в ходе вычислительного процесса в МПС возникает необходимость в какой-либо процедуре из этого набора, задача запускает соответствующую микропрограмму и передает ей необходимые данные.

Возможна и динамическая загрузка микропрограмм в память ППС-2000.

Тема 3. Векторные, матричные и ассоциативные ВС.

Тема самостоятельного изучения. Матричные вычислительные системы
Вид деятельности студентов: самостоятельное изучение учебно-методической литературы
Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: собеседование
План конспекта:
Матричные вычислительные системы. Структура матричных ВС. Архитектура матричных ВС. Примеры матричных ВС.
Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-2	1-2	1-2	1-5

Матричные ВС

Матричная ВС также является типичным представителем ОКМД, расширением принципа векторных ВС.

Классическим примером, прототипом и эталоном стала ВС ILLIAC 1V, разработанная в 1971 г. в Иллинойском университете и в начале 1972 г. установленная в Эймском научно-исследовательском центре NASA (Калифорния).

Одна последовательность команд программы управляет работой множества 64 ПЭ, одновременно выполняющих одну и ту же операцию над данными, которые могут быть различными и хранятся в ОП каждого ПЭ. (Производительность — до 200 млн. оп./с)

В целом структура центральной части может быть представлена аналогичной векторной ВС (рис. 1.10).

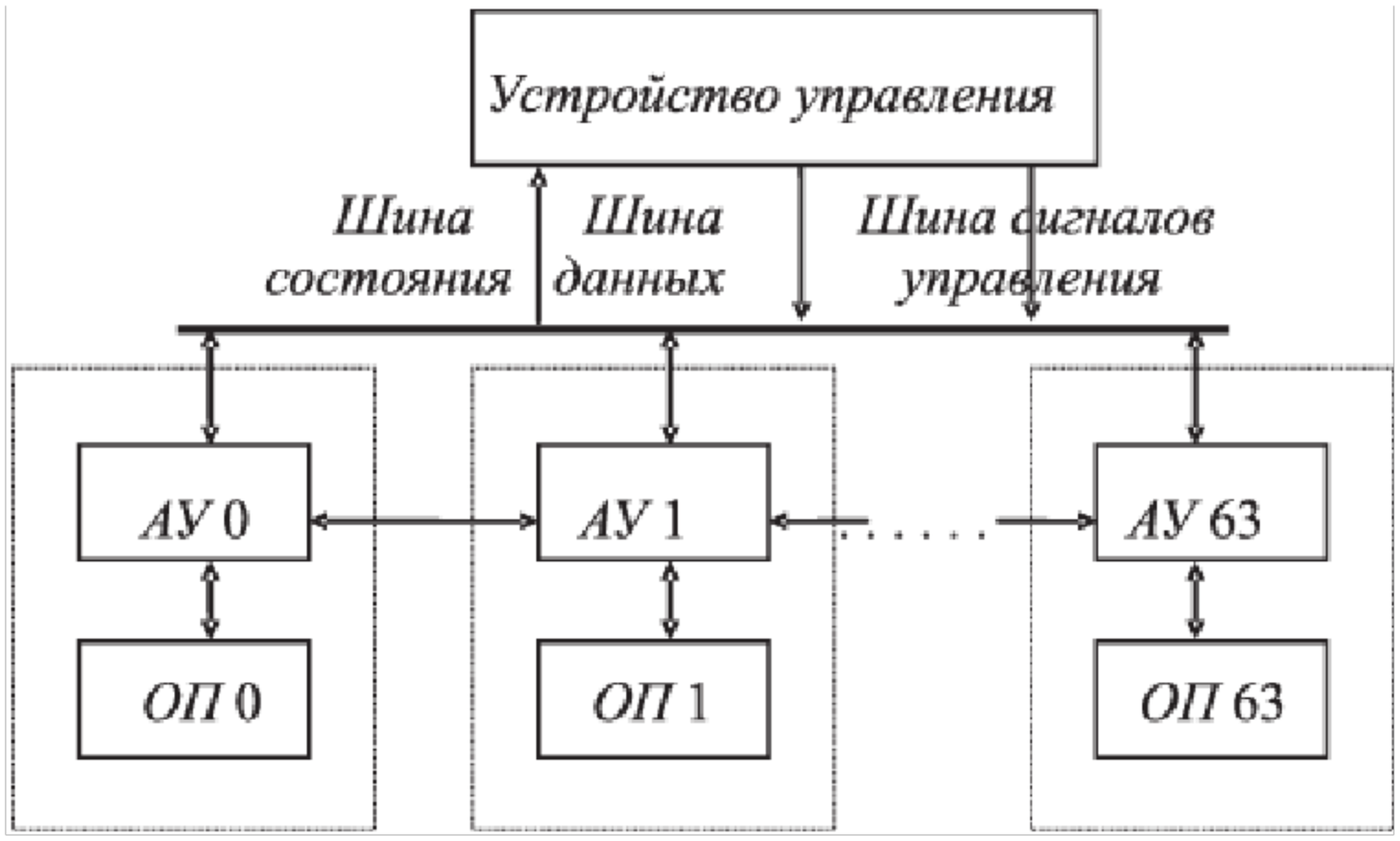


Рис. 1.10. Схема обмена в матричной ВС

УУ — фактически простая вычислительная машина небольшой производительности и может выполнять операции над скалярами одновременно с выполнением матрицей ПЭ операций над векторами или матрицами. Посылает команды в независимые ПЭ и передает адреса в их ОП.

К центральной части подключена система ввода-вывода с управляющей машиной В-6500, устройство управления вводом-выводом, файловые диски, буферная память и коммутатор ввода-вывода. Операционная система, ассемблеры и трансляторы размещаются в памяти В-6500.

Почему же ВС — матричная?

ПЭ образуют матрицу, в которой информация от одного ПЭ к другому может быть передана через сеть пересылок данных при помощи специальных команд обмена. Регистры пересылок (рис. 1.11) каждого i -го ПЭ связаны высокоскоростными линиями обмена с регистрами пересылок ближайшего левого (i-1 -го) и ближайшего правого (i+1)-го ПЭ, а также с регистрами пересылок ПЭ, отстоящего влево на 8 позиций от данного (i-8)-й и отстоящего вправо на 8 позиций от данного (i+8)-й, при этом нумерация ПЭ рассматривается как циклическая с переходом от 63 к 0 (нумерация возрастает по mod 64) слева направо и от 0 к 63 справа налево.

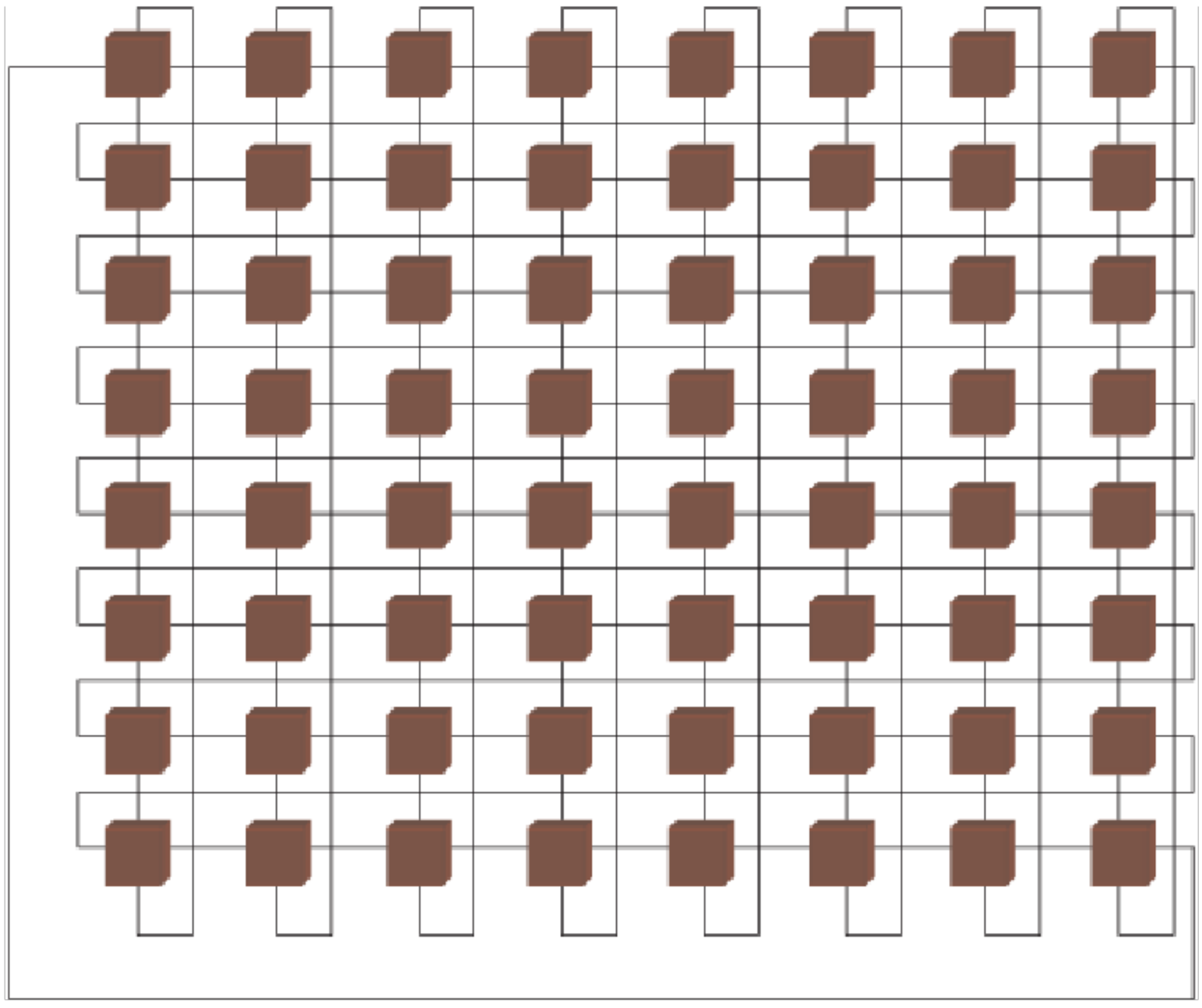


Рис. 1.11. Конфигурация "быстрых" связей в ВС ILLIAC-4

Расстояния между ПЭ в сети пересылок и соответствующие пути передачи информации задаются комбинациями из (-1, +1, -8, +8).

Предусмотрено маскирование ПЭ. Каждый ПЭ может индексировать свою ОП независимо от других ПЭ, что важно для операций матричной алгебры, когда к двухмерному массиву данных требуется доступ как по строкам, так и по столбцам.

Доступ к ОП имеют АУ соответствующего ПЭ, УУ системы и подсистема ввода-вывода. Управление доступом и разрешение конфликтов при доступе к ОП осуществляет УУ системы.

Система эффективна при решении задач большой размерности, использующих исчисление конечных разностей, матричной арифметики, быстрого преобразования Фурье, обработки сигналов и изображений, линейного программирования и др.

Например, приближенные методы решения задач математической физики и, прежде всего, — системы дифференциальных уравнений используют исчисление конечных разностей (конечно-разностные методы решения, "метод сеток").

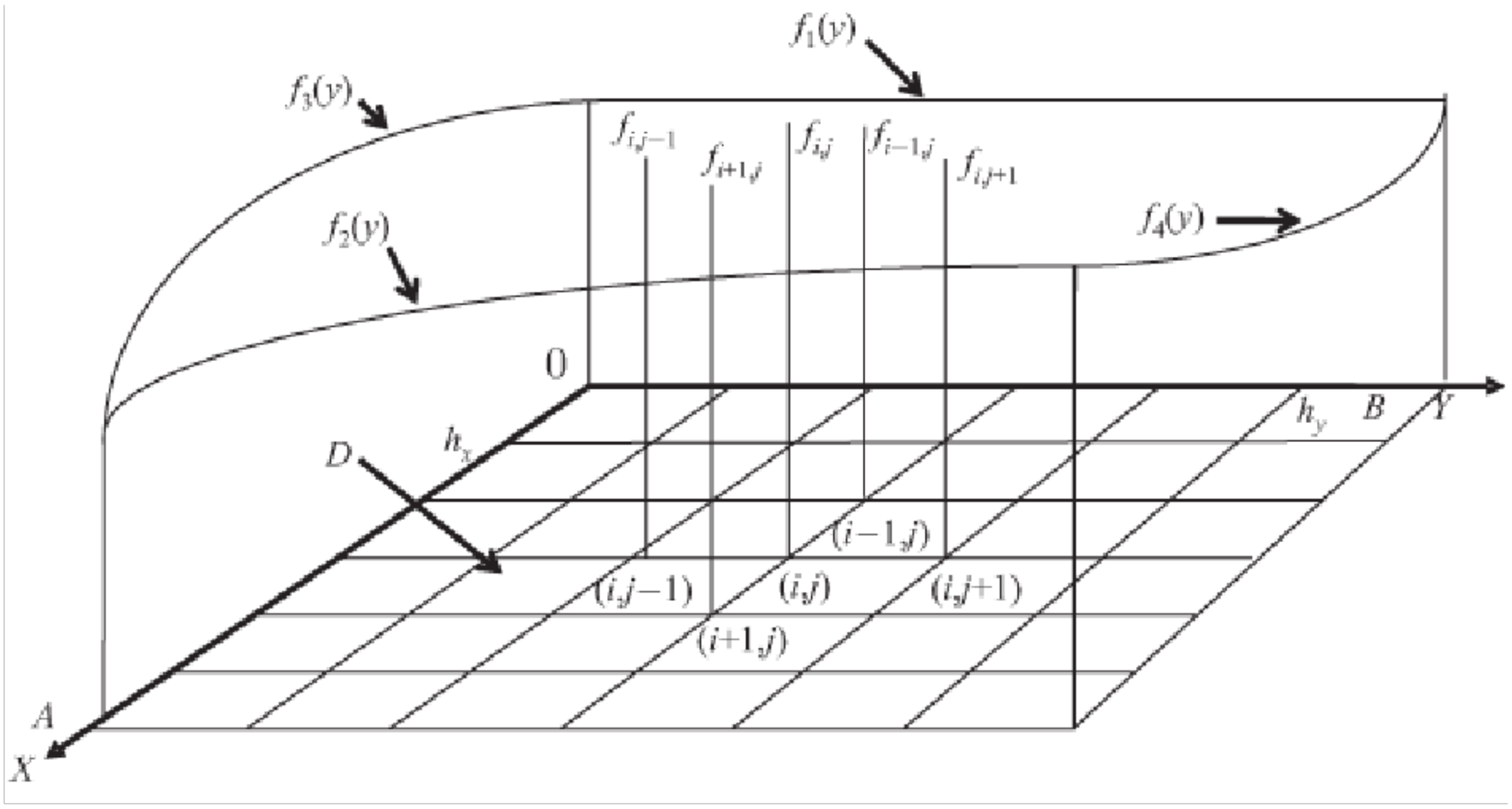


Рис. 1.12. Решение задачи методом "сеток"

Кратко рассмотрим план решения подобной задачи на матричной ВС.

Пусть необходимо решить уравнение с частных производных

$$\frac{\partial^2 f}{\partial x^2} - \frac{\partial f}{\partial y} = 0$$

на области В:

$$\begin{cases} 0 \leqslant x \leqslant A \\ 0 \leqslant y \leqslant B \end{cases}$$

Граничные условия:

f(0,y)= f1(y),

f(A,y)= f2(y),

f(x,0)= f3(x),

f(x,B)= f4(x).

Частные производные можно выразить через конечные разности несколькими способами, например:

$$\left.\frac{\partial f}{\partial x}\right|_{i,j} \cong \frac{f_{ij} - f_{i-1,j}}{h_x} \cong \frac{f_{i+1,j} - f_{ij}}{h_x};$$

$$\left.\frac{\partial^2 f}{\partial x^2}\right|_{i,j} \cong \frac{(f_{i+1,j} - f_{ij}) - (f_{ij} - f_{i-1,j})}{h_x^2} = \frac{f_{i+1,j} - 2f_{ij} + f_{i-1,j}}{h_x^2};$$

$$\left.\frac{\partial f}{\partial y}\right|_{i,j} \cong \frac{f_{i,j+1} - f_{i,j-1}}{2h_y}.$$

Подставив полученные выражения в уравнение, получим основное рекуррентное соотношение для нахождения значения функции через ее значения в соседних четырех узлах:

fi, j = 0,5 fi+1, j + 0,5 fi-1, j - 0,25 hx2/hy(fi, j+1 - fi, j-1).

Распараллеливание вычислений и "прокатывание" матрицей ПЭ области В — многократные, до получения необходимой точности — очевидны.

А именно: первоначально в одном цикле итерации 64 ПЭ по выведенной формуле рассчитывают 8 х 8 узловых значений функции близ вершины координат, "цепляя" при этом граничные значения. Затем производится смещение вправо на следующую группу узловых значений функции. По окончании обработки по горизонтали производится смещение по вертикали и т.д. При следующей итерации процесс повторяется. От итерации к итерации распространяется влияние граничных условий на рассчитываемые значения функции. Процесс сходится для определения значений функции-решения в узлах сетки с требуемой точностью.

Вернемся к отечественной разработке ПС-2000 (ПС-2100) — векторной ВС. В ней предусмотрена возможность сегментации процессоров, при которой последний процессор в сегменте связывается регулярным каналом (аналог высокоскоростной линии) с первым. Все 64 процессора могут составить один сегмент, могут быть сформированы два сегмента по 32 ПЭ, четыре по 16, 8 по 8 ПЭ. Итак, получается идеальная схема для "покрытия" двумерной подобласти обсчитываемой области. Правда, отсутствуют быстрые связи по вертикали, что должно быть возмещено хранением всей информации по столбцам в ОП каждого ПЭ. Зато, при смещении процессоров на смежную подобласть по горизонтали, последний ПЭ предыдущей подобласти остается "левым" для первого в новой подобласти. Т.е. при таком смещении сохраняется регулярность фактических связей между узлами сетки.

В ILLIAC-1V такая регулярность сохранялась бы, если бы существовали быстрые связи между ПЭ7 и ПЭ0, ПЭ15 и ПЭ8 и т.д., то есть циклически по строкам матрицы.

Как видим, такие связи учтены при более поздних разработках матричных ВС.

ВС Крей-1 ("Электроника ССБИС")

ВС Cray-1, несмотря на последующие разработки, остается эталоном типа МКОД. В России разработан аналог — векторно-конвейерная ЭВМ "Электроника ССБИС", отличающаяся некоторыми увеличенными параметрами комплектации. Структура ВС показана на рис. 1.13.

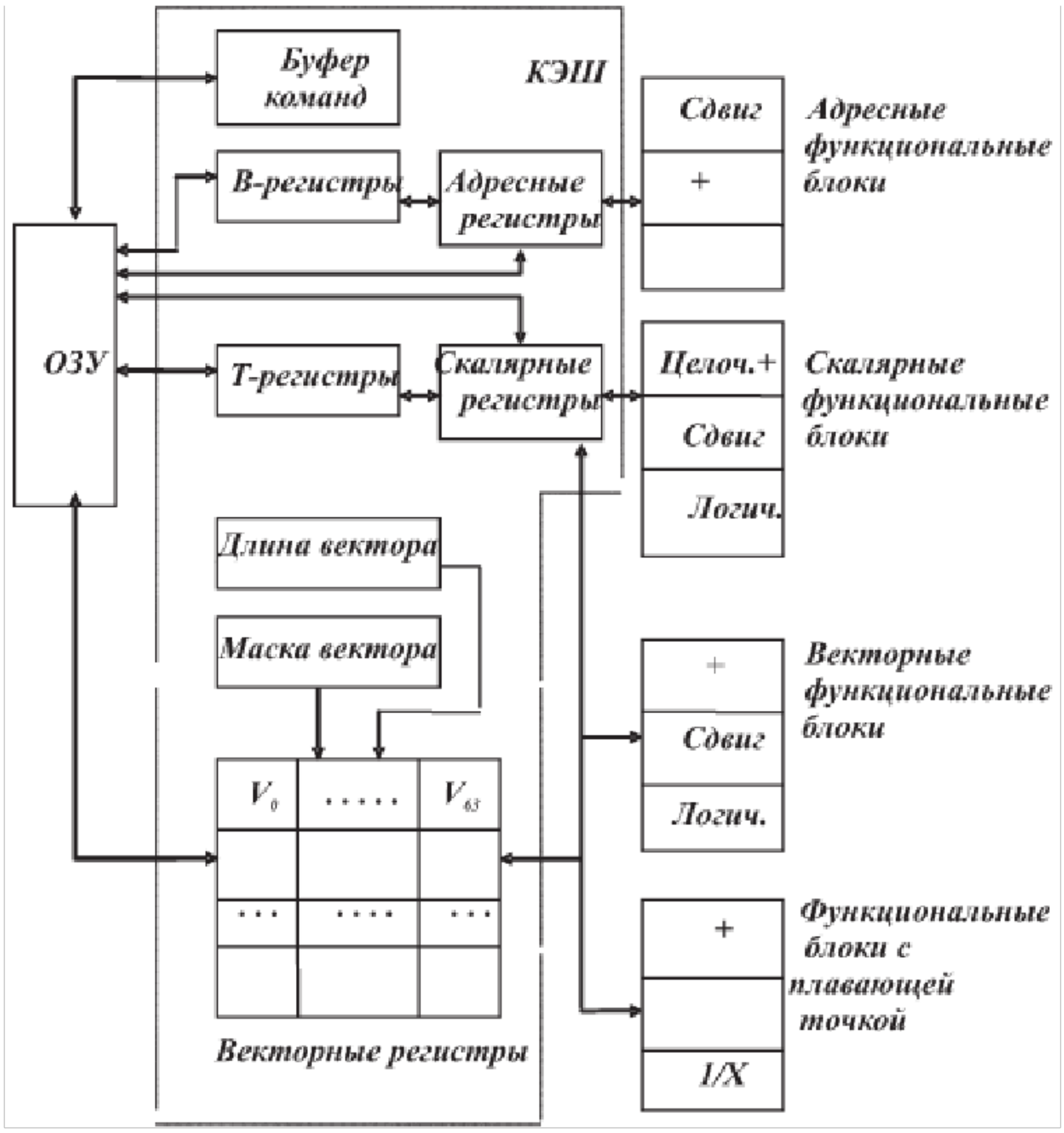


Рис. 1.13. Схема ВС Cray-1

Предназначена для выполнения как векторных, так и скалярных операций, Имеет 64 буферных (выравнивающих) регистра для 24 -битовых адресов (В -регистры), 64 буферных (выравнивающих) регистров для 64 -битовых слов данных (Т -регистры). В -регистры в свою очередь пополняют буфер для одной из основных групп регистров машины — 8 адресных регистров, а Т -регистры — для группы 8 скалярных регистров. Адресные регистры используются и как индексные. Третья основная группа регистров — 8 векторных регистров по 64 64 -битовых слов. Регистр длины вектора определяет число операций, выполняемых по векторным командам, т.е. действительную длину вектора. Маска вектора определяет элементы, над которыми выполняется операция.

Буфер команд состоит из четырех ЗУ, каждое из которых — из 64 16 -разрядных регистров. Представляет 4 программы, выполняемые в мультипрограммном режиме. (В "Э-ССБИС" — 16 таких ЗУ.)

12 специализированных функциональных блоков-конвейеров (в "Э-ССБИС" 16 блоков) выполняют арифметические, логические операции и сдвиг. Уровней конвейера — от 2 до 14. Блоки являются независимыми. Несколько функциональных блоков могут работать одновременно. Возможно "зацепление" векторов при выполнении операций вида A x B + C, когда два или более конвейеров выстраиваются в один и результат с одного немедленно поступает как операнд на другой.

Таким образом, параллелизм в обработке данных организуется с помощью:

1. одновременного использования различных функциональных блоков и различных векторных регистров;
2. применения режима "зацепления" векторов.

(Нас интересует архитектура, но не катастрофически стареющие характеристики. Однако для справки: такт машины 12{,}5 нс, цикл обращения к ОЗУ 50 нс, производительность считается: 38 млн. скалярных оп/с и 80 млн. векторных оп/с.)

МВК "Эльбрус-2"

Представляет тип МКМД. Имеет общую оперативную память и перекрестный коммутатор. Является основой сложных автоматизированных систем управления, работающих в реальном времени, использовался в космических исследованиях, а также в области ядерной физики.

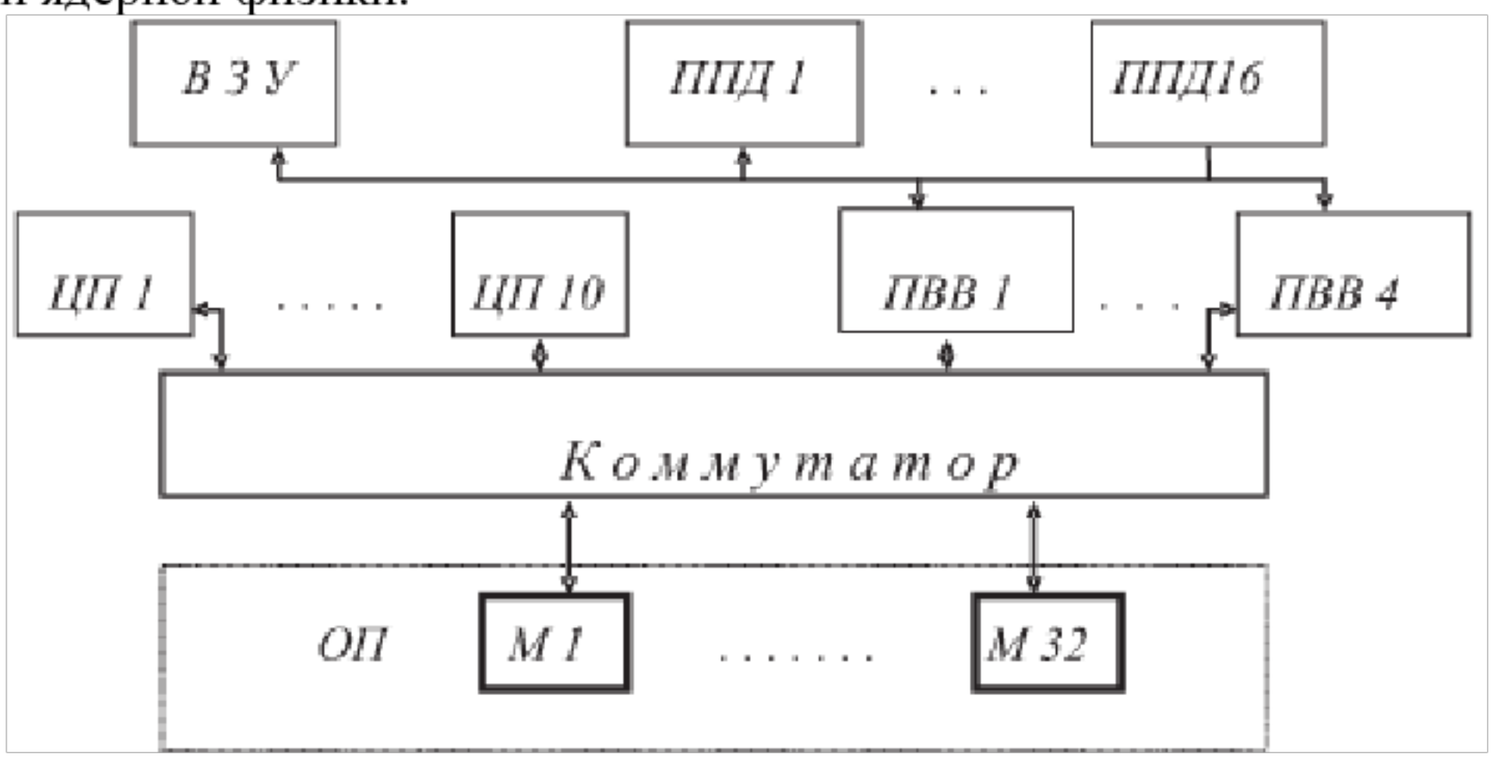


Рис. 1.14. Общая схема МВК "Эльбрус-2"

Производительность (информация имеет историческую ценность) в 10-процессорном варианте — до 120 млн. оп/с (12,5 млн оп/с на одном процессоре). Максимальный объем ОП — 16 млн 72 -разрядных слов (144 Мбайт), максимальная пропускная способность каналов ввода-вывода — 120 Мбайт/с. Эти характеристики позволили решить многие проблемы применения супер-ЭВМ, но сегодня не впечатляют.

Принципиальные предпосылки разработки:

1. Аппаратная поддержка ЯВУ.
2. Параллелизм на двух уровнях — мультипроцессорная обработка (распределение процессов между процессорами), конвейерная обработка команд и параллельная обработка данных в многофункциональном АЛУ (10 ИУ).

В МВК могут входить от 1 до 10 ЦП, от 1 до 32 модулей ОП и от 1 до 4 ПВВ.

Каждый ПВВ имеет 40 каналов связи: 8 быстрых (аналог селекторных) каналов (БК) для магнитных барабанов и дисков и 32 стандартных (СК) для прочих устройств ввода-вывода.

Работа с удаленными объектами по линиям связи осуществляется с помощью процессоров передачи данных (их — до 16), которые являются самостоятельными модулями с собственной системой команд и внутренней памятью.

Все компоненты системы работают параллельно и независимо друг от друга и динамически распределяются операционной системой между задачами.

ЦП МВК "Эльбрус" имеет безадресную систему команд, основанную на ПОЛИЗ и выполняемую на стеке. Обрабатывает числовые данные в 32 разряда (полслова), 64 разряда (слово), 128 разрядов (двойное слово). Нечисловая информация может быть представлена в виде битовых, цифровых или байтовых наборов. Каждое слово сопровождается 6 -разрядным тегом, определяющим его тип. Два разряда используются в аппаратном контроле.

Проект МВК "Эльбрус-3"

Так как ассемблерный язык исключен из интерфейса пользователя, то аппаратная реализация (она отражена в системе команд — в машинном языке) МВК "Эльбрус-3" значительно отличается от "Эльбрус-2". Структура МВК представлена на рис. 1.15.

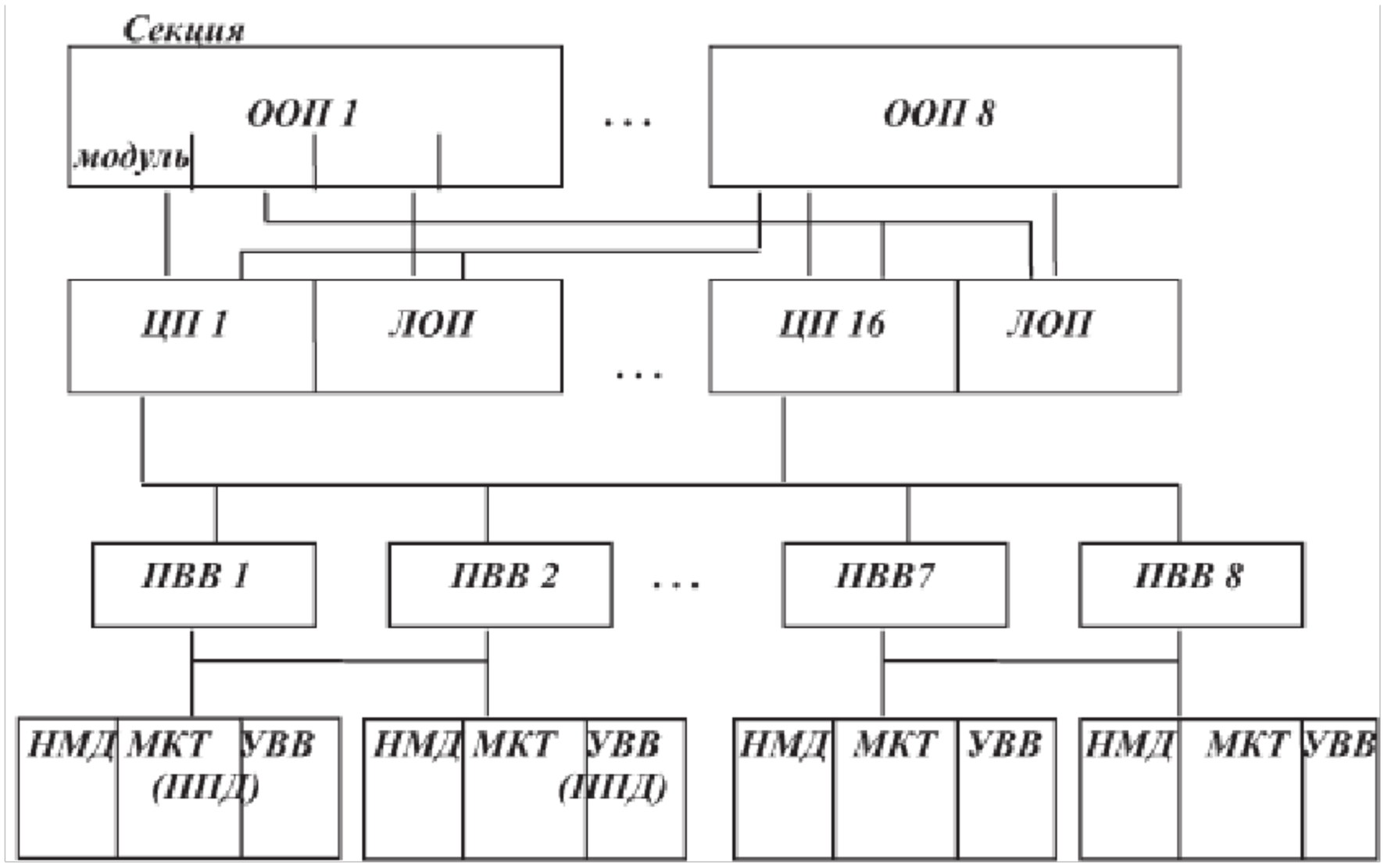


Рис. 1.15. Общая схема МВК "Эльбрус-3"

Здесь ООП — общая оперативная память (8 секций (блоков) по 4 модуля; модуль — 32 Мбайт); МКТ — модульные комплексы телеобработки (аналог ППД — в "Эльбрус-2"); ЛОП — локальная оперативная память, 4 — 16 Мбайт.

Принципы, лежащие в основе разработки:

1. Программная совместимость на уровне ЯВУ (Эль-76 — ЯВУ-ассемблер) с другими моделями "Эльбрус".
2. Теговая архитектура.
3. Дальнейшее развитие многопроцессорности на общей ОП, что повышает производительность и структурную надежность. Пиковая производительность одного (из 16) процессора достигает 700 млн. оп/с при такте машины 10 нс.
4. Архитектура с длинным командным словом обеспечивает микрораспараллеливание между 7 ИУ АЛУ на уровне операций (2 ИУ сложения, 2 — умножения, одно — деления, 2 — логических), а также параллельное обращение по 8 каналам в ЛОП или ОП.
5. Семь конвейерных и независимых ИУ обеспечивают загрузку процессора работами в каждый такт.
6. Управление машиной позволяет контролировать каждый такт. Т.е. "длинная" команда выдается каждый такт, непосредственно запуская устройства без дополнительного динамического аппаратного планирования их загрузки.
7. Оптимизация распределения ресурсов возлагается на высокооптимизирующий транслятор с ЯВУ Эль-76 и с других языков высокого уровня.
8. Функционально разделенное СОЗУ снижает конфликты при обращении и отражает типы хранимых данных. Оно состоит из ассоциативного буфера команд в 2 тыс. слов, буфера стека в 1 тыс. слов для хранения верхних уровней локальных данных выполняемых процедур, буфера массивов в 512 слов и ассоциативной памяти глобалов (общих данных процедур) в 512 слов.

ТЕМА 4. МНОГОПРОЦЕССОРНЫЕ И ОДНОРОДНЫЕ ВС.

Тема самостоятельного изучения. Многопроцессорные вычислительные системы с распределенной памятью

Вид деятельности студентов: самостоятельное изучение учебно-методической литературы
Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: собеседование
План конспекта:
Массивно-параллельные суперкомпьютеры серии Cry T3. Кластерные системы класса BEOWULF. Коммуникационные технологии, используемые при создании массово-параллельных суперкомпьютеров.
Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-2	1-2	1-2	1-5

Микропроцессорные системы и способы распараллеливания

Мультимикропроцессорные вычислительные системы

В настоящее время выбор сделан в пользу многопроцессорных симметричных ВС типа MIMD, обеспечивающих виртуализацию вычислительных ресурсов. Основу такой ВС составляет суперскалер, сосредоточивший в себе все способы достижения максимального быстродействия при выполнении одиночной программы. Векторные и векторно-конвейерные процессоры и системы получили своё место. Их эффективность как самостоятельных установок могла быть достаточно высокой только при решении специальных задач и тестов. Поэтому достаточно быстро выяснилось, что эти установки могут выполнять функции интеллектуальных терминалов при решении основной задачи на другом универсальном вычислительном средстве и выполнять лишь отдельные его заявки. Сегодня стало окончательно ясно, что первые эффективны лишь в роли специализированных вычислительных устройств для решения специальных задач. Вторые твердо заняли место в составе многофункциональных арифметическо-логических устройств (АЛУ) суперскалеров, ибо без конвейеров мы не мыслим себе выполнение всех операций ВС.

Складывается и структура памяти ВС, которая может совмещать в одной установке все способы доступа: от разделяемой (общей) до распределенной оперативной памяти. Однако ограниченные возможности эффективной работы с общей памятью часто диктуют иерархическую структуру ВС, где уровни иерархии (кластеры) отличаются или способом доступа к оперативной памяти, или тем, что каждый кластер имеет свою собственную физическую память в общем адресном пространстве. При этом принцип буферизации, основанный на многоуровневой по быстродействию (и, конечно, — различной по технологии) памяти, на активном использовании Кэш-памяти, продолжает развиваться. Кэш-память, как память самого высокого уровня, претерпевает функциональное разбиение в зависимости от типа данных, для хранения которых она предназначена, либо, в зависимости от вида обработки, — программ или данных.

Все сказанное выше подтверждает перспективность структурных решений при проектировании многопроцессорного комплекса "Эльбрус-3" и его микропроцессорного развития "Эльбрус-3М", "Эльбрус-2К". Таким образом, структура "длинного командного слова" (архитектура VLIW, лежащая в основе EPIC) попадает в разряд классических.

Сейчас микропроцессор, сконцентрировавший все достижения микроэлектроники, является основной составляющей элементно-конструкторской базы ВС. Поэтому понятие "мультимикропроцессорные ВС" пришло на смену понятию "микропроцессорные ВС".

Анализ современных мультимикропроцессорных ВС позволяет выделить те развиваемые характерные решения, которые в условиях микроминиатюризации и снижения энергоемкости, "экономного" логического развития обеспечивают необходимые свойства универсального применения.

Такими решениями являются следующие.

1. Многопроцессорные кристаллы. Воспроизведение многопроцессорной ВС на одном кристалле в значительной степени характерно для сигнальных вычислительных средств, специализирующихся на обработке двух- и трехмерных изображений, которые применяются в цифровом телевидении и радиовещании, при передаче изображений по каналам связи и др. Такие средства эффективно используются в качестве нейрокомпьютеров.

Например, на одном кристалле MVP (Multimedia Video Processor) семейства TMS 320 C80 (фирма Texas Instrument) расположены 4 32 -разрядных цифровых сигнальных процессора (DSP — Digital Signal Processor) с фиксированной запятой (ADSP-0 — ADSP-3). Их особенность — высокая степень конвейеризации и до 64 бит длина командного слова для параллельного выполнения нескольких операций. Система команд содержит команды над битовыми полями и структурами данных, несущими графическую информацию. Такая специализация обусловила понятие — DSP-архитектура.

Процессоры работают независимо. Т.е. ВС — типа MIMD — (Multiple-Instruction, Multiple-Data). Программируются отдельно на ассемблере или ЯВУ. Данными обмениваются через общую внутрикристальную память.

Каждый из ADSP содержит КЭШ-память команд (2 Кбайта), и через матричный коммутатор Crossbar получает доступ к 32 из имеющихся 50 Кбайт быстродействующей статической внутренней памяти. Память расслоенная — поделена на сегменты. Если два и более процессора в одном цикле попытаются обратиться к одному сегменту, аппаратная система управления доступом с циклическим изменением приоритета (round robin prioritization) позволит сделать это только одному процессору.

32 -разрядное АЛУ ADSP может работать как два 16 - или четыре 8 - разрядных АЛУ. Этого достаточно для обработки видеоизображений. Специальные блоки ускоряют обработку графики. Блоки генерации адресов формируют кольцевые (бесконечные) буферы. Аппаратно поддержаны три вложенных цикла.

RISC -процессор управляет четырьмя ADSP с помощью диспетчера. Диспетчер и планировщик заданий тесно взаимодействуют с контроллером пересылок. Кроме того, управляющий процессор самостоятельно выполняет вычисления и обеспечивает обмен с внешними устройствами. Содержит встроенный блок плавающей арифметики и набор векторных операций с плавающей запятой, оптимизированных для обработки изображений, звука и трехмерной графики.

2. Транспьютерная технология. Представленная выше архитектура обладает такой конструктивной законченностью, которая позволяет как встраивать ее в некоторую систему, так и организовать взаимодействие нескольких кристаллов. Это обеспечивается развитыми средствами связи и обмена данными.

Возможность комплексирования привлекла внимание еще на раннем этапе развития микропроцессоров (в середине 1980-х годов) и привела к построению транспьютеров — микропроцессоров, снабженных развитыми средствами комплексирования. Таким образом, создавались "кирпичики", на основе которых можно было создавать сложные структуры. Эта тенденция не только сохранилась, но является необходимым средством построения мультимикропроцессорных ВС.

Преследуя многофункциональность средств обмена, не обязательно требовать их размещения на одном кристалле с центральным процессором. Так, фирма Analog

Devices предлагает микропроцессоры ADSP-21060/62 SHARC ("АКУЛА") для цифровой обработки сигналов, специально предназначенные для комплексирования.

Средства комплексирования "АКУЛЫ":

- магистраль для подключения 6 "АКУЛ" и одного ХОСТ-процессора (управляющего, с привилегированным доступом к магистрали, а также к памяти каждого процессора — через специальный порт);
- сигнальные регистры в составе каждого процессора, непосредственно связанные (одной ножкой) с каждым из других процессоров — для контроля их состояния;
- ЛИНКи — каждый процессор имеет 6 выходов (ЛИНКов) для непосредственной связи "процессор - процессор".

3. Общее адресное пространство комплексируемых микропроцессоров "АКУЛА" обеспечивает псевдообщую память и исключает необходимость программной организации обмена данными. Если адрес физически принадлежит ОП другого процессора, то обмен организуется автоматически, без вмешательства пользователя (т.е. программно не предусматривается).

4. Межпроцессорный (магистральный) обмен инициируется в том случае, если адрес считывания или записи принадлежит адресному пространству другого процессора (единичный обмен). Аналогично возникают групповые пересылки данных с использованием "чужого" адресного пространства.

Пользователь не составляет программу обмена, даже для контроллера обмена данных. Достаточно указать "чужие" адреса.

Процессоры обмениваются сигналами состояния. Поэтому каждый процессор знает, кто является "хозяином" магистрали, т.е. ведет обмен, и свой приоритет в очереди к магистрали. По завершении каждого обмена производится циклическая смена приоритетов процессоров, которым нужна магистраль. Процессор с максимальным приоритетом становится "хозяином".Обмен может прерываться только ХОСТ-процессором.

Микропроцессор утверждается в роли основы элементно-конструкторской базы ВС, и это поняли ведущие разработчики.

В этом смысле привлекает внимание трансформация интересов "отца суперкомпьютеров" С.Крея, который признал определяющую роль принципа MIMD при построении ВС Cray Superserver 6400 System (CS640), выпускаемой корпорацией Cray Research в сотрудничестве с компанией SUN Microsystems (сотрудничество с фирмой SUN ныне характерно и для ведущих российских разработчиков).

Система предполагает наращиваемую конфигурацию от 4 до 64 процессоров SuperSPARC. Принято компромиссное решение на основе классической схемы разделения (общей) ОП при многопроцессорной обработке и распределенной памяти при параллельной обработке массивов. Чтобы работать с частично распределенной памятью в ОЗУ, ВС имеет в любой конфигурации 4 шины. Шина использует сетевую технологию "коммутации пакетов". Это позволяет находить путь обмена единицами информации в соответствии с занятостью или освобождением шин.

В целом, архитектуру следует считать шинной, хотя наличие нескольких шин делает ее промежуточной между шинной и использующей матричный коммутатор.

Направление "мини-супер" призвано поддержать персональный компьютер

То, что говорилось выше, "по умолчанию" соответствует разработке супер-ЭВМ, предназначенных для решения особо сложных задач в составе систем управления в реальном времени, моделирования сложнейших физических процессов, решения задач исследования операций, задач искусственного интеллекта, выполнения роли майнфреймов и серверов в локальных, корпоративных и глобальных сетях.

Супер-ЭВМ уникальна, мало тиражируема, цена ее высока.

С другой стороны, ничто уже не может остановить "победного шествия" персональных компьютеров. Область применения их стала всеобъемлющей. Они используются и там, где могут справиться с задачами, и там, где уже не справляются, несмотря на применение современных суперскалеров.

Тогда целесообразно поставить следующую проблему.

Введем в состав персонального компьютера (PC), как его внешнее устройство, мультимикропроцессорную систему (мультипроцессор), использование которого в монопольном и однозадачном режиме может обеспечить успешное решение задач повышенной сложности.

Действительно, разрешение этой проблемы позволило бы заполнить определенную нишу между супер-ЭВМ и PC, вывести персональный компьютер на уровень мини-супер-ЭВМ. Применение мультипроцессора PC в однопрограммном режиме, при жестком распределении памяти, использование (см. далее) прогрессивной технологии "одна программа — много потоков данных" позволяют существенно снизить издержки производительности на работу ОС, легко "врезать" их в современные операционные системы компьютеров. Сборка такой системы должна производиться на основе существующей микропроцессорной элементно-конструкторской базы, с минимальным использованием вновь разрабатываемых элементов.

Здесь воспроизводится упомянутая выше идея о наличии мониторной системы, на которой решается основная задача, и о наличии интеллектуального терминала, который берет на себя функции, обеспечивающие общую эффективность системы.

Общая схема такой установки показана на рис. 2.1. Выбраны конкретные значения параметров.

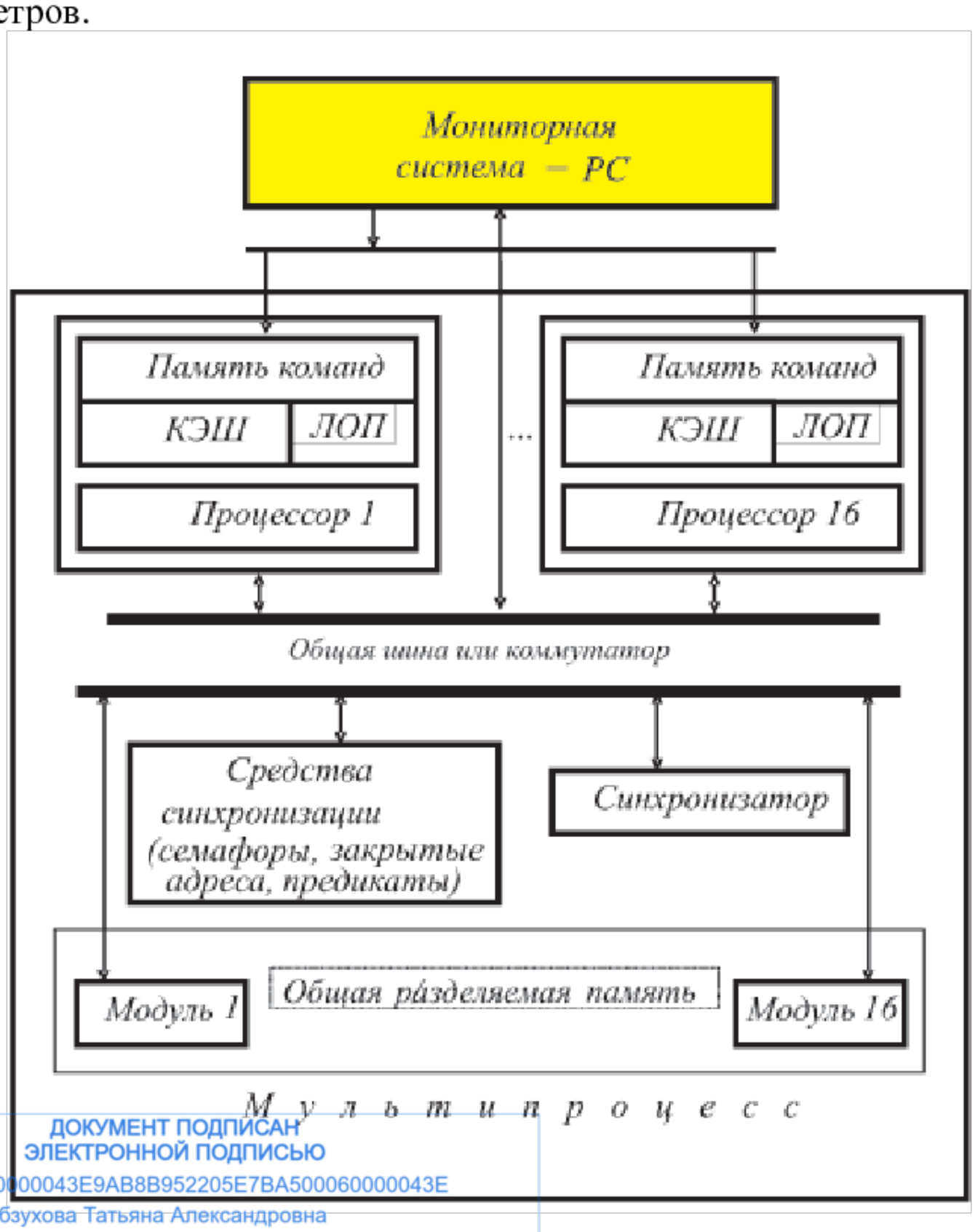


Рис. 2.1. Схема ВС для персонального компьютера

Мультимикропроцессорную приставку к персональному компьютеру целесообразно разработать на основе исследования принципов построения локально-асинхронной архитектуры (SPMD -технологии). Важным достоинством архитектуры является сведение традиционных функций ОС на уровень команд. Т.е. система команд мультипроцессора такова, что позволяет реализовать функции управления параллельным процессом, не требуя запуска процедур ОС. Способствует простоте управления параллельным процессом также монопольный и однозадачный режим использования мультипроцессора. Ниже мы подробнее остановимся на принципах SPMD -технологии. Предполагая первоначальное знакомство с этими принципами, отметим следующее.

Известно (см. далее), что семафоры — универсальное средство синхронизации. Однако семафоры традиционно используют ОС. Чтобы этого избежать, семафоры следует реализовать с помощью предикатного механизма, т.е. с использованием памяти предикатов.

Семафорный механизм может быть эффективно реализован с помощью механизма закрытия адресов (памяти закрытых адресов).

Тогда, в общем случае применения семафоров, должны быть введены команды следующего вида.

Считать по семафору (Сч(С) А). Считывание по адресу производится в случае, если указанный семафор (реализованный в памяти регистрового типа, наряду с индексными и базовыми регистрами) открыт. Если семафор закрыт, реализуется ожидание данного ПЭ без прерывания (т.е. в данном применении пользователь может быть допущен к операциям над семафорами типа "жужжать").

Записать по семафору (Зап(С) А). Запись по адресу производится аналогично предыдущей команде.

При использовании памяти закрытых адресов необходима лишь команда Закрыть адрес. Любое последующее считывание по этому адресу циклически возобновляется (в режиме "жужжания") до тех пор, пока по этому же адресу другой процессор не произведет запись.

В случае использования механизма предикатов адрес некоторой булевой переменной записывается в специальные разряды командного слова. Команда, для которой указанный в ней предикат имеет значение 0, выполняется, в соответствии с кодом операции, в спекулятивном режиме в двух модификациях:

- ожидается присвоение данному предикату значения 1 (в режиме "жужжания");
- пропускается выполнение данной команды.

ПЭ реализует идею RISC -архитектуры и представляет собой функционально законченное устройство, состоящее из микропроцессора, схем обрамления и локальной оперативной памяти (ЛОП). Локальная память процессора содержит область для хранения стеков вычислительного процесса, в том числе — стеков подпрограмм и вложенных циклов. В других областях этой памяти хранятся модификаторы, дескрипторы массивов и локальные величины. Здесь же находятся микропрограммы, реализующие систему команд ВС.

Общая (разделяемая) память (ОП) содержит М модулей с общим адресным пространством и реализует принцип интерливинга, предполагающий, что смежные ячейки памяти находятся в разных модулях.

Синхронизатор предназначен для обеспечения одновременного пуска программ или их модулей.

Возможно использование простейших коммутаторов для обмена ПЭ с модулями памяти.

Распределенный и разделяемый вычислительный ресурс второго уровня. Решающие поля

Произведем некоторые обобщения.

Итак, второй уровень распараллеливания предполагает распределение команд, инструкций, операций, элементарных функций и других несложных процедур — для выполнения исполнительными устройствами процессоров или в общем вычислительном ресурсе симметричной ВС. Здесь существуют свои проблемы, связанные с "элементарным" характером операций, небольшим объемом содержащихся в них работ, с их еще большей критичностью по отношению к "накладным расходам" на организацию и синхронизацию. Мы предполагаем, что исполнительные устройства ВС образуют вычислительный ресурс второго уровня (распараллеливания).

Сложились традиции построения этого ресурса, где основное внимание уделяется построению многофункциональных АЛУ. Однако в ряде архитектур пока еще робко пробивает себе дорогу объединение АЛУ в единый разделяемый ресурс системы — построение решающих полей.

Проработка этой идеи проводилась неоднократно в отечественной практике разработки ВС. Она проявлялась во включении в состав ВС специализированных процессоров на правах интеллектуальных терминалов для эффективного выполнения определенных операций. Это были векторные процессоры с доступом от нескольких ЦП (ПС-3100), векторный сопроцессор в ПС-2100. Эта же идея фактически воплощена в семействе "Эльбрус", допускающем включение в свой состав спецпроцессоров — эмуляции других систем, векторно-конвейерных модулей и др.

При реализации идеи решающего поля проблема выбора и развития вычислительного ресурса неотделима от выбора вариантов архитектуры системы вообще. Единственным средством обоснования и исследования этого выбора является моделирование. Построение детерминированных имитаторов позволяет с любой детализацией выявить целесообразные технические решения и обосновать язык системы. Целью стохастических моделей является оценка эффективности различных архитектур на основе полноты полезной загрузки оборудования.

На основе традиций разработки многопроцессорных симметричных вычислительных систем можно сделать вывод о практике и тенденции развития вычислительного ресурса второго уровня.

Все модели семейства МВК "Эльбрус" предполагают наличие в составе АЛУ процессора нескольких исполнительных устройств (ИУ), специализированных по типам операций. Тогда в целом для ВС можно сказать, что вычислительный ресурс второго уровня является распределенным и неоднородным (рис. 2.2).

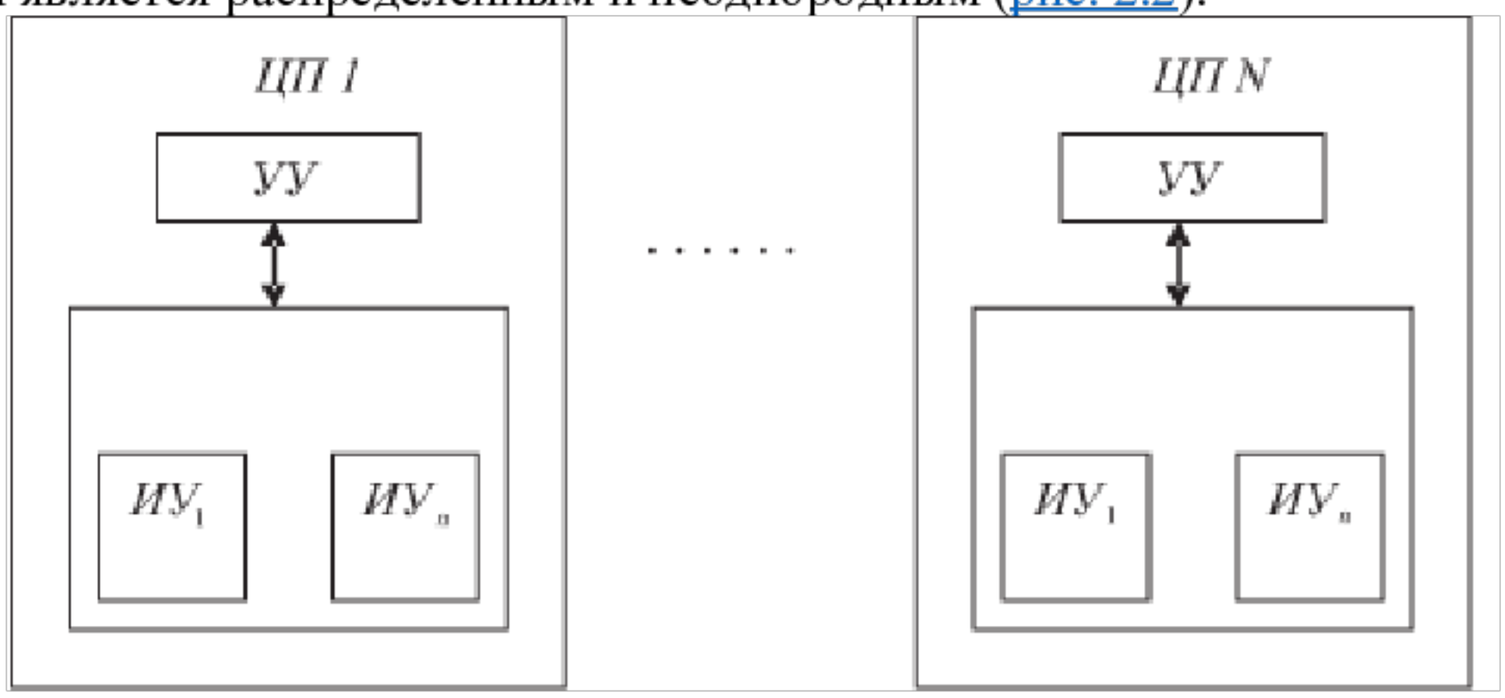


Рис. 2.2. ВС с распределённым решающим полем

Выше говорилось, что использование ресурса второго уровня неотделимо от общих идей функционирования процессоров ВС — от их архитектуры и от архитектуры ВС в целом. Поэтому сказанного о распределенном ресурсе недостаточно, надо говорить и о способе его использования.

Так, в МВК "Эльбрус-2" применена динамическая загрузка ИУ в процессе выполнения последовательности безадресных команд программы, которую подробно

рассмотрим в лекции 3. Обобщенный алгоритм такой загрузки основан на промежуточном переводе безадресных команд в трехадресные. Появление адресов аргументов и результатов в каждой команде позволяет на основе совместного анализа нескольких команд, представленных в "окне просмотра", выявлять их частичную упорядоченность и выделять независимые команды для одновременного выполнения. Это — адресный метод распараллеливания, который рассматривался в архитектуре data flow. Команды проходят стадии обработки, как показано на рис. 2.3.

Рис. 2.3. Схема оптимизатора-компоновщика "длинных" командных слов

Однако проект МВК "Эльбрус-3", породивший микропроцессорное воплощение — МВК "Эльбрус-3М", — основан на использовании идеи "длинного" командного слова и управления каждым тактом системы. Динамическое распределение работ между ИУ заменено статическим — предписанием каждому ИУ, что он должен начать делать в данном такте. В "длинном" командном слове, в соответствующих позициях, записаны инструкции каждому ИУ.

Это означает, что решение проблем оптимального использования ИУ, их синхронизации при выполнении данного алгоритма возлагаются на оптимизирующий транслятор. Он фактически производит диспетчирование, оптимальное планирование параллельного вычислительного процесса на одном процессоре.

Способы распараллеливания

Различают два основных способа распараллеливания: по управлению и по информации.

Первый способ — представление алгоритма задачи в виде частично-упорядоченной последовательности выполняемых работ. Затем в результате диспетчирования реализуется оптимальный план выполнения работ в ВС при ограничениях на время выполнения всего алгоритма или за минимальное время.

Основой является представление алгоритма граф-схемой G, отражающей информационные связи между работами (задачами, процессами, процедурами, операторами, макрокомандами и т.д.), на которые разбит алгоритм. Граф G — взвешенный, ориентированный, без контуров.

Для исследования графа и диспетчирования используют матрицы следования S ; их дополняют столбцом T весов — получают расширенные матрицы следования S* (рис. 2.4).

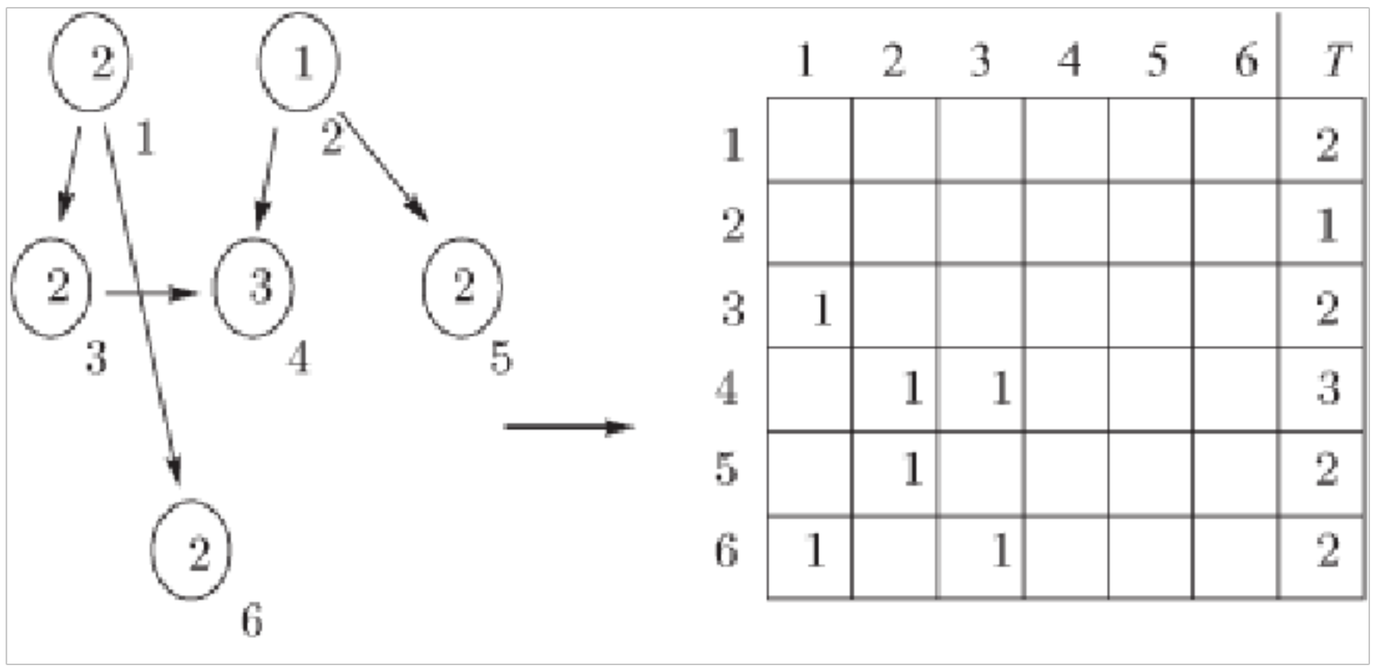


Рис. 2.4. Исходная информация для распараллеливания

Здесь предполагаем, что ВС — однородная, с общей (разделяемой) памятью, т.е. потерями времени на обмен между работами можно пренебречь.

Пусть ВС содержит два процессора (n = 2). Тогда в результате оптимального распределения получим план (рис. 2.5).

Рис. 2.5. Временная диаграмма параллельного выполнения работ

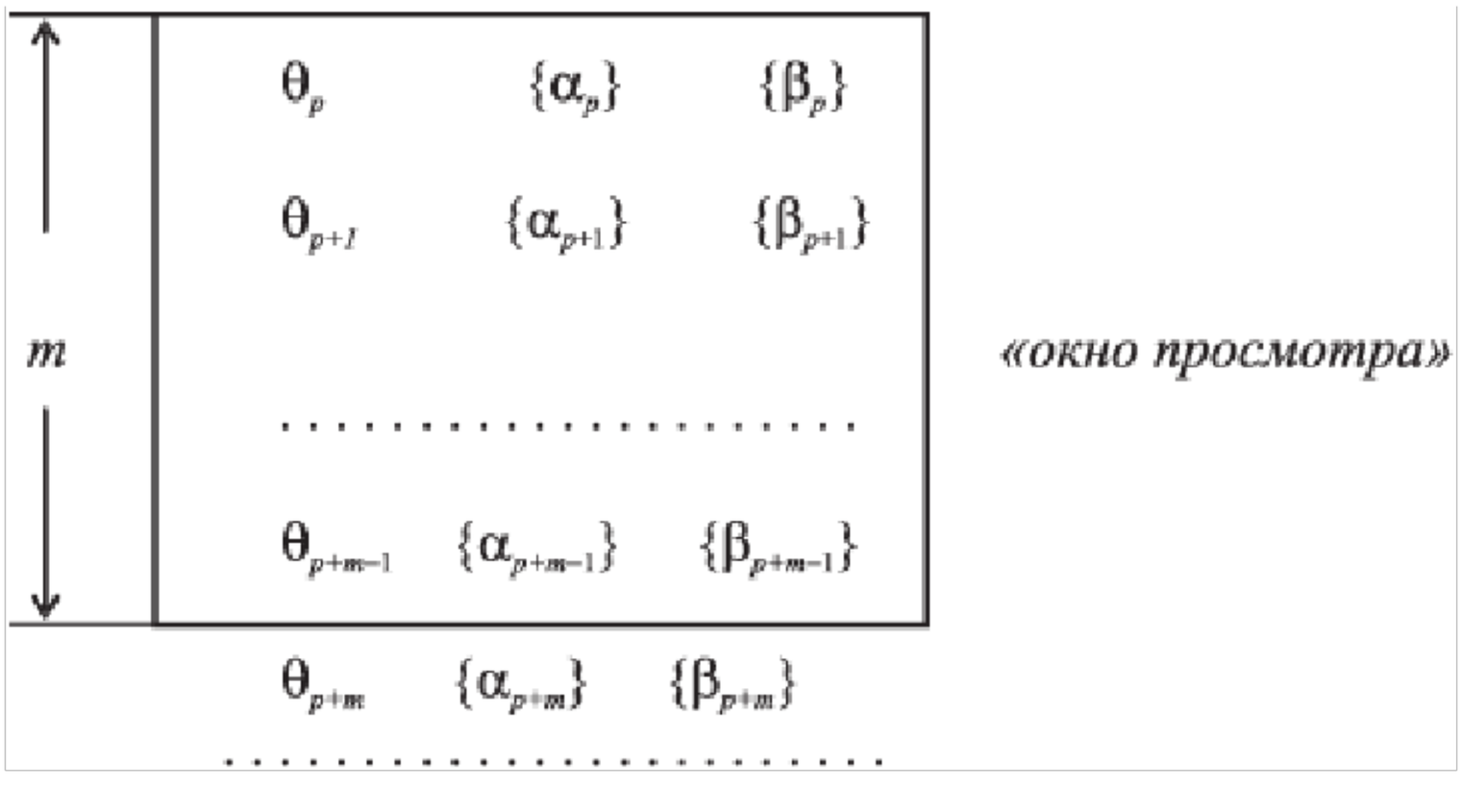
План действительно совпадает с оптимальным, т.к. длина расписания T = 7, что совпадает с длиной критического пути в графе, Ткр = 7 (путь 1 -> 3 -> 4).

В общей схеме организации параллельного вычислительного процесса мы не полностью раскрыли содержание блока 3 — интерпретации потока макроинструкций в виде, удобном для работы диспетчера. Сейчас мы определили, что такой вид — это матрица следования. Значит, в случае необходимости автоматического формирования матрицы следования надо определять информационную взаимосвязь макроинструкций в пределах видимости, т.е. в "окне просмотра". Таким образом, по текущему содержимому "окна просмотра" надо формировать текущий вид матрицы следования.

Вспомним, что мы уже в упрощенном виде решали подобную задачу, например, когда по формируемому потоку трехадресных команд определяли их информационную взаимосвязь и определяли возможность одновременного выполнения этих команд.

Обобщим эту задачу.

Возвращаясь к названной схеме, представим себе, что поток макроинструкций (блок 2) следует через "окно просмотра" так, что для планирования оптимальной загрузки процессоров диспетчер может анализировать некоторое множество этих макроинструкций и из них выбирать вариант назначения их на процессоры для выполнения. Каждая макроинструкция может интерпретироваться и как процедура, где можно выделить имя θ_μ, множество $\{\alpha_\mu\}$ входных параметров, множество $\{\beta_\mu\}$ выходных параметров. На рис. 2.6 отображено "окно просмотра", через которое следует поток макроинструкций.



«окно просмотра»

Рис. 2.6. Обработка "окна просмотра"

Составим по его содержимому соответствующую матрицу следования размерности m x m:

$$S = ||\alpha_{\mu\nu}||_I^m = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \dots & \alpha_{1m} \\ \dots & \dots & \dots & \dots \\ \alpha_{m1} & \alpha_{m2} & \dots & \alpha_{mm} \end{pmatrix}$$

$$\alpha_{\mu\nu} = \begin{cases} 1, \text{ если } \varepsilon_{\mu\nu} \neq \varnothing, \\ 0, \text{ в противном случае;} \end{cases}$$

$$\varepsilon_{\mu\nu} = (\{\alpha_\mu\} \cap \{\beta_\nu\}) \cup (\{\beta_\mu\} \cap \{\alpha_\nu\} \cup \{\beta_\nu\}) \text{ для всех } \nu < \mu.$$

По матрице следования S диспетчер производит назначение.

После выполнения макроинструкций они исключаются из "окна прросмотра", оставшиеся макроинструкции уплотняются вверх, а снизу "окно просмотра" пополняется новыми макроинструкциями. С учетом вновь поступивших макроинструкций уточняется текущий вид матрицы следования S и процесс диспетчирования продолжается.

По такой же схеме, а именно, на основе первого способа распараллеливания — по управлению — решается другая важная задача распараллеливания: компоновки длинных командных слов в оптимизирующем трансляторе. Назначение работы на ИУ осуществляется здесь в виде записи соответствующей инструкции в позицию длинного командного слова, соответствующую ИУ. Т.е. план параллельного выполнения работ (команд, операций) фиксируется в длинных командных словах, в которых предусмотрены инструкции каждому ИУ, которые они должны начать выполнять с данного такта.

Второй способ распараллеливания — по информации — используется тогда, когда можно распределить обрабатываемую информацию между процессорами для обработки по идентичным алгоритмам (по одному алгоритму).

1. Рассмотрим задачу умножения матриц $A \times B = C$:

$$\begin{pmatrix} a_{11} & \dots & a_{1m} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mm} \end{pmatrix} \times \begin{pmatrix} b_{11} & \dots & b_{1m} \\ \dots & \dots & \dots \\ b_{m1} & \dots & b_{mm} \end{pmatrix} = \begin{pmatrix} c_{11} & \dots & c_{1m} \\ \dots & \dots & \dots \\ c_{m1} & \dots & c_{mm} \end{pmatrix}$$

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$$

Развернем матрицу — результат C — в линейный (одномерный) массив, переименуем ее элементы и заменим два индекса на один:

c_{11}	c_{12}	. . .	c_{1m}	c_{21}	. . .	c_{2m}	c_{31}	. . .	c_{mm}
d_1	d_2	. . .	d_m	d_{m+1}	. . .	d_{2m}	d_{2m+1}	. . .	d_{m^2}

Пусть ВС содержит n процессоров. Выберем следующий план счета элементов матрицы С:

процессор 1 считает элементы d1, d1+n, d1+2n, ...

процессор 2 считает элементы d2, d2+n, d2+2n, ...

.....

процессор n считает элементы dn, d2n, d3n, ...

По-видимому, все они будут выполнять одну и ту же программу, но обрабатывать разные наборы данных. (Мы снова столкнулись с целесообразностью SPMD -технологии.)

Здесь не потребовалась какая-либо синхронизация параллельного вычислительного процесса.

2. Рассмотрим задачу счета способом "пирамиды".

Эту задачу мы исследовали при рассмотрении ВС типа SPMD. Посмотрим еще раз, какая синхронизация нам здесь потребуется.

Пусть необходимо перемножить все элементы некоторого массива {a1,a2,... , a10}. Каждый элемент занимает одну ячейку памяти. Пусть число процессоров в ВС n=4. Чтобы распараллелить этот процесс, примем схему счета "пирамидой" (рис. 2.7).

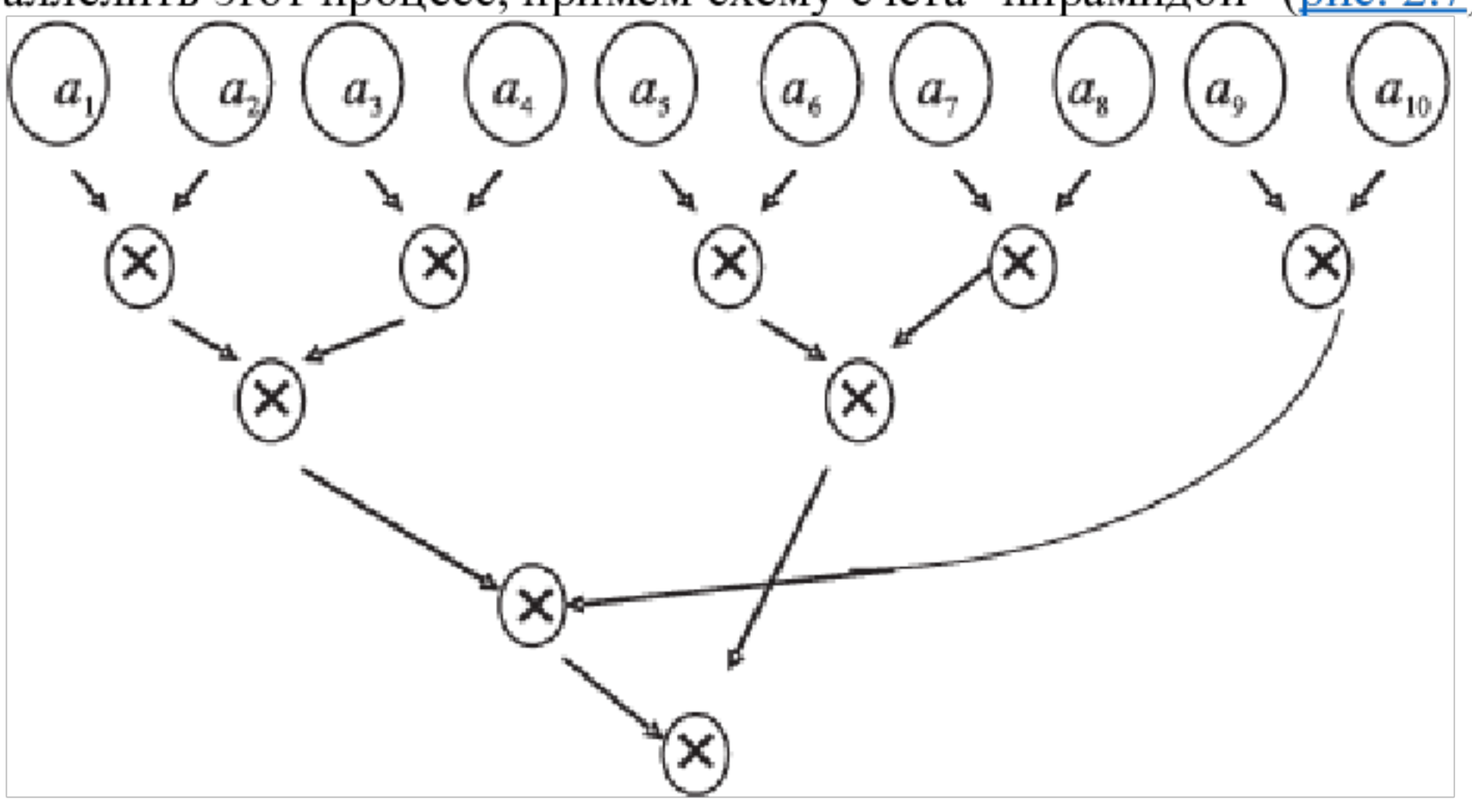


Рис. 2.7. Граф-схема выполнения операции "свёртки"

Количество уровней операций в ней]log2 m[=]log210[=4 ( ]x[— ближайшее целое, не меньшее x).

Расширим массив, дополнив его ячейками, в которых будем хранить промежуточные частные произведения. Тогда весь план счета примем таким, как показано на рис. 2.8. Отмечены процессоры, выполняющие указанную операцию.

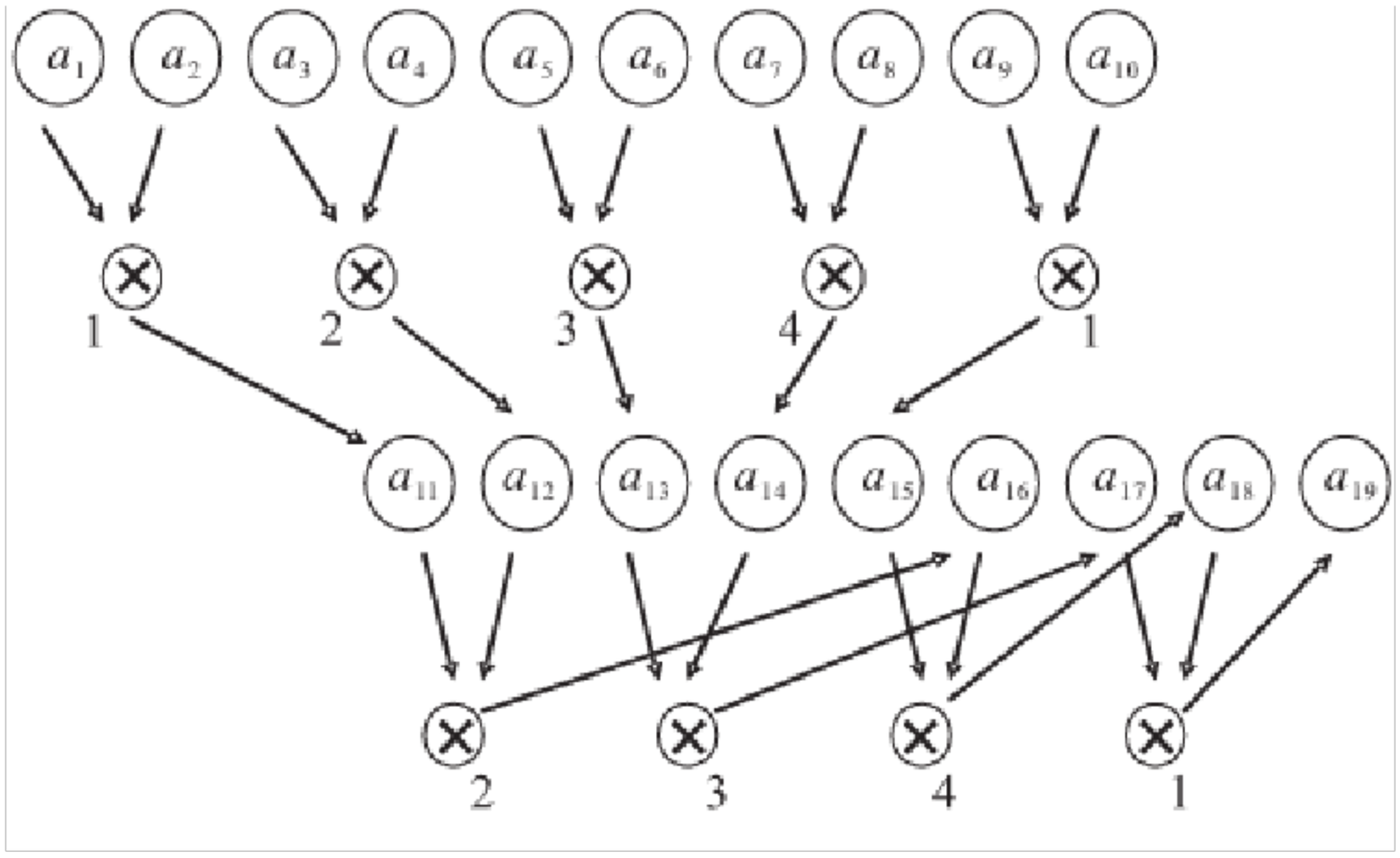


Рис. 2.8. Схема выполнения операции свёртки четырьмя процессорами

Следовательно, надо так написать программу, одну для всех процессоров, предусмотрев необходимую переадресацию для выборки и вычисления "своих" данных, чтобы по ней выбирались два соседних элемента этого удлиненного массива, а результат их умножения отправлялся в очередную ячейку этого "удлинения".

Возникает только одна трудность: для первых пяти произведений данные есть, а вот последующие произведения должны выполняться тогда, когда для них будут найдены исходные данные.

Значит, процессоры, которым выпало произвести такие умножения, должны "уметь" обнаруживать отсутствие данных и дожидаться их появления. Т.е. требуется синхронизация процессоров по использованию общих данных.

Здесь распараллеливание по данным смыкается с распараллеливанием по управлению.

Возможная схема общей для всех процессоров программы — на рис. 2.9. Она реализована в примере для ВС типа SPMD.

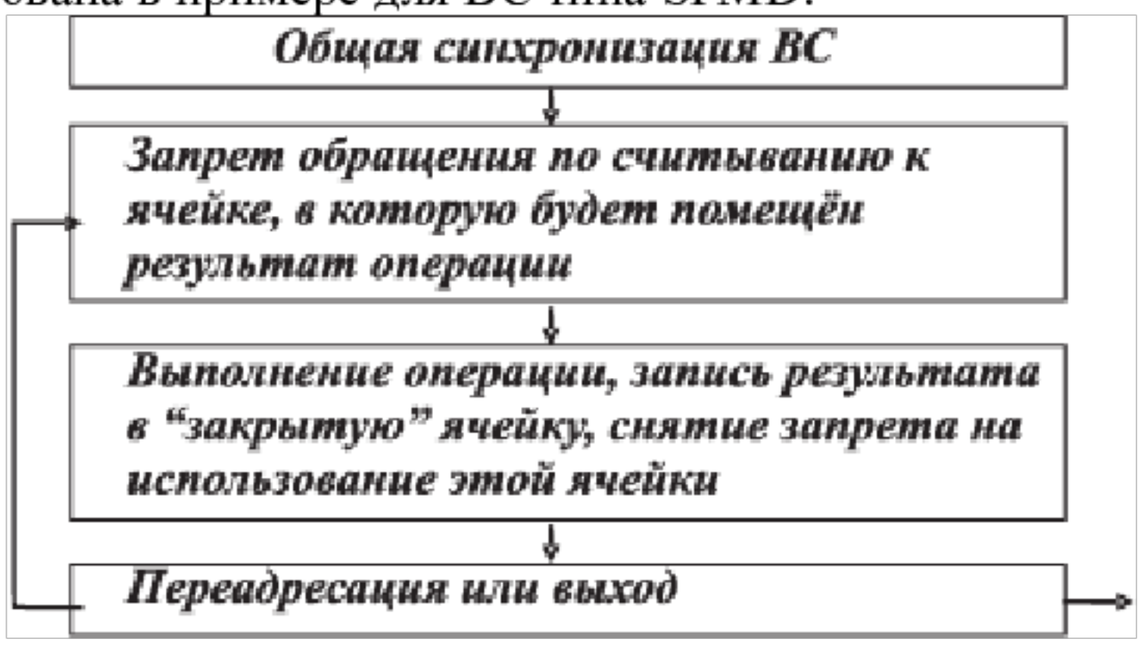


Рис. 2.9. Схема программной синхронизации при выполнении операции "свёртки"

8. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Оценка «отлично» выставляется студенту, если он продемонстрировал глубокие, исчерпывающие знания и творческие способности в понимании, изложении и использовании учебно-программного материала; логически последовательные, содержательные, полные, правильные и конкретные ответы на все поставленные вопросы

и дополнительные вопросы преподавателя; свободное владение основной и дополнительной литературой, рекомендованной учебной программой.

Оценка «хорошо» выставляется студенту, если он продемонстрировал твердые и достаточно полные знания всего программного материала, правильное понимание сущности и взаимосвязи рассматриваемых процессов и явлений; последовательные, правильные, конкретные ответы на поставленные вопросы при свободном устранении замечаний по отдельным вопросам; достаточное владение литературой, рекомендованной учебной программой.

Оценка «удовлетворительно» выставляется студенту, если он продемонстрировал твердые знания и понимание основного программного материала; правильные, без грубых ошибок ответы на поставленные вопросы при устранении неточностей и несущественных ошибок в освещении отдельных положений при наводящих вопросах преподавателя; недостаточное владение литературой, рекомендованной учебной программой.

Оценка «неудовлетворительно» выставляется студенту, если он продемонстрировал неправильные ответы на основные вопросы, допущены грубые ошибки в ответах, непонимание сущности излагаемых вопросов; неуверенные и неточные ответы на дополнительные вопросы.

9. ОРГАНИЗАЦИЯ КОНТРОЛЯ ЗНАНИЙ СТУДЕНТОВ

Успешное освоение дисциплины основывается на систематической работе студентов. В процессе самостоятельной работы студенты в течение одного – двух дней прорабатывают материалы лекционных и лабораторных занятий по конспектам и рекомендованной основной литературе.

Конспекты дополняются материалами, полученными при проработке дополнительной литературы. При подготовке к письменной контрольной работе необходимо самостоятельно проработать задания из соответствующих глав рекомендуемой литературы.

Написание конспекта по темам самостоятельного изучения является одной из форм обучения студентов. Данная форма направлена на организацию и повышение уровня самостоятельной работы студентов.

Конспект, как форма обучения студентов - это краткий обзор максимального количества доступных публикаций по заданной теме, подготовка самого реферативного обзора и презентации по нему. При проведении обзора должна проводиться и исследовательская работа, но объем ее ограничен, так как анализируется уже сделанные выводы и в связи с небольшим объемом данной формы работы. Преподавателю предоставляется сам конспект и презентация к нему. Сдача конспекта происходит в форме защиты-доклада с использованием подготовленной презентации.

При проверке конспектов дается анализ качества их ведения. Отмечаются допущенные ошибки, даются рекомендации по улучшению качества конспектирования изучаемого материала.

Тема и направленность контрольной работы представлена в методических рекомендациях. Контрольная работа составляется из типовых заданий, рассмотренных на занятиях. При выполнении контрольной работы студенты должны выполнить задания, показав при этом понимание теоретического материала и навыки решения практических задач. При выполнении контрольной работы студенты должны кроме основной и дополнительной рекомендованной литературы использовать и другие источники.

Формы контроля знаний студентов

Контроль и оценка знаний, умений и навыков студентов осуществляется на лабораторных занятиях, консультациях, при сдаче зачета. В ходе контроля знаний преподаватель оценивает понимание студентом содержания дисциплины «Вычислительные системы».

Контроль знаний студентов может осуществляться в следующих формах:

- текущий контроль знаний;
- итоговый контроль знаний.

Текущий контроль знаний студентов имеет целью: дать оценку работы каждого студента по усвоению им учебного материала, выявить недостатки в его подготовке и оказать практическую помощь в их устранении;

Основными формами текущего контроля знаний студентов являются:

- устный контрольный опрос;
- защита лабораторной работы;
- проверка конспектов лекций;
- проверка конспектов по темам, вынесенных на самостоятельное изучение.

Устный контрольный опрос студентов проводится на лекциях (и лабораторных занятиях). По его результатам преподаватель оценивает качество подготовки студента к занятию.

На лабораторных занятиях знания и практические навыки студентов оцениваются по 5-балльной системе.

Рекомендации по подготовке к зачету

Зачет, как итоговый контроль знаний студентов имеет целью проверить и оценить учебную работу студентов, уровень полученных знаний и практических навыков.

Зачет с оценкой проводится в 1 семестре после защиты всех лабораторных работ в объеме учебной программы.

Рекомендации по работе с литературой и источниками

Изучение литературы и источников необходимо начинать с прочтения соответствующих глав учебных изданий, учебных пособий или литературы, рекомендованной в качестве основной или дополнительной по дисциплине «Вычислительные системы», которые прямо или косвенно относятся к изучаемой теме.

При изучении литературы и источников студенту рекомендуется вести краткий конспект. Однако не следует переписывать все содержание изучаемой темы, нужно выписывать лишь основные идеи и главные мысли. В отдельных случаях, когда встречаются важные определения, понятия, необходимый фактический материал и примеры, статистическая информация, имеющие отношение к изучаемой теме, необходимо выписать их в виде цитат с полным указанием библиографических источников.

Конспектирование рекомендуемой литературы и источников необходимо вести с распределением собранных материалов по отдельным главам и параграфам согласно учебно-тематическому плану. Необходимо выписывать все выходные данные по используемой литературе и источникам.

Основой технологии интенсификации обучения являются учебно-иллюстрационные материалы (презентации) по дисциплине.

Работа с учебно-иллюстрационными материалами имеет следующие этапы.

1. Изучение теоретических основ учебного материала в аудитории: изложение преподавателем изучаемого материала студентам с объяснением;

2. Самостоятельная работа: индивидуальная работа студентов по конспекту;

3. Первое повторение - воспроизведение содержания заданной темы конспекта по памяти.

4. Устное проговаривание материала конспекта – необходимый этап внешне речевой деятельности при усвоении учебного материала.

5. Второе повторение – взаимоопрос и взаимопомощь студентов друг другу.

Применение учебно-иллюстрационных материалов позволяет обобщить сложный по содержанию материал, активизировать мыслительную деятельность студентов.

Необходимо помнить, что главное для студента в самостоятельной работе с рекомендуемой литературой и источниками - это формирование своего индивидуального стиля, который может стать основой в будущей профессиональной деятельности.

10. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Рекомендуемая литература

Основная литература

1. Бройдо, В.Л. Вычислительные системы, сети и телекоммуникации: учебник/ В. Л. Бройдо, О. П. Ильина- СПб.: Питер, 2011.
2. Пятибратов, А.П. Вычислительные системы, сети и телекоммуникации: учебник/ А. П. Пятибратов, Л. П. Гудыно, А. А. Кириченко; ред. А. П. Пятибратов - М.: Финансы и статистика, 2011.

Дополнительная литература

1. Олифер, В. Г. Компьютерныесети. Принципы, технологии, протоколы: учеб. пособие / В. Г. Олифер, Н. А. Олифер. - 4-е изд. - СПб.: Питер, 2011.
2. Таненбаум, Э. С. Архитектуракомпьютера / Э. С. Таненбаум; пер.: Ю. Гороховский, Д. Шинтяков. - 5-е изд. - СПб.: Питер, 2009.

Интернет-ресурсы

1. http://www.intuit.ru – сайт дистанционного образования в области информационных технологий;
2. http://www.iqlib.ru - интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия;
3. http://www.biblioclub.ru - электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике, а так же содержит материалы по точным наукам;
4. http://window.edu.ru – образовательные ресурсы ведущих вузов;

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Направление подготовки	**09.04.02**
Направленность (профиль)	**Информационные системы и технологии** **«Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Методические указания содержат перечень вариантов заданий и требования к оформлению контрольных работ.

Теоретической основой подготовки специалиста являются знания в области вычислительной систем.

5. ЦЕЛЬ, ЗАДАЧИ И РЕАЛИЗУЕМЫЕ КОМПЕТЕНЦИИ

Методические указания составлены с учетом требований стандарта высшего образования по дисциплине: «Вычислительные системы». Целью освоения дисциплины «Вычислительные системы» является получение магистрантами знаний о принципах организации современных вычислительных систем, их структуре и функционировании.

код	Формулировка:
ПК-1	способность осуществлять управление, развитием баз данных, включая развертывание, сопровождение, оптимизацию функционирования баз данных, являющихся частью различных информационных систем
ПК-4	способность выполнять разработку систем управления базами данных, операционных систем, организацию разработки системного программного обеспечения, интеграция разработанного системного программного обеспечения

6. ФОРМУЛИРОВКА ЗАДАНИЯ И ЕГО ОБЪЕМ

Контрольная работа включат в себе вопросы по темам, которые нужно изучить самостоятельно и по ним подготовить отчет и презентации. Результаты выполнения контрольной работы предоставляются в электронном виде. Объем контрольной работы составляет 10-15 печатных листов формата А 4.

Варианты заданий выбираются из таблицы по последним двум цифрам зачетной книжки.

Послед цифра	Предпоследняя цифра									
	0	1	2	3	4	5	6	7	8	9
0	1,11	2,12	3,13	4,14	5,15	6,16	7,17	8,18	9,19	10,21
1	11,22	12,23	13,24	14,25	15,26	16,27	17,27	18,28	19,29	20,30
2	1,30	2,29	3,28	4,27	5,26	6,25	7,24	8,23	9,22	10,21
3	11,29	12,28	13,27	14,26	15,25	16,24	17,23	18,22	19, 21	20,10
4	19,31	18, 32	17, 33	16,34	15,35	14,18	13,19	12,20	11,21	10,22
5	9,35	8,34	7,33	6,32	5,31	4,30	3,29	2,28	1,27	30,26
6	30,11	29,10	28,9	27,8	26,7	25,6	24,5	23,4	22,3	21, 2

7	20,31	19, 30	18,29	17,28	16,27	15,26	14,25	13,24	12,23	11,22
8	10,29	9,31	8,33	7,35	6,25	5,23	4,21	3,19	2,17	1,15
9	1, 20	2,26	3,24	4,23	5,22	8,20	9,18	11,17	8,30	7, 31

7. ОБЩИЕ ТРЕБОВАНИЯ К НАПИСАНИЮ И ОФОРМЛЕНИЮ РАБОТЫ

Контрольная работа выполняется и сдается в электронном виде на CD/CDRW носителе. На конверте необходимо указать название дисциплины, ФИО студента, факультет, номер группы, шифр зачетной книжки, № варианта задания, и список всех созданных в ходе выполнения задания файлов.

Приведенный в конце методических указаний список литературы может использоваться студентами при выполнении контрольной работы.

8. ВАРИАНТЫ ЗАДАНИЙ ДЛЯ СТУДЕНТОВ ЗАОЧНОЙ ФОРМЫ ОБУЧЕНИЯ

1. Назначение, область применения и способы оценки производительности многопроцессорных вычислительных систем.
2. Архитектура вычислительных систем. Классификация архитектур по параллельной обработке данных.
3. Вычислительные системы нетрадиционной архитектуры.
4. Параллельные структуры вычислительных систем.
5. Микропроцессорные системы и способы распараллеливания.
6. Распараллеливание в ВС на уровне исполнительных устройств.
7. Параллельная обработка стека и статическое распараллеливание в решающем поле.
8. Оптимальное программирование в архитектуре управления каждым тактом
9. Асинхронная ВС на принципах "data flow".
10. Архитектура вычислительных систем. SMP и MPP-архитектуры. Гибридная архитектура (NUMA). Организация когерентности многоуровневой иерархической памяти.
11. Архитектура вычислительных систем. PVP-архитектура. Кластерная архитектура.
12. Архитектура высокопроизводительных процессоров и кластерных систем.

13. Примеры архитектур кластерных вычислительных систем: Blue Gene/L, семейство SGI Altix.
14. Основы программирования на MPI.
15. Принципы построения коммуникационных сред.
16. Способы организации высокопроизводительных процессоров. Ассоциативные процессоры. Конвейерные процессоры. Матричные процессоры.
17. Способы организации высокопроизводительных процессоров. Клеточные и ДНК-процессоры. Коммуникационные процессоры.
18. Способы организации высокопроизводительных процессоров. Процессоры баз данных.
19. Способы организации высокопроизводительных процессоров. Потоковые процессоры. Нейронные процессоры.
20. Способы организации высокопроизводительных процессоров. Процессоры с многозначной (нечеткой) логикой.
21. Коммутаторы для многопроцессорных вычислительных систем. Простые коммутаторы.
22. Коммутаторы для многопроцессорных вычислительных систем. Составные коммутаторы. Распределенные составные коммутаторы.
23. Требования к компонентам МВС.
24. Надежность и отказоустойчивость МВС.
25. Кластеры и массивно-параллельные системы различных производителей. Примеры кластерных решений IBM.
26. Кластеры и массивно-параллельные системы различных производителей. Примеры кластерных решений HP.
27. Кластеры и массивно-параллельные системы различных производителей. Примеры кластерных решений SGI.
28. Кластеры и массивно-параллельные системы различных производителей. SMP Power Challenge фирмы Silicon Graphics. Семейство SUN Ultra Enterprise компании SUN.
29. Кластеры и массивно-параллельные системы различных производителей. Семейство массивно-параллельных машин ВС МВС-100 и МВС-1000. ВС с распределенной памятью производства Sequent и DATA GENERAL. Кластеры
30. Кластеры и массивно-параллельные системы различных производителей. Современные суперкомпьютеры Hitachi SR8000.

31. Кластеры и массивно-параллельные системы различных производителей. Современные суперкомпьютеры серии Fujitsu VPP5000,Cray T3E-1200, ASCI White.
32. Кластеры и массивно-параллельные системы различных производителей. Современные суперкомпьютеры Cray T3E-1200.
33. Кластеры и массивно-параллельные системы различных производителей. Современные суперкомпьютеры ASCI White.
34. Аппаратная поддержка языка пользователя — основная концепция мультипроцессорных систем.
35. Задача логического вывода и когерентность кэш-памяти в ВС SPMD-архитектуры.

9. ПЛАН-ГРАФИК ВЫПОЛНЕНИЯ ЗАДАНИЯ

Дата получения задания	Дата предоставления выполненного задания
Установочная сессия. Сентябрь 2019 г.	Зимняя сессия за две недели до начала сессии. Ноябрь 2019 г.

10. КРИТЕРИИ ОЦЕНИВАНИЯ РАБОТЫ

Оценка «отлично» выставляется студенту, если он продемонстрировал глубокие, исчерпывающие знания и творческие способности в понимании, изложении и использовании учебно-программного материала; логически последовательные, содержательные, полные, правильные и конкретные ответы на все поставленные вопросы и дополнительные вопросы преподавателя; свободное владение основной и дополнительной литературой, рекомендованной учебной программой.

Оценка «хорошо» выставляется студенту, если он продемонстрировал твердые и достаточно полные знания всего программного материала, правильное понимание сущности и взаимосвязи рассматриваемых процессов и явлений; последовательные, правильные, конкретные ответы на поставленные вопросы при свободном устранении замечаний по отдельным вопросам; достаточное владение литературой, рекомендованной учебной программой.

Оценка «удовлетворительно» выставляется студенту, если он продемонстрировал твердые знания и понимание основного программного материала; правильные, без грубых ошибок ответы на поставленные вопросы при устранении неточностей и несущественных

ошибок в освещении отдельных положений при наводящих вопросах преподавателя; недостаточное владение литературой, рекомендованной учебной программой.

Оценка «неудовлетворительно» выставляется студенту, если он продемонстрировал неправильные ответы на основные вопросы, допущены грубые ошибки в ответах, непонимание сущности излагаемых вопросов; неуверенные и неточные ответы на дополнительные вопросы.

11. ПОРЯДОК ЗАЩИТЫ РАБОТЫ

Защита контрольной работы проводится в виде научного дискурса с презентацией выполненных заданий, в соответствии с графиком защиты. После доклада студенту задаются вопросы, как преподавателем, так и студентами группы.

В процессе защиты своей работы студент делает доклад продолжительностью 7-10 минут. Доклад должен быть предварительно подготовлен студентом. Лучшее впечатление производит доклад, в форме пересказа, без зачтения текста, которым следует пользоваться только для уточнения цифрового материала. Студент должен свободно ориентироваться в своей работе.

В выступлении необходимо корректно использовать демонстрационные материалы, которые усиливают доказательность выводов и облегчают восприятие доклада студента. Они оформляются в виде презентации в системе Power Point.

12. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Рекомендуемая литература:

Основная литература

1. Бройдо, В.Л. Вычислительные системы, сети и телекоммуникации: учебник/ В. Л. Бройдо, О. П. Ильина- СПб.: Питер, 2011;
2. Пятибратов, А.П. Вычислительные системы, сети и телекоммуникации: учебник/ А. П. Пятибратов, Л. П. Гудыно, А. А. Кириченко; ред. А. П. Пятибратов - М.: Финансы и статистика, 2011.

Дополнительная литература

1. Олифер, В. Г. Компьютерныесети. Принципы, технологии, протоколы: учеб. пособие / В. Г. Олифер, Н. А. Олифер. - 4-е изд. - СПб.: Питер, 2011;
2. Таненбаум, Э. С. Архитектуракомпьютера / Э. С. Таненбаум; пер.: Ю. Гороховский, Д. Шинтяков. - 5-е изд. - СПб.: Питер, 2009.

Интернет-ресурсы

1. http://www.intuit.ru – сайт дистанционного образования в области информационных технологий;
2. http://www.iqlib.ru - интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия;
3. http://www.biblioclub.ru - электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике, а так же содержит материалы по точным наукам;
4. http://window.edu.ru – образовательные ресурсы ведущих вузов;