Архитектура параллельных вычислительных систем

к.т.н., доцент
Костичев Сергей Валентинович

Тенденции развития высокопроизводительных вычислительных систем

snenv@mail.ru

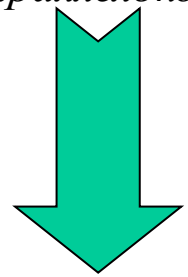
Учебные вопросы:

1. Основные понятия параллелизма
2. Формальная модель ускорения. Закон Амдаля-Уэра
3. Сдерживающие факторы применение параллелизма
4. Пути увеличения производительности ЭВМ
   - Совершенствование элементной базы
   - Совершенствование архитектуры ЭВМ
     - Совершенствование процесса выполнения инструкций
     - Смена парадигмы организации вычислений

Основные понятия параллелизма

Опр.1 ***Степенью параллелизма*** численного алгоритма называется число его операций, которые можно выполнять одновременно

Пример $\vec{a}+\vec{b}: \underbrace{a_i+b_i}_{\text{независимо и параллельно}}, i=1,\ldots,n$

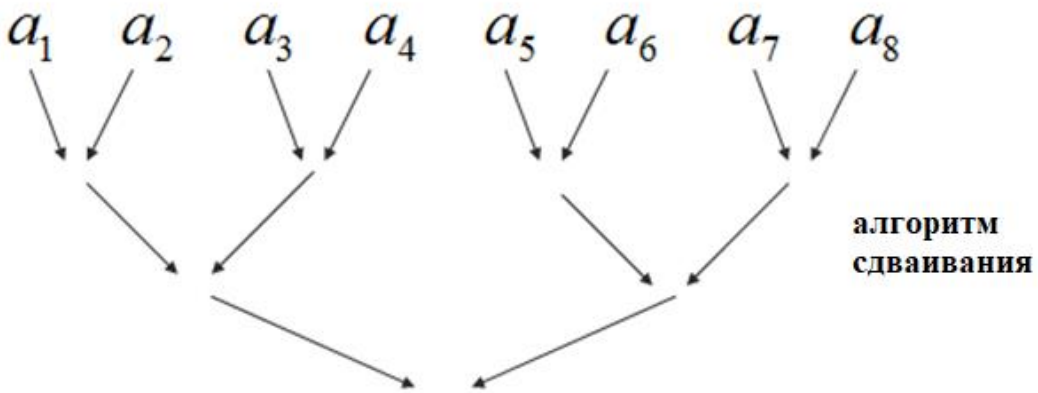


степень параллелизма n

Основные понятия параллелизма и векторизации

$$a_1 + a_2 + \ldots + a_n = \sum a_i$$

$s = a_1, \quad s \leftarrow s + a_i, \quad i = 2, \ldots, n$ непригоден для параллельного исполнения



Степень параллелизма изменяется от этапа к этапу **n/2, n/4, ….., 1**

Для алгоритма сдваивания число этапов q и число элементов n связаны соотношением

2^q = n или $q = \log_2 n$

Основные понятия параллелизма и векторизации

Опр.2 ***Средней степенью параллелизма*** численного алгоритма называется отношение общего числа операций алгоритма к числу его этапов.

Для алгоритма сдваивания $$\frac{1}{q}\left(\frac{n}{2}+\frac{n}{4}+\ldots+1\right)=\frac{2^q-1}{q}=\frac{n-1}{\log_2 n}$$

Опр.3 ***Ускорением*** параллельного алгоритма называется отношение

$$S_p=\frac{T_{\text{выполнения алгоритма на одном процессоре}}}{T_{\text{выполнения алгоритма на системе } p \text{ процессоров}}}$$

Как правило $S_p <= p$. Для тривиальной задачи сложения векторов $S_p = p$

Цель распараллеливания – достичь линейного ускорения на максимально большом числе процессоров $\boldsymbol{S_p \approx p}$

Основные понятия параллелизма и векторизации

Вопросы

Какое время брать за время выполнения последовательной программы?

-Время лучшего известного алгоритма (в смысле вычислительной сложности)?

-Время лучшего теоретически возможного алгоритма?

Ответ: **Время лучшего известного алгоритма**

Что считать временем выполнения Tp параллельной программы?

- Среднее время выполнения потоков программы?
- Время выполнения потока, завершившего работу первым?
- Время выполнения потока, завершившего работу последним?

Ответ: **Время выполнения потока, завершившего работу последним**

Основные понятия параллелизма и векторизации

Ускорение S_p сравнивает поведение данного алгоритма для одного и p процессоров. Однако **параллельный алгоритм может быть не лучшим для последовательного компьютера**

Опр.4 ***Ускорением*** параллельного алгоритма по сравнению с наилучшим последовательным алгоритмом называется отношение

$$S_p' = \frac{T_{\text{выполнения наилучшего последовательного алгоритма на одном процессоре}}}{T_{\text{выполнения параллельного алгоритма на системе p процессоров}}} = T_1/T_p$$

T_1 - время выполнения однопоточной программы а не время выполнения многопоточной программы на одном процессоре.

Если параллельная версия алгоритма эффективна, то $T_1 > T_p$
Если издержки реализации параллельной версии алгоритма большие, то $T_1 < T_p$

Основные понятия параллелизма и векторизации

Интерес представляет ускорение в пересчёте на один процессор – эффективность системы из p процессоров

Опр.5 ***Эффективностью*** параллельного алгоритма называется

$$E_p = \frac{S_p}{p}; \quad E_p' = \frac{S_p'}{p}$$

Очевидно $S_p' \le S_p \Rightarrow E_p' \le E_p$ Обычно $S_p \le p; \quad E_p \le 1$

Случай $S_p = p$ – линейное ускорение (возможность ускорения вычислений пропорционально числу процессоров)

Случай $S_p > p$ – суперлинейное ускорение (например, из-за большего коэффициента кеш-попаданий)

Основные понятия параллелизма и векторизации

Отсутствие максимального ускорения обусловлено различными факторами

1. **отсутствие максимального параллелизма в алгоритме** и/или несбалансированность нагрузки процессоров
2. обмены, конфликты памяти и затраты на времена синхронизации

Природа различная – результат один: **задержки**

Опр.6 ***Временем подготовки данных*** называются задержки, вызванные обменами, конфликтами памяти или синхронизации и необходимые для того, чтобы разместить данные, требующиеся для продолжения вычислений, в соответствующие ячейки памяти.

Основные понятия параллелизма и векторизации

Опр.7 **Масштабируемость** параллельной программы (scalability) - характеристика программы, показывающая как изменяются ее показатели производительности при варьировании числа параллельных процессов на конкретной ВС

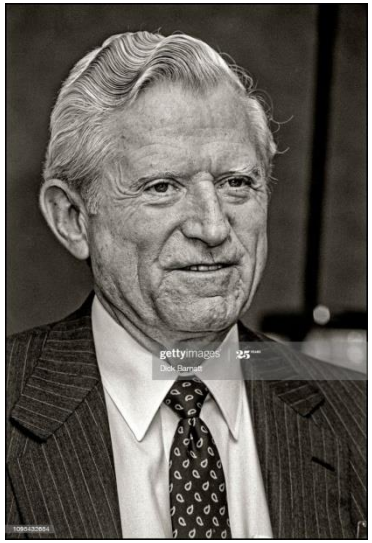
Параллельная программа (алгоритм), ускорение которой линейно растет с увеличением p называется линейно масштабируемой или просто **масштабируемой** (scalable)

Формальная модель ускорения

Большинство алгоритмов можно рассматривать как смесь фрагментов с **тремя** различными степенями параллелизма:

- максимальной,
- частичной
- минимальной.

Джин Амдал (Gene Amdahl) (1967)



Формальная модель ускорения

$$S_p = \frac{T_1}{(\alpha_1 + \alpha_2 / k + \alpha_3 / p) T_1 + t_d}$$

T_1 – время выполнения алгоритма на одном процессоре
α_1 – доля последовательных вычислений
α_2 – доля вычислений со средней степенью параллелизма k<p
α_3 – доля вычислений с максимальной степенью параллелизма p
t_d – общее время на подготовку данных
p – количество процессоров

$$\alpha_1 + \alpha_2 + \alpha_3 = 1$$

Формальная модель ускорения

Следствие 1. $\alpha_1 = \alpha_2 = 0, \alpha_3 = 1, t_d = 0$ $\quad S_p = p$

Следствие 2. $\alpha_1 = \alpha_3 = 0, \alpha_2 = 1, t_d = 0$ $\quad S_p = k$

Следствие 3. $\alpha_2 = 0, \alpha_1 = \alpha, \alpha_3 = 1- \alpha, t_d = 0$

$$S_p = \frac{1}{\alpha + (1-\alpha)/p}$$

Закон Амдаля-Уэра

Показывает влияние доли последовательных вычислений при идеальных условиях

Формальная модель ускорения

$$\lim_{p \to \infty} S_p = 1/\alpha$$

Пример:

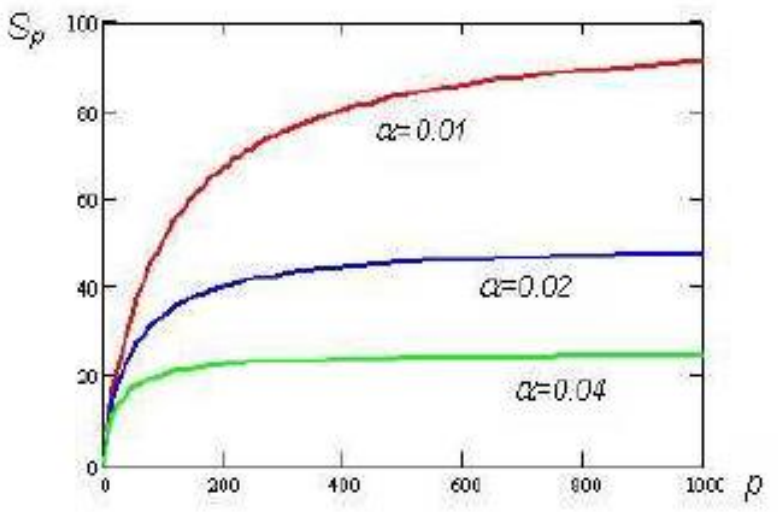
Доля последовательных вычислений 20%, → теоретически невозможно получить ускорение вычислений более чем в 5 раз

Главную роль играет доля последовательных вычислений, а не число процессоров

Формальная модель ускорения

Пример: изменение ускорения при малых α

$\alpha \backslash p$	10	100	1000
0,01	9,17	50,25	90,99
0,02	8,47	33,56	47,66
0,04	7,35	20,16	24,41



Выводы:

1. Наличие α приводит к появлению некоторого значения, больше которого ускорение быть не может
2. Рост α приводит к более резкому падению ускорения при большом количестве процессоров

Формальная модель ускорения

Следствие 4. t_d /= 0

Каковы бы не были значения α_1, α_2 , α_3 при достаточно большом t_d можно получить **S_p <1,** т.е **использование нескольких процессоров оказывается менее выгодным, чем один**

Сдерживающие факторы применения параллелизма

Применение параллелизма не получило столь широкого распространения

Возможные причины

- высокая стоимость параллельных систем

За:
Выгоднее получить требуемую вычислительную мощность приобретением одного производительного процессора (последовательная ЭВМ) , чем использование нескольких менее быстродействующих процессоров.

Против:
Рост быстродействия последовательных ЭВМ не может продолжаться бесконечно.
Влияние этого фактора снижено за счет массового выпуска типовых конструктивных элементов для параллельных вычислительных комплексов

Сдерживающие факторы применения параллелизма

- потери производительности для организации параллелизма

За:
Гипотеза Минского (Minsky): ускорение, достигаемое при использовании параллельной системы, пропорционально двоичному логарифму от числа процессоров (т.е. при 1000 процессорах возможное ускорение оказывается равным 10).

Против:
Существует большое количество задач, при параллельном решении которых достигается 100% использование всех имеющихся процессоров параллельной ВС

Сдерживающие факторы применения параллелизма

- **постоянное совершенствование последовательных компьютеров**

За:
-Закон Мура: мощность последовательных процессоров возрастает практически в 2 раза каждые 18-24 месяцев

Против:
Рост быстродействия последовательных ЭВМ не может продолжаться бесконечно.
Новый закон Мура:
число ядер удваивается каждые 18 месяцев
(вместо удвоения частоты)

Сдерживающие факторы применения параллелизма

-существование последовательных вычислений

За:

Закон Амдала

- Ускорение зависит от потенциального параллелизма задачи
- Предельное ускорение определяется свойствами задачи

В табл. 11.1 показано, на какое максимальное ускорение можно рассчитывать в зависимости от доли последовательных вычислений и числа доступных процессоров

Таблица 11.1.

Число ПЭ	Доля последовательных вычислений				
	50%	25%	10%	5%	2%
2	1,33	1,60	1,82	1,90	1,96
8	1,78	2,91	4,71	5,93	7,02
32	1,94	3,66	7,80	12,55	19,75
512	1,99	3,97	9,83	19,28	45,63
2048	2,00	3,99	9,96	19,82	48,83

Против:

Одна из самых серьезных проблем в области параллельного программирования. Доля последовательных вычислений может быть существенно снижена при **выборе более подходящих для распараллеливания алгоритмов**

Сдерживающие факторы применения параллелизма

-существующее программное обеспечение ориентировано в основном на последовательные ЭВМ

За:
Для большого количества задач уже имеется подготовленное программное обеспечение и все эти программы ориентированы главным образом на последовательные ЭВМ

Против:
Если существующие программы обеспечивают решение поставленных задач, то переработка этих программ и не является необходимой.

Сдерживающие факторы применения параллелизма

-зависимость эффективности параллелизма от учета характерных свойств параллельных систем

За:
Отсутствие мобильности для параллельных программ. Перенос параллельных алгоритмов и программ между разными типами систем становится затруднительным

Против:
«Однородность" последовательных ЭВМ также является кажущейся и их эффективное использование тоже требует учета свойств аппаратуры. Инвариантность создаваемого ПО может быть обеспечена при помощи использования типовых программных средств поддержки параллельных вычислений (типа программных библиотек MPI, PVM и др.)

Пути увеличения производительности ЭВМ

- Совершенствование элементной базы
- Совершенствование архитектуры ЭВМ
  - Совершенствование процесса выполнения инструкций
  - Смена парадигмы организации вычислений

Пути увеличения производительности ЭВМ

1. Совершенствование элементной базы

-Электромеханические реле, вакуумные лампы

-Транзисторы

-Микросхемы низкой степени интеграции

-БИС, СБИС, микропроцессоры

-***Оптические, квантовые, молекулярные процессоры***

2. Совершенствование архитектуры ЭВМ

-Совершенствование процесса выполнения инструкций

Совершенствование архитектуры набора команд (**Instruction Set Architecture –ISA**):RISC, CISC, MISC, VLIW

Параллелизм уровня инструкций (Instruction Level Parallelism – ILP):конвейерная архитектура, суперскалярная обработка, VLIW

Параллелизм уровня потоков (Thread Level Parallelism –TLP): многопроцессорные системы, одновременная многопоточность, многоядерные процессоры

Параллелизм данных (Data Parallelism): векторная обработка данных (векторные процессоры/инструкции)

- Смена парадигмы организации вычислений

Архитектура с управлением потоком команд (Control flow, классическая архитектура фон Неймана) –последовательность выполнения инструкций задана программой

Архитектура с управлением потоком данных (Data flow)–нет счетчика инструкций, команды выполняются по готовности входных данных (операндов), порядок выполнения операций заранее неизвестен

Пути увеличения производительности ЭВМ

-Совершенствование процесса выполнения инструкций

Совершенствование архитектуры набора команд (**Instruction Set Architecture –ISA**):RISC, CISC, MISC, VLIW

Параллелизм уровня инструкций (Instruction Level Parallelism – ILP):конвейерная архитектура, суперскалярная обработка, VLIW

Параллелизм уровня потоков (Thread Level Parallelism –TLP): многопроцессорные системы, одновременная многопоточность, многоядерные процессоры

Параллелизм данных (Data Parallelism): векторная обработка данных (векторные процессоры/инструкции)

Совершенствование архитектуры набора команд
(Instruction Set Architecture –ISA)

RISC архитектура (История)

80-е годы результаты исследований

1) реализация сложных команд требует увеличение сложности и размеров МПУ;

2) в откомпилированной программе операторы ЯВУ реализуются в виде пп (от 15 до 45% вычислительной нагрузки приходится на вызов/возврат);

3) половину операций в ходе вычислений составляет операция присваивания:

4) большинства программ наиболее активно используется около 20-30% сравнительно простых команд арифметики и управления

5) сравнительно небольшой набор команд можно эффективно реализовать аппаратными средствами так, что каждая операция выполнялась бы за один такт.

Анализ результатов: пересмотр традиционных архитектурных решений,
Следствие: появление *архитектуры с сокращенным набором команд* (**RISC** - Reduced Instruction Set Computer).

RISC архитектура

RISC (Reduced Instruction Set Computer) – архитектура процессора с сокращѐнным набором инструкций.

Основной **лозунг** - меньше команд, выше скорость выполнения

Основной **закон** - система команд должна содержать минимальный набор, наиболее часто используемых и наиболее простых команд.

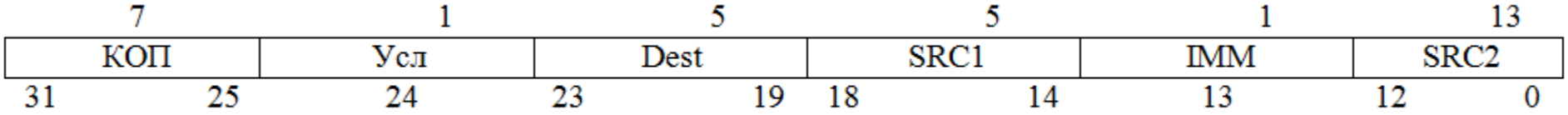
RISC архитектура

В основе RISC архитектуры 5 принципов:

- Основной набор команд реализуется **аппаратным обеспечением** (не МК).
- Любая операция должна выполняться за **один такт**, вне зависимости от ее типа.
- Все команды должны иметь **одинаковую длину и минимальное число форматов**
- Два вида формата команды: **операции обработки и обмен с памятью** Операции обработки данных реализуются только в формате «регистр - регистр»; при этом количество регистров должно быть велико
- Состав системы команд должен быть удобен для компиляции операторов ЯВУ (**компиляторы для RISC на порядок сложнее, чем компиляторы для CISC**)

RISC архитектура

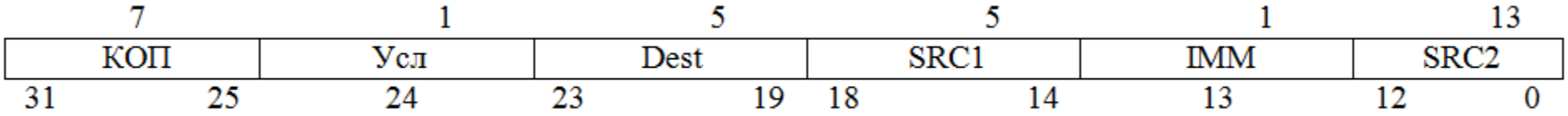
Все *операционные команды (операции обработки)* в большинстве RISC-процессоров являются трехадресными



КОП – код операции.
Усл – бит условия (для команд переходов).
если Усл =0 ⇒ признаки результата не устанавливаются
Dest – номер регистра назначения (длина пять бит – NPOH = 32).
SRC1 – номер регистра-источника 1.
SRC2 – номер регистра или непосредственного значения источника 2:
если IMM = 1 ⇒ SRC2 – непосредственное данное,
если IMM = 0 ⇒ SRC2 – регистр.

RISC архитектура

Формат *команды чтения/записи в память*



КОП – код операции.

Усл – бит условия (для команд переходов).

если Усл =0 $\Rightarrow$ признаки результата не устанавливаются

Dest – номер регистра в который при чтении заносится операнд, а при записи, из которого содержимое заносится в память (длина пять бит – NPOH = 32).

SRC1 – номер индексного регистра.

SRC2 – смещение.

RISC архитектура

При обращении к подпрограмме вместо запоминания содержимого регистров в стеке или памяти **подпрограмме выделяется новый набор регистров** (около 140 регистров)

В RISC-процессорах **увеличена аппаратная поддержка арифметических операций** благодаря уменьшению места на кристалле для размещения управляющей части процессора.

Все RISC-процессоры по размерам обрабатываемых данных **удовлетворяют стандарту ANSI**:

- с фиксированной запятой – 32 бит
- с плавающей запятой – 64 бит

Начиная с процессора Pentium, корпорация Intel начала внедрять элементы RISC-технологий в свои изделия

Вопрос

Какими достоинствами обладают RISC процессоры ?

Основные ***достоинства RISC- процессоров:***

- Повышение производительности обработки программ вычислительных задач.
- Благодаря использованию простых команд и минимума их форматов сокращается время разработки RISC-процессора.
- Улучшение технологичности RISC-процессоров благодаря большей свободе в размещении их элементов на кристалле интегральной схемы.

Вопрос

Какими недостатками обладают RISC процессоры ?

Основные ***недостатки RISC- процессоров:***

- Увеличивается **семантический разрыв** между исходным описанием и машинным кодом.
- **Сложность построения компилятора**, поскольку программа с языка высокого уровня должна транслироваться в микрокод с оптимизацией использования регистров. .
- Высокие требования к **быстродействию памяти**
- Сейчас для производительности ВС важным является требование **совместимости с созданным ранее ПО**, которому многие RISC-процессоры не удовлетворяют, что сдерживает их распространение.

CISC архитектура

CISC (англ. Complex Instruction Set Computer) - компьютер с полным набором команд - концепция проектирования процессоров.

Свойства:

- нефиксированное значение длины команды;
- небольшое число регистров, каждый из которых выполняет строго определённую функцию;
- каждая команда CISC-процессора включает в себя несколько операций;
- микропрограммная реализация набора команд
- большой набор разноформатных команд с использованием многочисленных способов адресации

CISC архитектура

Классическая архитектура процессоров, начала свое развитие с появлением первых компьютеров.

Пример: микропроцессоры семейства Pentium.
- выполняют более 200 команд разной степени сложности,
- размер команд от 1 до 15 байт
- более 10 различных способов адресации.
- число РОНов обычно составляет 8-16.

Достоинство: позволяет реализовать наиболее эффективные алгоритмы решения различных задач.
Недостаток: **усложняется структура микропроцессора (УУ), -> увеличение размеров и стоимости кристалла, снижение производительности**.

CISC архитектура

Во многих современных CISC-процессорах **используется RISC-ядро**, выполняющее обработку данных.

Сложные и разноформатные команды предварительно преобразуются в последовательность простых RISC-операций.

Пример: последние модели микропроцессоров Pentium и K7, которые по внешним показателям относятся к CISC-процессорам.

MISC архитектура

MISC (англ. minimal instruction set computer — «компьютер с минимальным набором длинных команд») — архитектура для проектирования процессора, которая отличается наилучшей эффективностью и простотой в сравнении с CISC и RISC.

Основные принципы работы
• идея укладки нескольких команд в одно большое слово (связку, bound).
• стековая вычислительная модель с ограниченным числом команд (примерно 20-30).

Причина не популярности данной архитектуры– **сложность написания программ** под различные процессоры .

VLIW архитектура

VLIW (Very long instruction word — «очень длинная машинная команда») (до 128 бит и более) - архитектура процессоров с несколькими ВУ, входящими в структуру процессора.
Компилятор перед выполнением прикладной программы проводит ее анализ и определяет группу команд, которые могут выполняться параллельно. Каждая такая группа образует одну сверхдлинную команду. Эта команда выполняется параллельно в различных ВУ.
В процессорах VLIW **задача распределения работы решается во время компиляции**.
Появилась относительно недавно - в 1990-х годах.

VLIW отличается от суперскалярности. Для суперскалярности отбор групп одновременно выполняемых команд **происходит непосредственно в ходе выполнения прикладной программы** (а не заранее), что усложняет структуру процессора и замедляет его скорость

Вопрос

Какими достоинствами и недостатками обладает **VLIW архитектура** ?

Достоинства

- VLIW сильно упрощает архитектуру процессора, перекладывая задачу распределения вычислительных устройств на компилятор
- Т.к. отсутствуют большие и сложные узлы, сильно снижается энергопотребление

Недостатки

- код для VLIW обладает невысокой плотностью
- архитектура VLIW непривычна для программиста. Программирование на уровне машинных кодов для VLIW-архитектур практически невозможно вручную. Приходится полагаться на оптимизацию компилятора, который сам может содержать ошибки.

-Совершенствование процесса выполнения инструкций

Совершенствование архитектуры набора команд (**Instruction Set Architecture –ISA**):RISC, CISC, MISC, VLIW

Параллелизм уровня инструкций **(Instruction Level Parallelism – ILP**):конвейерная архитектура, суперскалярная обработка, VLIW

Параллелизм уровня потоков (Thread Level Parallelism –TLP): многопроцессорные системы, одновременная многопоточность, многоядерные процессоры

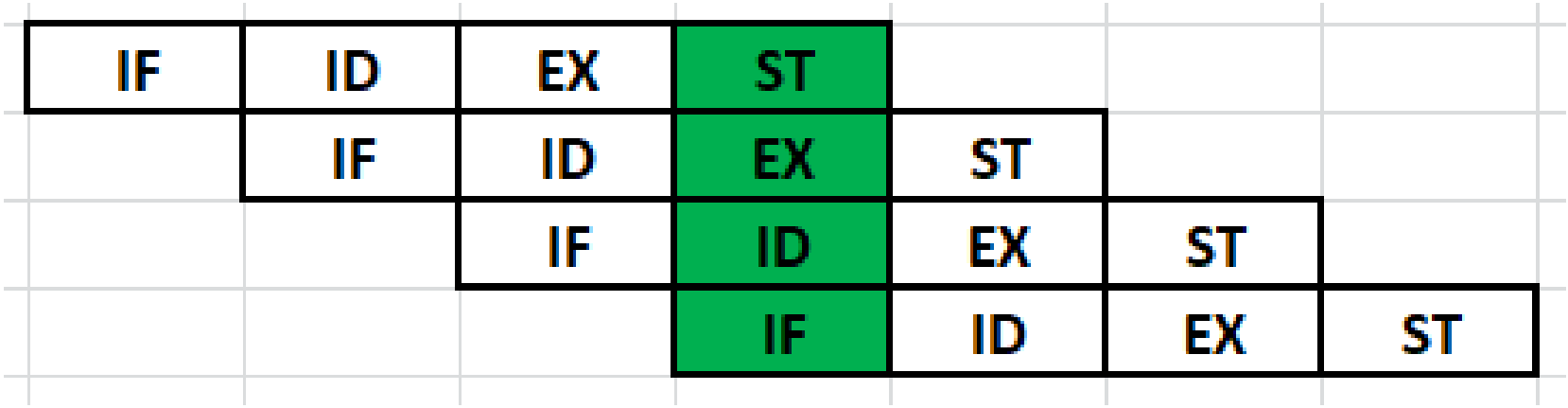
Параллелизм данных (Data Parallelism): векторная обработка данных (векторные процессоры/инструкции)

Параллелизм уровня инструкций
(Instruction Level Parallelism –ILP)

Вычислительный конвейер

1. Выборка команды (IF).
2. Декодирование инструкции (ID).
3. Исполнение операции (EX).
4. Запись результата в регистры и память (WB, ST).

Максимальное ускорение равно числу этапов конвейера



Конфликты данных

- **Текущий шаг конвейера не может быть выполнен, так как зависит от результатов выполнения предыдущего шага**
- Возможные причины:
  - **Read After Write (RAW)** – True dependency
    i1: R2 = R1 + R3
    i2: R4 = R2 + R3
  - **Write After Read (WAR)** – Anti-dependency
    R4 = R1 + R3
    R3 = R1 + R2
  - **Write After Write (WAW)** – Output dependency
    R2 = R4 + R7
    R2 = R1 + R3

Step	IF	ID	EX	ST
1	i1			
2	i2	i1		
3		i2	i1	
4			i2	i1
5				

Read After Write (RAW)
Для разрешения проблемы конвейер приостанавливается

Конфликты управления

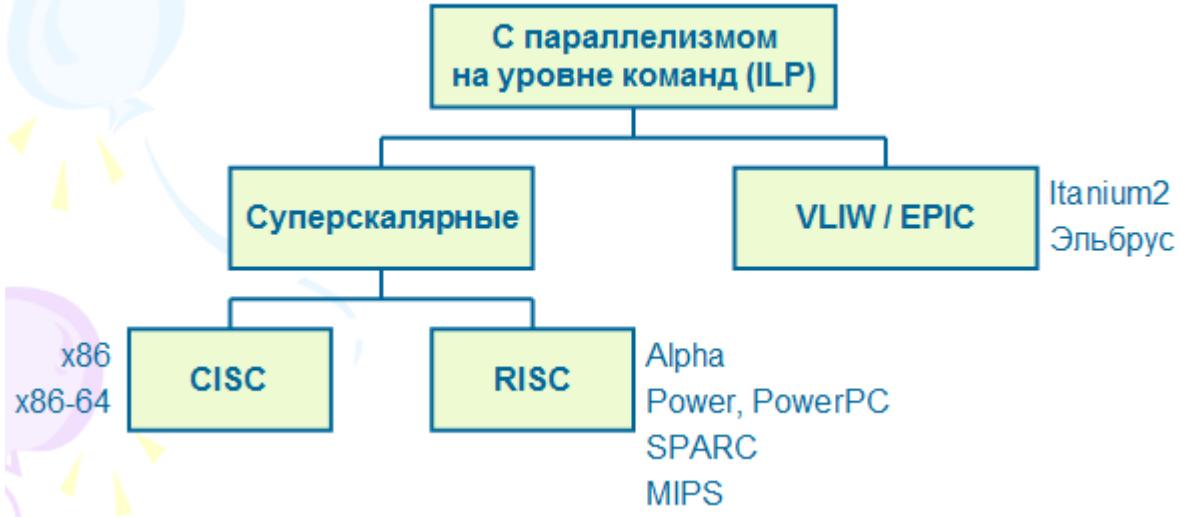
```
.text
    movl  %ebx, %eax
    cmpl  $0x10, %eax
    jne   not_equal
    movl %eax, %ecx
    jmp end

not_equal:
    movl $-0x1, %ecx

end:
```

Step	IF	ID	EX	ST
1	movl			
2	cmpl	movl		
3	jne	cmpl	movl	
4	???	jne	cmpl	movl
5				
6				
7				

Какую инструкцию выбирать из памяти (IF) на шаге 4?
(инструкция jne еще не выполнена)

В процессоре присутствует **модуль предсказания переходов** (Branch Prediction Unit – BPU), который управляет счетчиком команд

Параллелизм уровня инструкций (ILP)



ILP-процессоры
- Имеют несколько исполнительных устройств
- Могут исполнять несколько команд одновременно

Два подхода к выявлению параллелизма на уровне команд:
- аппаратные средства
- программное обеспечение

Параллелизм уровня инструкций (ILP)

ILP-процессоры

- Имеют несколько исполнительных устройств
- Могут исполнять несколько команд одновременно

Суперскалярные процессоры

• Процессор сам распределяет ресурсы

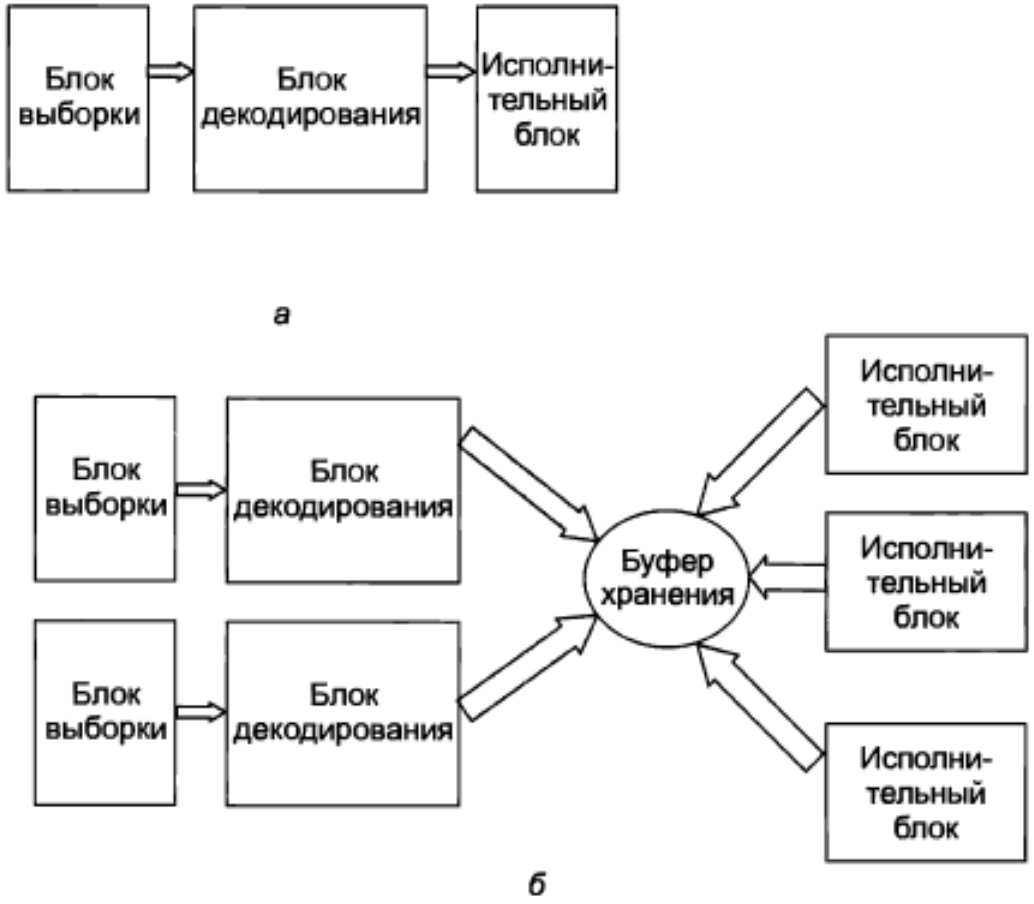
VLIW / EPIC-процессоры

Very Long Instruction Word / Explicitly Parallel Instruction Computing

• Компилятор распределяет ресурсы процессора

Цель разработчиков компилятора и процессора заключается в выявлении и получении от ILP максимально возможной выгоды

Суперскалярный процессор



Конвейер с тремя стадиями (а); суперскалярный процессор (б)

Суперскалярные процессоры (Superscalar)

- Исполняющие модули конвейера присутствуют в нескольких экземплярах(несколько ALU, FPU, Load/Store-модулей)
- За один такт процессор выполняет параллельно несколько инструкций
- Процессор динамически проверяет входные инструкции на зависимость по данным – динамическое планирование выполнения инструкций (идеи dataflow-архитектуры)
- Внеочередное исполнение команд (Out-of-order execution)– переупорядочивание команд для максимально загрузки ALU, FPU, Load/Store (выполнение инструкций по готовности их данных)

Процессоры с широким командным словом (VLIW)

VLIW (Very long instruction word — «очень длинная машинная команда») (до 128 бит и более) - архитектура процессоров с несколькими ВУ, входящими в структуру процессора.
Компилятор перед выполнением прикладной программы проводит ее анализ и определяет группу команд, которые могут выполняться параллельно. Каждая такая группа образует одну сверхдлинную команду. Эта команда выполняется параллельно в различных ВУ.
В процессорах VLIW **задача распределения работы решается во время компиляции**.
Появилась относительно недавно - в 1990-х годах.

VLIW отличается от суперскалярности. Для суперскалярности отбор групп одновременно выполняемых команд **происходит непосредственно в ходе выполнения прикладной программы** (а не заранее), что усложняет структуру процессора и замедляет его скорость

Архитектура VLIW / EPIC

VLIW – Very Long Instruction Word

EPIC – Explicitly Parallel Instruction Computing

- На входе процессора последовательность больших команд, состоящих из нескольких простых операций, которые могут исполняться параллельно.
- **Преимущества** перед суперскалярами:
  - Меньше места на процессоре тратится на управление, больше остается на ресурсы: регистры, исполнительные устройства, кэш-память.
  - Более тщательное планирование дает лучшее заполнение исполнительных устройств (больше команд за такт).
- **Недостатки**:
  - Долгое время планирования потока команд.
  - Невозможность учесть динамику исполнения программы.

Сравнение суперскалярных и VLIW/EPIC-процессоров

Суперскалярные	VLIW-EPIC
Простой компилятор, процессор планирует поток команд	**Сложный компилятор планирует поток команд**
Меньше команд за такт: • 3, 4, 5 (в среднем < 50%)	**Больше** команд за такт: • 6, 8, до 23 (в среднем > 50%)
Сложный исполнительный конвейер	**Простой исполнительный конвейер**
Меньше места на кристалле для ресурсов процессора • Исполнительные устройства • Регистры, кэш-память	**Больше** места на кристалле для ресурсов процессора • Исполнительные устройства • Регистры, кэш-память

Какие задачи управления приходится решать, чтобы процессор в суперскалярах и VLIW/EPIC работал быстро:

- При параллельном исполнении команд нужно найти независимые команды
  - **SS: Независимые команды ищет процессор**
  - **EPIC: Независимые команды ищет компилятор**

- При спекулятивное исполнение команд нужно заранее угадать, выполнится ли переход
  - **SS: Процессор автоматически предсказывает переход**
  - **EPIC: Компилятор подсказывает процессору как поступить**

Какие задачи управления приходится решать, чтобы процессор в суперскалярах и VLIW/EPIC работал быстро:

- При спекулятивной загрузке данных нужно проверить корректность преждевременной загрузки данных
  - **SS: Процессор автоматически проверяет корректность**
  - **EPIC: Компилятор использует специальную команду проверки**

- При размещении данных на регистрах Нужно оптимально использовать регистры процессора
  - **SS: Процессор автоматически отображает программные регистры на аппаратные и управляет стеком регистров**
  - **EPIC: Компилятор размещает данные на аппаратных регистрах и управляет стеком регистров с помощью специальных команд**

-Совершенствование процесса выполнения инструкций

Совершенствование архитектуры набора команд (**Instruction Set Architecture –ISA**):RISC, CISC, MISC, VLIW

Параллелизм уровня инструкций (Instruction Level Parallelism – ILP):конвейерная архитектура, суперскалярная обработка, VLIW

Параллелизм уровня потоков (Thread Level Parallelism –TLP): многопроцессорные системы, одновременная многопоточность, многоядерные процессоры

Параллелизм данных (Data Parallelism): векторная обработка данных (векторные процессоры/инструкции)

Параллелизм уровня потоков
(Thread Level Parallelism –TLP)

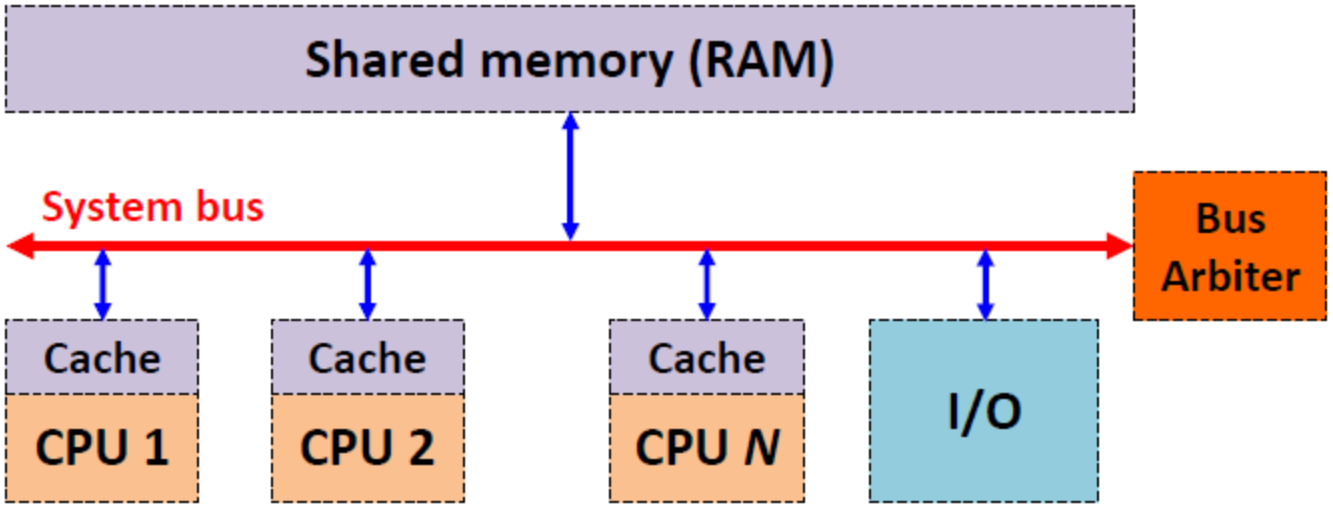
Одновременная многопоточность (Simultaneous multithreading–SMT, —технология, позволяющая выполнять инструкции из нескольких потоков выполнения (программ) на одном суперскалярном конвейере.

Потоки разделяют один суперскалярный конвейер процессора (ALU, FPU, Load/Store)

SMT позволяет повысить эффективность использования модулей суперскалярного процессора (ALU, FPU, Load/Store) за счет наличия большего количества инструкций из разных потоков выполнения

Примеры реализации:
- IBM ACS-360 (1968г.),
- Intel Pentium 4 (2002г., **Intel Hyper-Threading**, 2-way SMT)
- Intel Xeon Phi (4-way SMT),

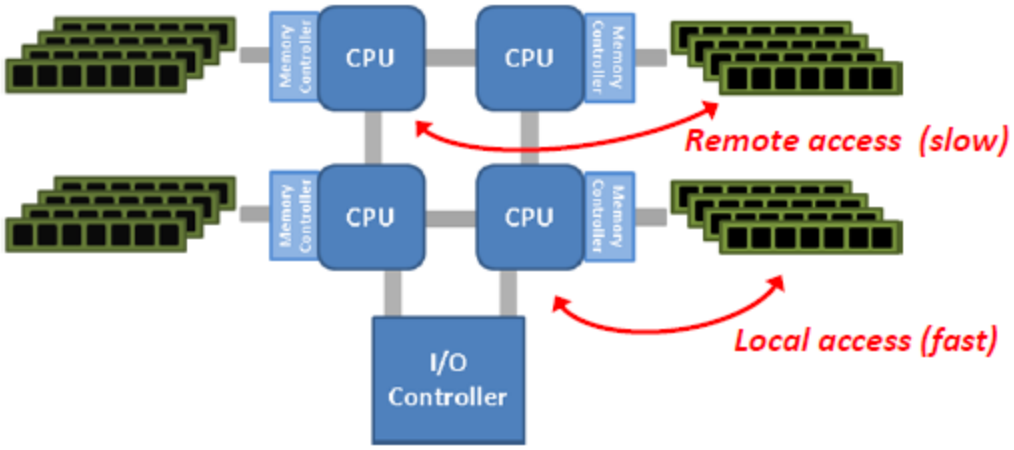
Многопроцессорные SMP-системы (Symmetric multiprocessing)



- Процессоры SMP-системы имеют одинаковое время доступа к разделяемой памяти (симметричный доступ)
- Системная шина (System bus) – это узкое место, ограничивающее масштабируемость вычислительного узла

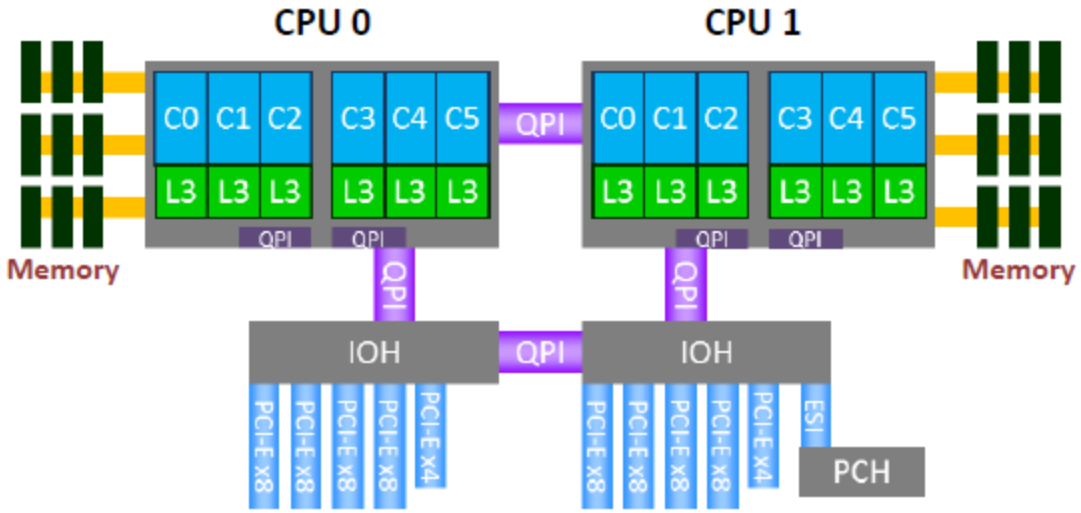
Многопроцессорные NUMA-системы (AMD)

- **NUMA** (Non-Uniform Memory Architecture) – это архитектура вычислительной системы с неоднородным доступом к разделяемой памяти
- Процессоры сгруппированы в NUMA-узлы со своей локальной памятью
- Доступ к локальной памяти NUMA-узла занимает меньше времени по сравнению с временем доступом к памяти удаленных процессоров



- 4-х процессорная NUMA-система
- Каждый процессор имеет интегрированный контроллер и несколько банков памяти
- Процессоры соединены шиной **Hyper-Transport** (системы на базе процессоров AMD)
- Доступ к удаленной памяти занимает больше времени (для Hyper-Transport ~ на 30%, 2006 г.)

Многопроцессорные NUMA-системы (Intel)



Intel Nehalem based systems with QPI
2-way Xeon 5600 (Westmere) 6-core, 2 IOH

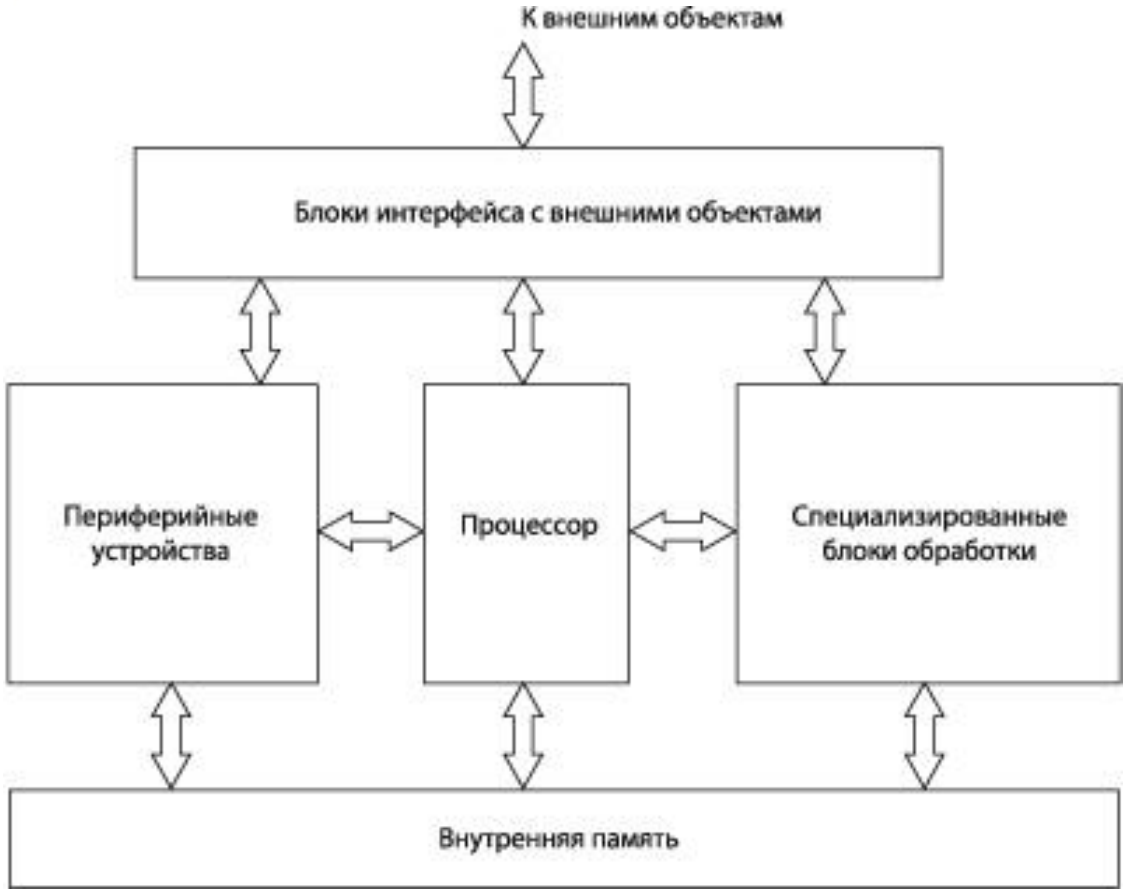
- 4-х процессорная NUMA-система
- Каждый процессор имеет интегрированный контроллер и несколько банков памяти
- Процессоры соединены шиной Intel QuickPath Interconnect (QPI) – решения на базе процессоров Intel

Многоядерные процессоры (Multi-core processors) (CMP – Chip MultiProcessor)

- Процессорные ядра размещены на одном чипе (Processor chip)

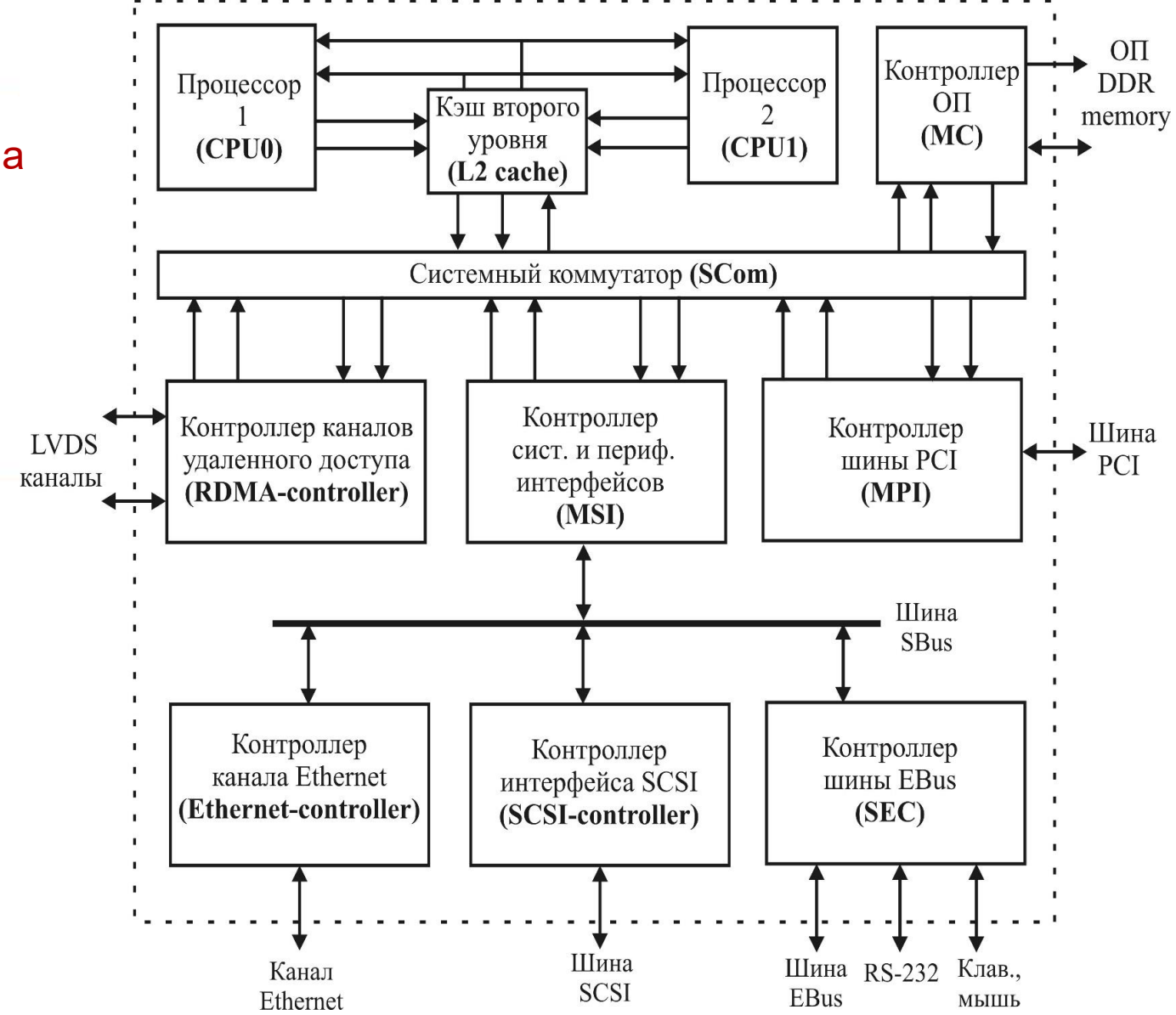
- Ядра процессора могут разделять некоторые ресурсы (например, кеш-память)

Системы на кристалле.



СНК — это СБИС, интегрирующая на кристалле различные функциональные блоки, которые образуют законченное изделие для автономного применения в электронной аппаратуре.

Пример: Система на кристалле «МЦСТ-R500S»



Многоядерные процессоры (CMP) с поддержкой SMT

- Многоядерный процессор может поддерживать одновременную многопоточность(Simultaneous multithreading –SMT, Intel Hyper-threading)

- Каждое ядро может выполнять несколько потоков на своем суперскалярном конвейере (2-way SMT, 4-way SMT, 8-way SMT)

Современные системы на базе многоядерных процессоров

-Apple iPhone 6
-Samsung Galaxy S4
-Специализированные ускорители:
Intel Xeon Phi
GPU –Graphics Processing Unit

Специализированные многоядерные процессоры

-Sony Playstation3 IBM Cell
-Microsoft XBox360 IBM Xenon
-Cisco Routers MIPS

-Совершенствование процесса выполнения инструкций

Совершенствование архитектуры набора команд (**Instruction Set Architecture –ISA**):RISC, CISC, MISC, VLIW

Параллелизм уровня инструкций (Instruction Level Parallelism – ILP):конвейерная архитектура, суперскалярная обработка, VLIW

Параллелизм уровня потоков (Thread Level Parallelism –TLP): многопроцессорные системы, одновременная многопоточность, многоядерные процессоры

Параллелизм данных (**Data Parallelism**): векторная обработка данных (векторные процессоры/инструкции)

Параллелизм уровня данных
(Data Parallelism –DP)

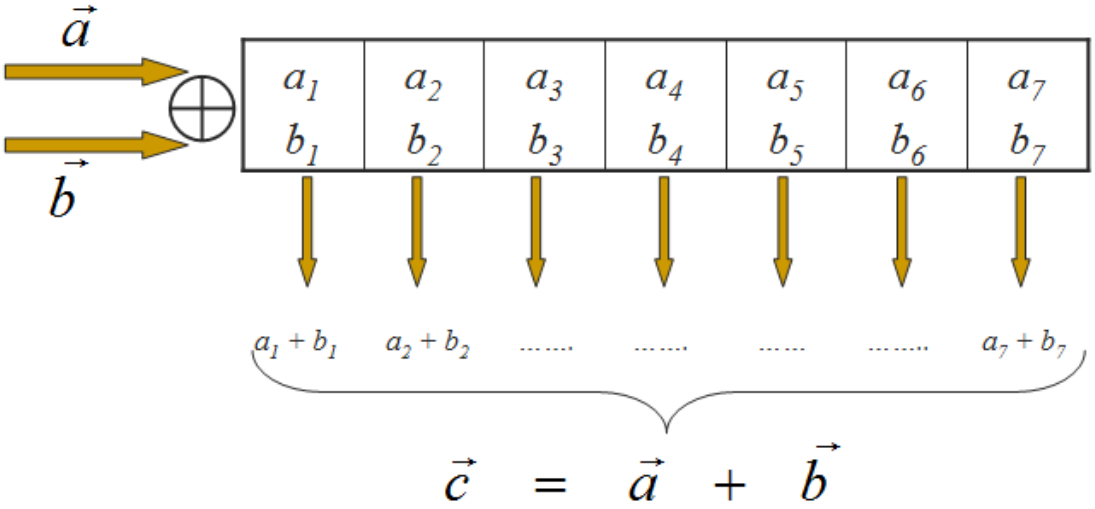
Векторные процессоры

Векторный процессор - процессор, поддерживающий на уровне системы команд операции для работы с векторами (есть аппаратная реализация работы с векторами):

Это подкласс SIMD систем (по Флинну)

SIMD-инструкции в современных процессорах
- Intel MMX (1996)
- Motorola PowerPC,
- AMD 3DNow! (1998)
- Intel SSE (Intel Pentium III, 1999)
- Intel SSE2, SSE3, SSE4

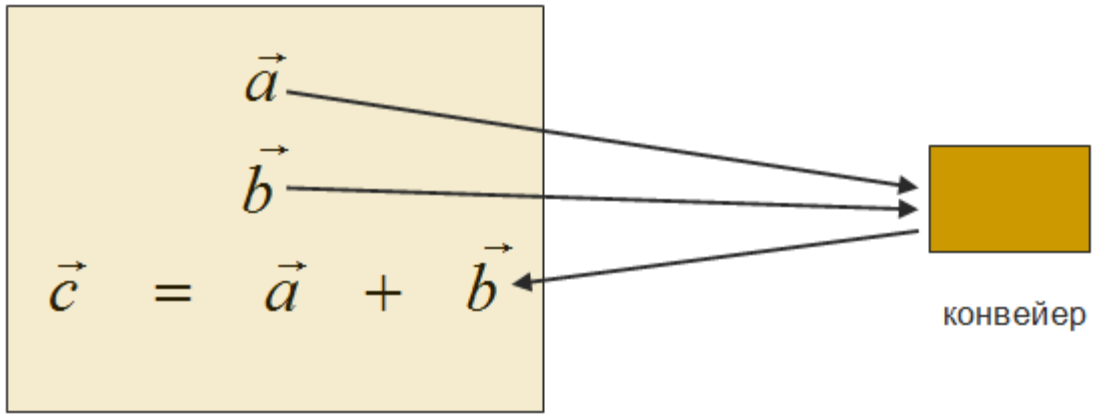
Основа векторных вычислений – концепция ***конвейеризации***
т.е. явное сегментирование арифметического устройства



Для реализации возможности ускорения нужно **подавать данные в АУ достаточно быстро.**

Предложены 2 основные формы организации подачи данных на конвейер:
- процессор типа «память-память»
- процессор типа «регистр-регистр»

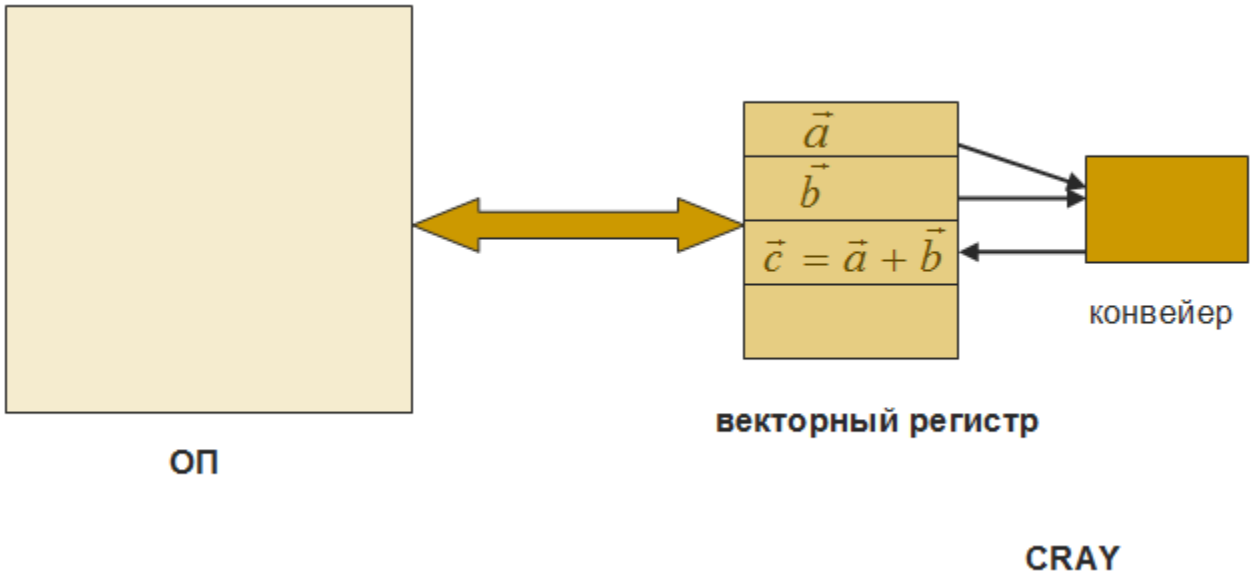
Процессор типа «память-память»



Control Data Corporation – CDC

CYBER 203, CYBER 205

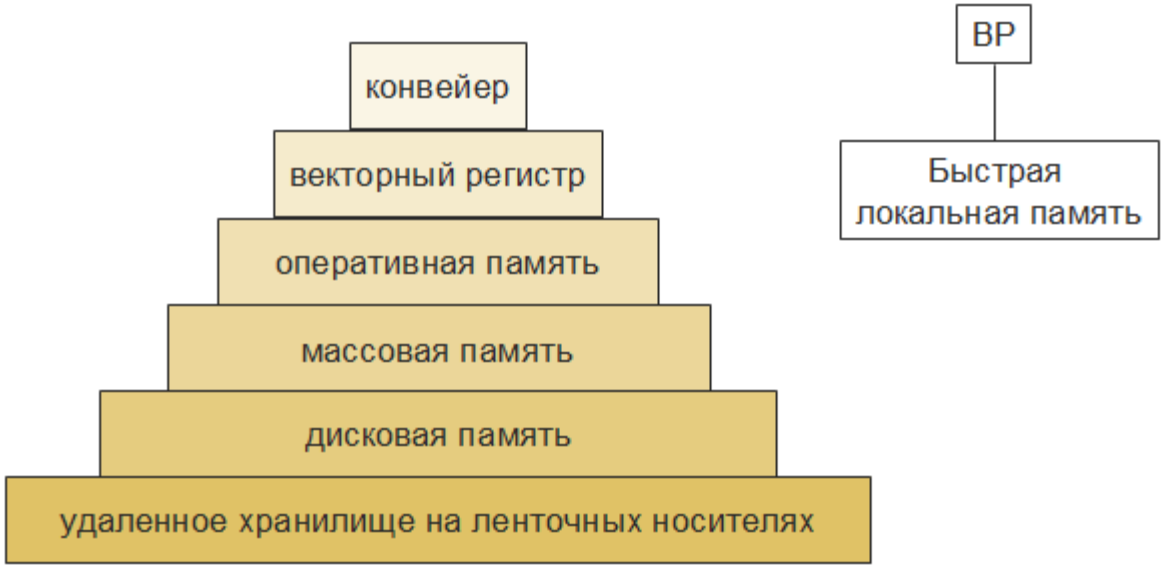
Процессор типа «регистр-регистр»



CRAY

Усложнение схемы перемещения данных на конвейер привело к появлению дополнительных требований к алгоритму

Векторные регистры – аналог кэш-памяти компьютера в иерархия памяти



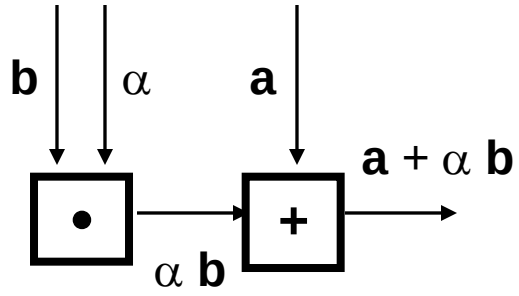
Конвейер

Конвейеризированное арифметическое устройство разной конструкции:

- **CDC** – изменяемая конфигурация
- **CRAY** – специализированные устройства

Векторные операции

1. сложение
2. поэлементное умножение
3. **зацепление**
4. покомпонентное деление
5. нахождение обратного значения
6. длина вектора
7. скалярное произведение
8. ……

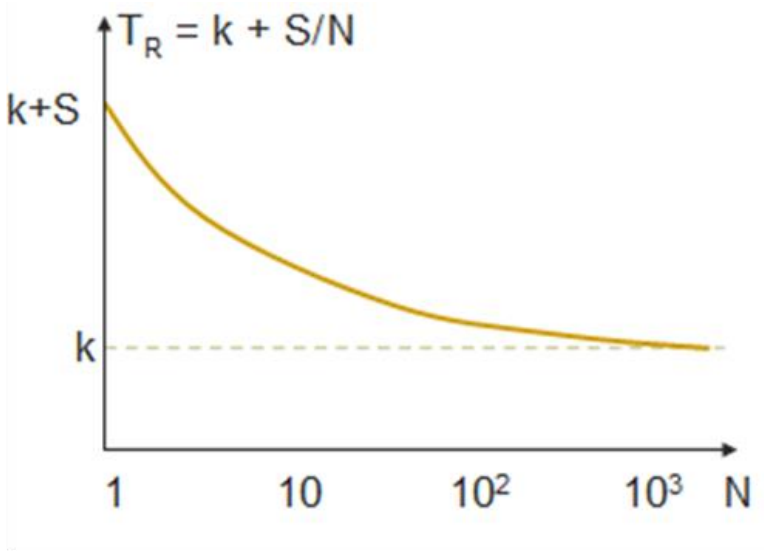


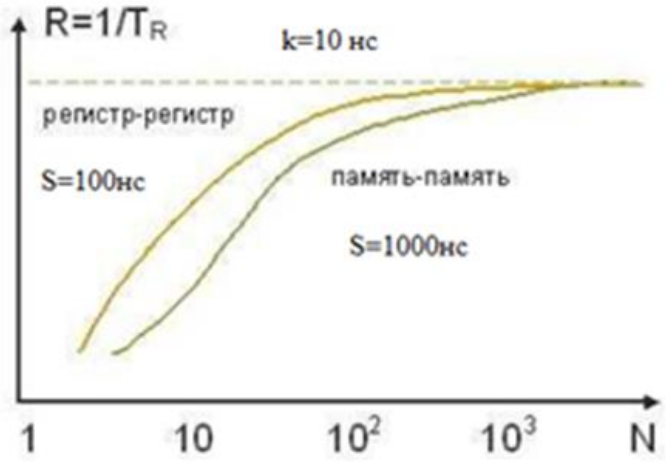
Производительность конвейера

$T_R = k + S/N$ – среднее время получения одного результата

S – время запуска – время, необходимое для заполнения конвейера
k – время получения результата на конвейере
N - длина обрабатываемого вектора

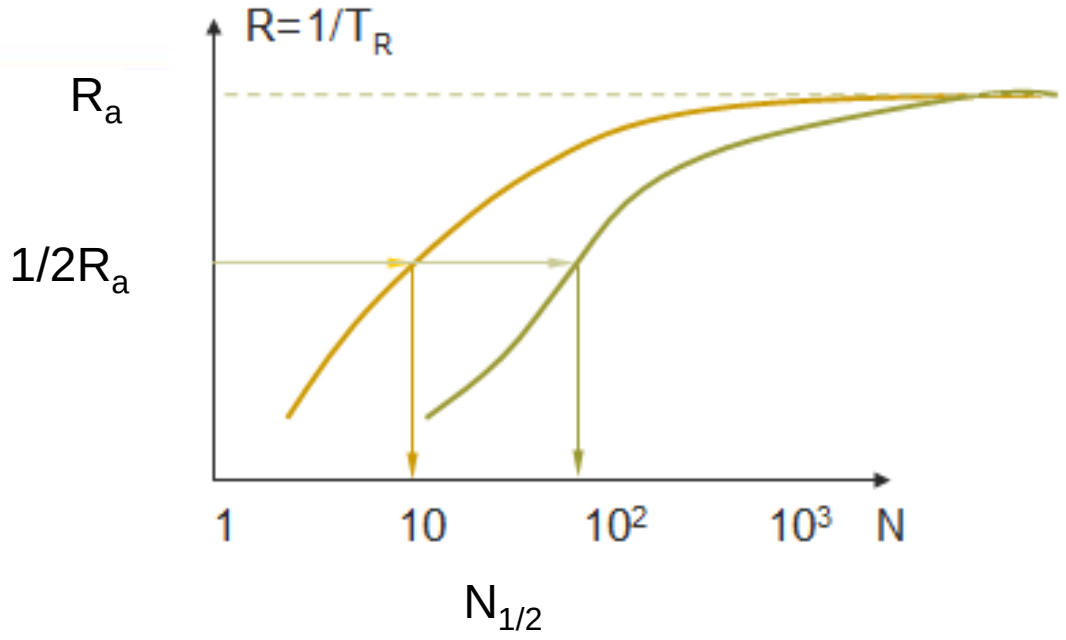
$R = 1/T_R$ – производительность конвейера





Величины, характеризующие производительность конвейера

1. $N_{1/2}$ – длина вектора, для которой достигается **половина асимптотической производительности**



2. N_c – граничная длина - длина вектора, при которой ***векторная арифметика* сравнивается по производительности со *скалярной арифметикой***

Примеры:

а) пусть скалярную арифметику можно выполнить со скоростью 1 гигафлоп. Тогда Nc – значение N, при котором **R= 1 гигафлоп**

Пусть **S=10нс и k=0.1нс**. Тогда из формулы

$$R = T_R^{-1} = \frac{N}{S + kN}$$

имеем

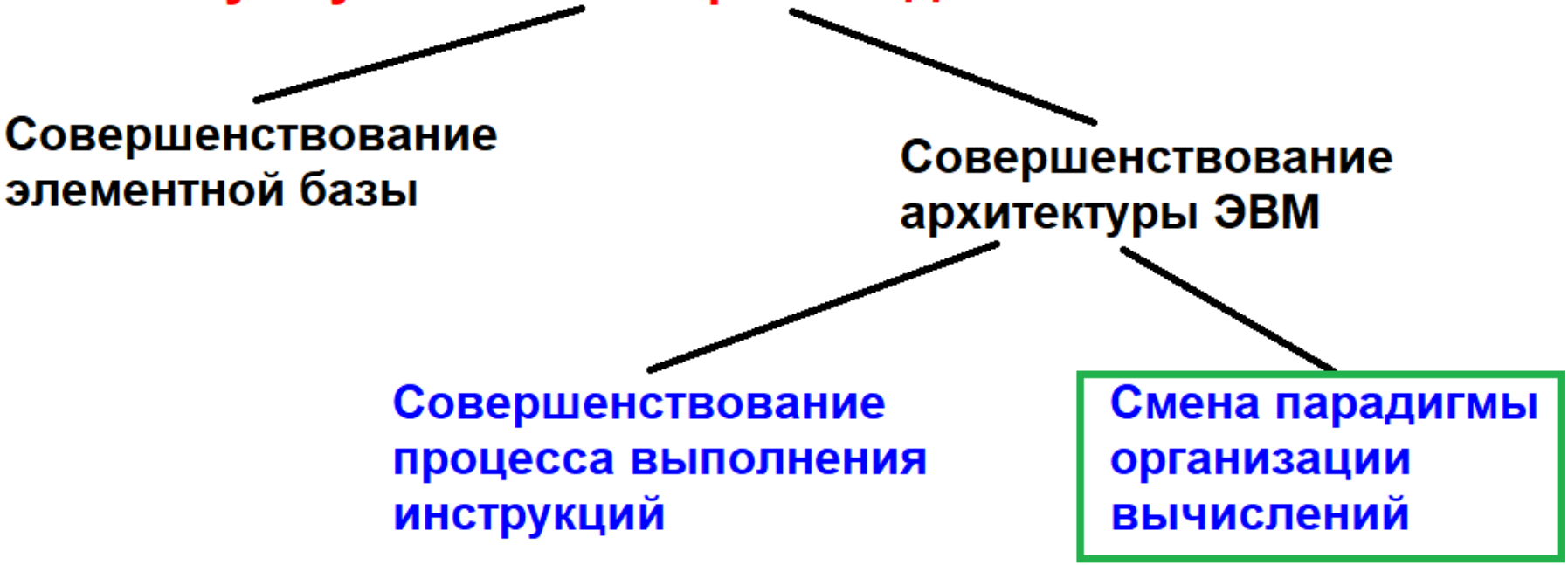
$$\frac{N}{(10 + 0.1N)10^{-9}} \geq 1 \times 10^9$$

и получаем **Nc =11**

Итак, для векторов длины меньше 11 векторная арифметика медленнее скалярной

б) пусть S=1нс . Тогда Nc =2 и векторные операции эффективнее для всех длин

Пути увеличения производительности ЭВМ



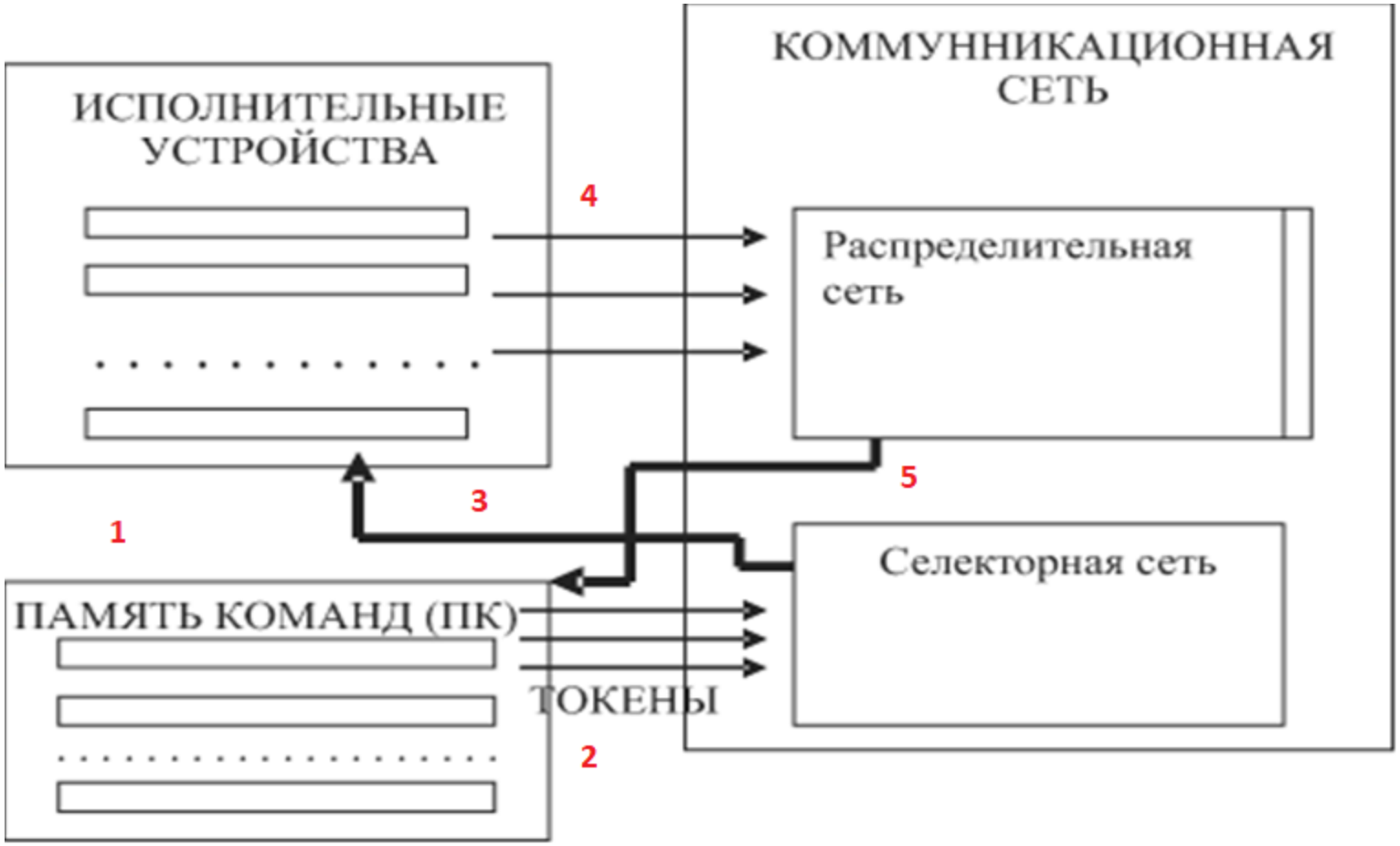
- Смена парадигмы организации вычислений

Архитектура с управлением потоком команд (Control flow, классическая архитектура фон Неймана) –последовательность выполнения инструкций задана программой

Архитектура с управлением потоком данных (Data flow)–нет счетчика инструкций, команды выполняются по готовности входных данных (операндов), порядок выполнения операций заранее неизвестен

Архитектура с управлением потоком данных
(Data flow)

ВС управляются *потоком данных (data flow)*.



1. Программа размещается в **памяти команд (ПК)**, которая является ассоциативной памятью.
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.

2. 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.
После выдачи токена в селекторную сеть операнды в тексте команды уничтожаются, что обеспечивает повторное выполнение команды в цикле, если это необходимо.

3. Токен поступает по сети на одно из **исполнительных устройств**, где операция выполняется.

4. В **распределительную сеть** выдается результирующий пакет, содержащий результат вычислений и адреса назначения в ПК.

5. По этим адресам результат поступает в ПК, создавая возможность активизации новых ячеек.

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
- параллельной работы многих исполнительных устройств.

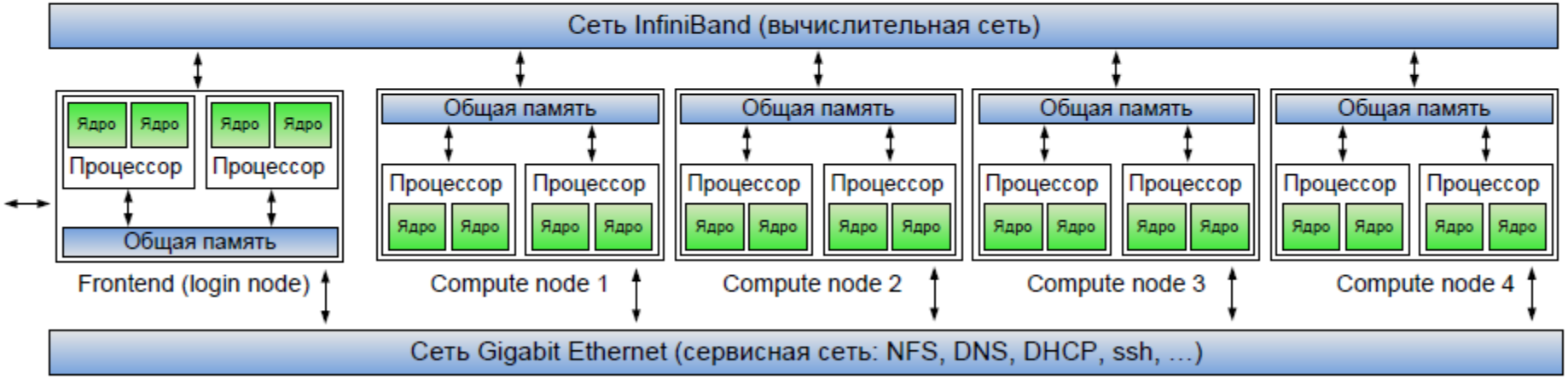
Вопросы?

Дополнение

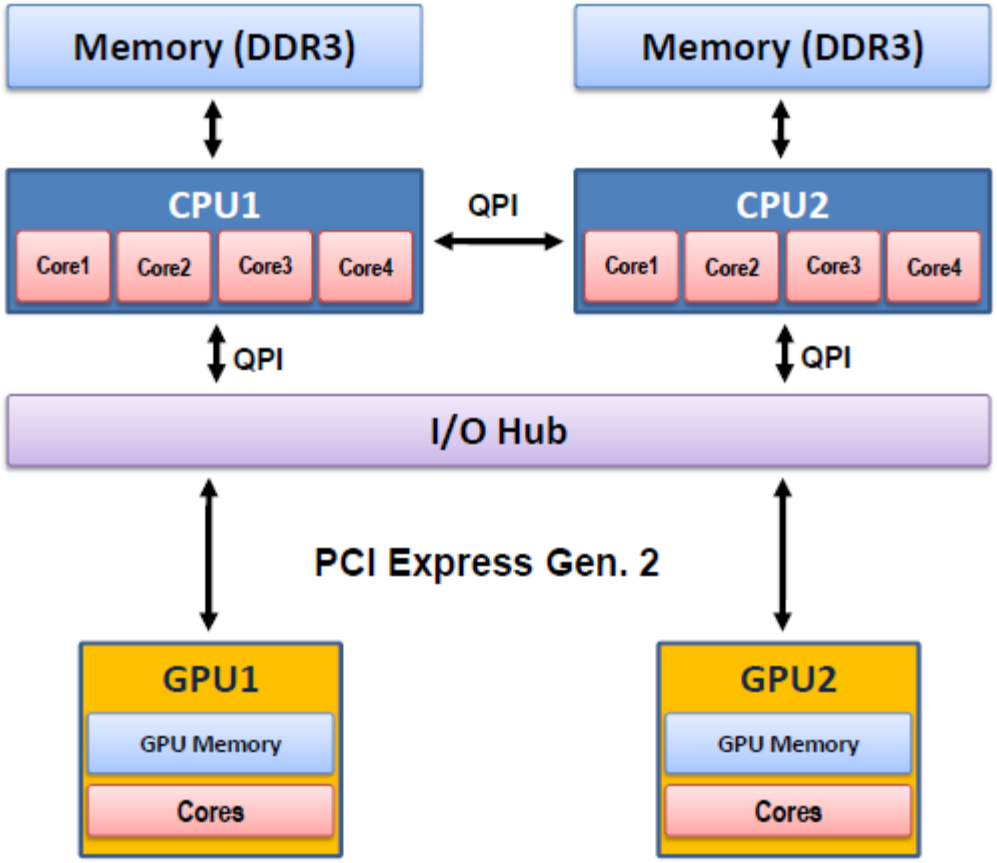
Распределенные вычислительные системы (Distributed Computer Systems)

Saint Petersburg Electrotechnical University

Вычислительные кластеры



Гибридные вычислительные узлы



Формальная модель ускорения для кластеров
(плоская решетка, трехмерный и четырехмерные кубы)

Пусть
T1 – время выполнения алгоритма на одном процессоре
α– доля последовательных вычислений
1- α - доля распараллеленных вычислений на p процессорах

Время выполнения алгоритма на системе из p процессоров

$$T_p = \alpha T_1 + \frac{1-\alpha}{p} T_1,$$

Тогда ускорение

$$S_p = \frac{T_1}{T_p} = \frac{p}{1+\alpha(p-1)},$$

что соответствует закону Амдаля