



# **НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО**

## **ЦЕНТР КОМПЕТЕНЦИЙ ONEAPI В ННГУ**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ**



**НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. Н.И. ЛОБАЧЕВСКОГО**  
**ЦЕНТР КОМПЕТЕНЦИЙ oneAPI в ННГУ**  
**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ**

***ВВЕДЕНИЕ В АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ  
И ОПТИМИЗАЦИЮ ПРОГРАММ***

**Раздел I. Архитектурные механизмы,  
влияющие на производительность.  
Уровни параллелизма.**

# Содержание

---




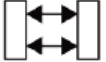
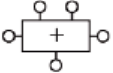




- ❑ Архитектура современных вычислительных систем
- ❑ Конвейерная обработка инструкций
- ❑ Суперскалярность и векторизация
- ❑ Иерархия памяти
- ❑ NUMA-архитектура
- ❑ Архитектура GPU

# АРХИТЕКТУРА СОВРЕМЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

# Архитектура вычислительных систем

## Уровни абстракции...

- ❑ **Физика (Physics)** – поведение электронов в ЭВМ описывается квантовой механикой и уравнениями Максвелла (теоретическая физика)
- ❑ **Устройство (Devices)** – полупроводниковые устройства, такие как транзисторы (физика)
- ❑ **Аналоговая схема (Analog Circuits)** – полупроводниковые устройства соединены в функциональные компоненты, такие как усилители и т.п. (физика)
- ❑ **Цифровая схема (Digital Circuits)** – уровень логических вентилей (logic gates), которые используют строго ограниченное число дискретных уровней напряжения (проектирование схем, логика)




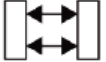
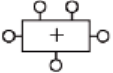




Application Software		Programs *
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons

\*Источник: Дэвид М. Харрис и Сара Л. Харрис. Цифровая схемотехника и архитектура компьютера.

# Архитектура вычислительных систем

## Уровни абстракции...

- ❑ **Логический уровень (Logic)** – комбинированная логика, объединяющая набор вентилей в отдельные схемы, в том числе базовые блоки, такие как сумматоры и т.п. (проектирование схем, математическая логика и теория алгоритмов)
- ❑ **Микроархитектура (Microarchitecture)** – соединение цифровых элементов в логические блоки, выполняющие определённые команды. Способ связи разработчиков «железа» и программистов (проектирование схем, методы оптимизации)
  - Связь логического и архитектурного уровней




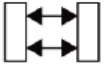
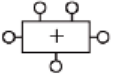




Application Software		Programs *
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons

\*Источник: Дэвид М. Харрис и Сара Л. Харрис. Цифровая схемотехника и архитектура компьютера.

# Архитектура вычислительных систем

## Уровни абстракции

- ❑ **Архитектура (Architecture)** – описание компьютера с точки зрения программиста как некоторого набора ресурсов и команд (x86)
- ❑ **Операционная система (Operating systems)** – управление операциями нижнего уровня, такими как доступ к памяти и т.д. (программирование, в т.ч. низкоуровневое)
- ❑ **Программное обеспечение (Application software)** – решение конкретных прикладных задач (программирование, как правило, высокоуровневое)

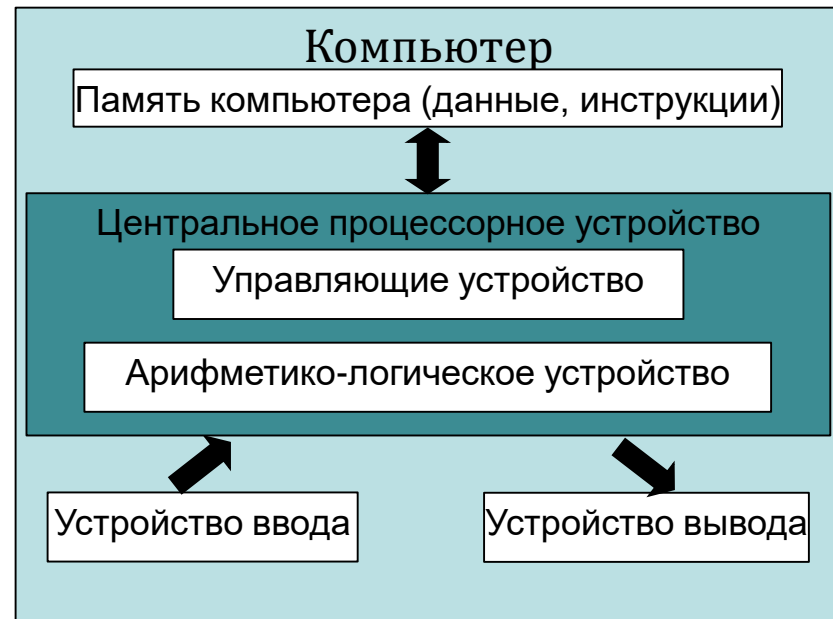
Application Software		Programs *
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons

\*Источник: Дэвид М. Харрис и Сара Л. Харрис. Цифровая схемотехника и архитектура компьютера.

# Архитектура вычислительных систем

## Архитектура фон Неймана

- Принципы машины фон Неймана:
  - Использование двоичной системы счисления
  - Программа хранится в памяти вместе с данными (однородность памяти)
  - Ячейки памяти имеют адреса в виде набора последовательных положительных целых чисел (адресность)
  - Программное управление
  - Возможность выполнения условного перехода при выполнении программы



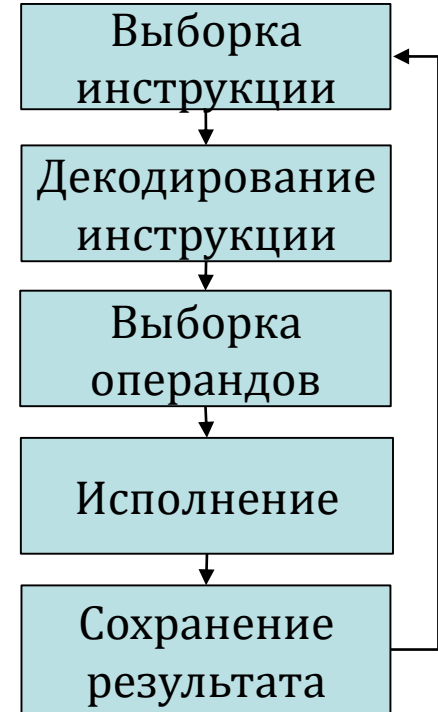
- Модель вычислений подразумевает последовательное выполнение команд



# Архитектура вычислительных систем

## Шаги обработки инструкций

- ❑ Выбор следующей инструкции программы из памяти, на неё указывает программный счетчик
- ❑ Определение размера инструкции и требуемых действий
- ❑ Нахождение и получение данных операндов
- ❑ Исполнение инструкции – вычисление значения ее результата
- ❑ Запись результата в запоминающие устройство для дальнейшего использования



# Архитектура вычислительных систем

## Архитектура системы команд

---

- ❑ Организация программируемых мест хранения (память, регистры)
- ❑ Типы и структуры данных
- ❑ Набор инструкций
- ❑ Кодирование инструкций
- ❑ Режимы адресации и доступа к элементам данных и инструкциям
- ❑ Условия возникновения исключений

# Архитектура вычислительных систем

## Набор инструкций

Тип операции	Примеры операции	Возможная стоимость в тактах
Арифметико-логический	Операции целочисленной арифметики и логики	1 (Add, Or ..), 1-7 (Mul), 10-50 (Div)
Передача данных	Загрузить, сохранить, переместить	1-4 (L1), 10-16 (L2), 30-70 (L3), 100-150 (RAM), 100-300 (NUMA L3), 300-500 (NUMA RAM)
Управляющие	Ветвление, переходы, вызовы процедур и возвраты, прерывания	1-2 («правильный» if), 10-20 («ошибочный» if), 15-60 (вызов функции)
Вещественные	Операции с плавающей запятой	1-7 (Add, Mul), 10-40 (Div)
Системные	Системные вызовы ОС, инструкции по управлению виртуальной памятью	200-500 (резервирование и освобождение памяти), 2000-10000+ (смена контекста)
Векторные	Операции, применяемые ко многим данным (SSE, AVX)	1-10 (Add, Mul), 10-70 (Div)

# Архитектура вычислительных систем

## Частота использования инструкций

Ранг	Инструкция	Процент от всех выполняющихся
1	Load	22%
2	Conditional branch	20%
3	Compare	16%
4	Store	12%
5	Add	8%
6	And	6%
7	Sub	5%
8	Mov register->register	4%
9	Call	1%
10	Return	1%
	<b>Всего</b>	<b>96%</b>

# Архитектура вычислительных систем

## Пример программы

- Скалярное произведение,  $N = 16$

```
double A[N], B[N], Res;  
Res = 0.0;  
for( int i = 0; i < N; i ++ ){  
    Res = Res + A[i] * B[i];  
}
```

- Типичные варианты операций в цикле

- $C[i] = A[i] + B[i];$
- $C[i] = A[i] + F * B[i];$
- $D[i] = A[i] + B[i] * C[i];$

# Архитектура вычислительных систем

## Пример программы

□ Скалярное произведение – код, выполняемый процессором

```
MOV    0.0, F1           // Инициализация суммы
MOV    0, R1              // Инициализация индекса i
MOV    16, R2             // R2 = N = 16
LEA    (A), R3            // R2 содержит адрес начала массива A в памяти
LEA    (B), R4            // R3 содержит адрес начала массива B в памяти
LOAD   [R3+8*R1], F2      // Загрузка A[i] из памяти в регистр ЦП
LOAD   [R4+8*R1], F3      // Загрузка B[i] из памяти в регистр ЦП
DMUL   F2, F3, F4         // F4 = F2 * F3
DADD   F1, F2, F1         // F1 = F1 + F2
INC     R1                // i ++
CMP     R1, R2            // Сравнение i и N
JL      -6                // К загрузке следующего A[i]
ST      F1, [C]           // Сохранение суммы в память
```

# КОНВЕЙЕРНАЯ ОБРАБОТКА ИНСТРУКЦИЙ. ПАРАЛЛЕЛИЗМ УРОВНЯ ИНСТРУКЦИЙ (ILP)

# Конвейер команд

## Упорядоченный конвейер

- ❑ Конвейерная обработка инструкций – это метод реализации CPU, при котором разные шаги обработки нескольких инструкций могут выполняться одновременно (перекрываться)
  - Конвейерная обработка инструкций использует/реализует параллелизм уровня инструкций (Instruction-Level Parallelism, ILP)
- ❑ Конвейеризация увеличивает пропускную способность CPU – среднее число инструкций, завершенных за такт
  - В идеальном случае происходит завершение одной инструкции за машинный такт
- ❑ Конвейеризация не сокращает время выполнения отдельной инструкции (также называемое временем задержки завершения инструкции, латентность)
  - Минимальное время задержки завершения инструкции –  $n$  тактов, где  $n$  – число ступеней конвейера
- ❑ Описанный конвейер называется **упорядоченным (in-order) конвейером**, так как инструкции обрабатываются или исполняются в порядке, указанном в исходной программе

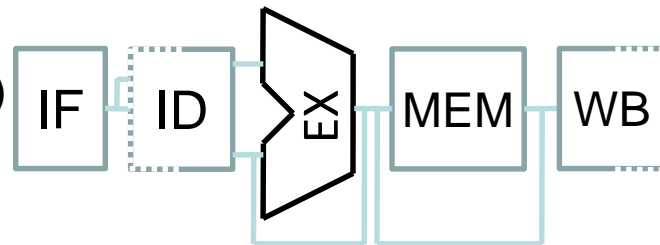


# Конвейер команд

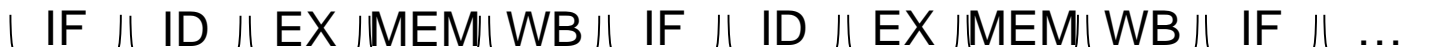
## Исполнение команд

- Шаги обработки инструкций называются ступенями (этапами, стадиями). В качестве примера мы будем рассматривать конвейер MIPS из 5 ступеней (в другом ЦП их может быть любое количество):

- IF – выборка инструкции (Instruction Fetch)
- ID – декодирование инструкции (Instruction Decode)
- EX – исполнение (Execution)
- MEM – обращение к памяти (Memory Access)
- WB – запись результата (Write Back)



- Инструкции могут обрабатываться пошагово одна за другой:



- На момент работы ступени исполнения инструкции (EX) аппаратные компоненты, ответственные за выборку (IF) и декодирование (ID), свободны. Почему бы их не использовать?

# Конвейер команд

## Конвейерное исполнение команд

- Использование конвейера позволяет одновременно выполнять несколько инструкций на разных ступенях, сохраняя порядок их исполнения и завершения

Завершение инструкции K

Инструкция\Время в тактах	1	2	3	4	5	6	7	8	9
Инструкция K	IF	ID	EX	MEM	WB				
Инструкция K+1		IF	ID	EX	MEM	WB			
Инструкция K+2			IF	ID	EX	MEM	WB		
Инструкция K+3				IF	ID	EX	MEM	WB	
Инструкция K+4					IF	ID	EX	MEM	WB

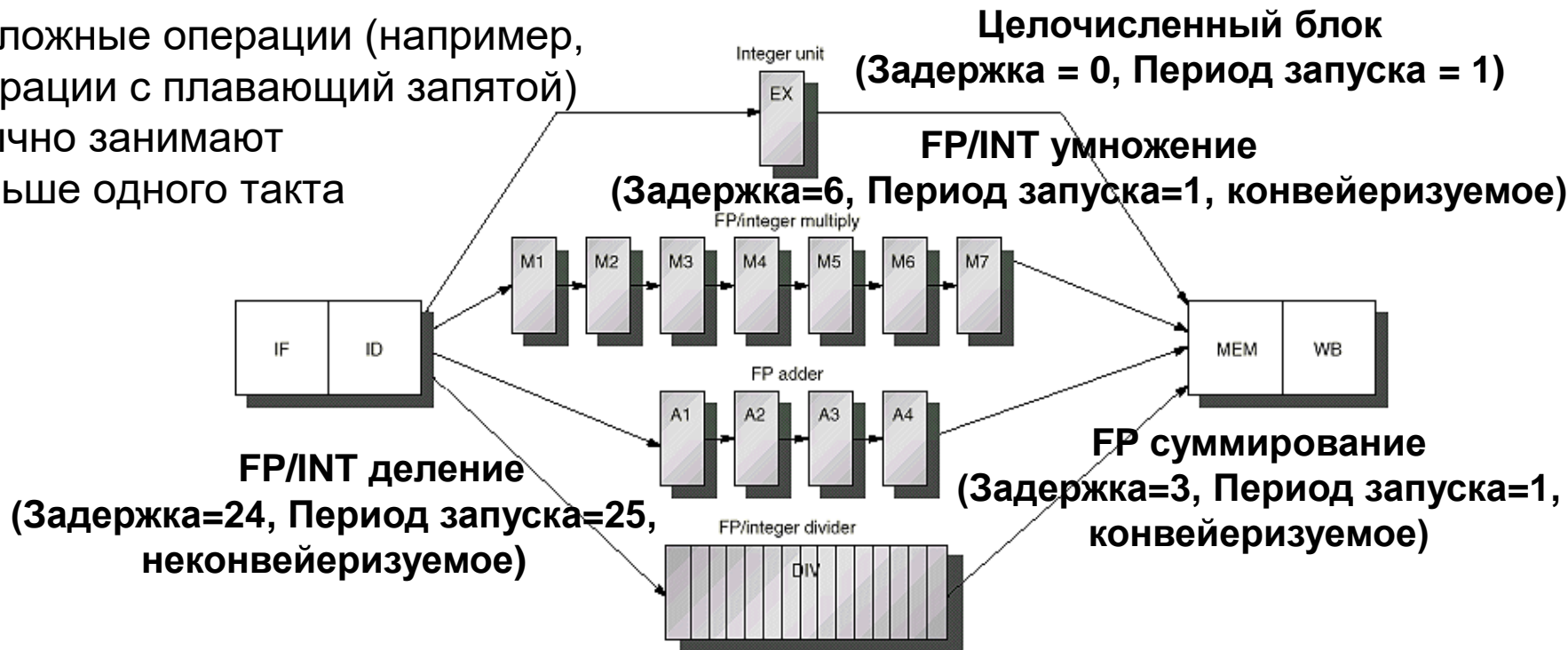
Число тактов заполнения (время разгона) = число стадий – 1 = 4

Завершение инструкции K+4

# Конвейер команд

## Многотактный конвейер вещественных операций

❑ Сложные операции (например, операции с плавающей запятой) обычно занимают больше одного такта

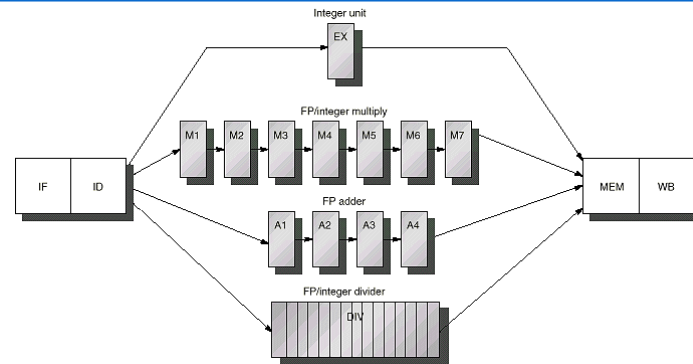


❑ Разное время выполнения команд и ограниченность ресурсов может вызвать ряд конфликтов

# Конвейер команд

## Многотактный конвейер вещественных операций

- Ступень исполнения сложных инструкций (например, операций с плавающей запятой) может занимать больше одного такта – и для многих операций в свою очередь может допускать конвейеризацию



Операция	Задержка	Период запуска	Конвейеризуемость
Целочисленные операции	0	1	–
FP/INT умножение	6	1	Да
FP суммирование	3	1	Да
FP/INT деление	24	25	Нет

- Разное время выполнения команд и ограниченность ресурсов могут быть дополнительной причиной **конфликтов**

# Конвейер команд

## Латентность и пропускная способность некоторых операций

Instruction\DisplayFamily_DisplayModel	Latency 06_3AH	Latency 06_2AH, 06_2DH	Throughput 06_3AH	Throughput 06_2AH, 06_2DH
VMOVDDUP ymm1, ymm2		1		1
VMULPD/PS ymm1, ymm2, ymm3		5		1
VSUBPD/PS ymm1, ymm2, imm		3		1
VDIVPD ymm1, ymm2, ymm3	35	45	28	44
VDIVPS ymm1, ymm2, ymm3	21	29	14	28
VSQRTPD ymm1, ymm2	35	45	28	44
VMULPD/PS ymm1, ymm2, ymm3		5		1
VRSQRTPS ymm1, ymm2		7		1
FSQRT EP		43		X87 FPU
F2XM1		90-150		X87 FPU
FCOS		190-240		X87 FPU
FPATAN		150-300		X87 FPU

# Конвейер команд

## Конфликты в конвейере

- ❑ Основное уравнение производительности
  - $\text{Время выполнения} = \text{Cycles per instruction} * \text{Instruction count} * \text{Clock time}$
  - В отсутствие конвейера CPI = 5, на идеально работающем конвейере CPI = 1
- ❑ Структурные конфликты
  - Возникают из-за недостатков аппаратных ресурсов, когда доступное аппаратное обеспечение не в состоянии поддерживать все возможные комбинации параллельного исполнения инструкций
- ❑ Конфликты данных
  - Возникают когда инструкция зависит от результата выполнения предыдущей инструкции так, что это проявляется при перекрытии инструкций в конвейере
- ❑ Конфликты управления
  - Возникают при конвейеризации условных переходов и других инструкций, которые изменяют PC/IP (program counter/instruction pointer)

# Конвейер команд

## Конфликт конвейера – структурный...

- Структурные конфликты – возникают из-за недостатков аппаратных ресурсов, когда доступное аппаратное обеспечение не в состоянии поддерживать все возможные комбинации параллельного исполнения инструкций

Несколько инструкций на одной стадии конвейера

Инструкция\Время в тактах	1	2	3	4	5	6	7	8	9
Инструкция K (FP)	IF	ID	EX	EX	MEM	WB			
Инструкция K+1 (FP)		IF	ID	EX	EX	MEM	WB		
Инструкция K+2 (FP)			IF	ID	EX	MEM	WB		
Инструкция K+3 (INT)				IF	ID	EX	MEM	WB	
Инструкция K+4 (INT)					IF	ID	EX	MEM	WB

Одновременное обращение в память

Выполнение одной и той же неконвейеризируемой операции

- Обычное разрешение таких конфликтов – простой конвейера

# Конвейер команд

## Конфликт конвейера – структурный

- ❑ Стадии имеют различный смысл, но могут выполняться с использованием одних и тех же ресурсов

Инструкция\Время в тактах	1	2	3	4	5	6	7	8	9
Инструкция K (FP)	IF	ID	EX	EX	MEM	WB			
Инструкция K+1 (FP)		IF	ID	O	EX	EX	MEM	WB	
Инструкция K+2 (FP)			IF	O	ID	O	EX	MEM	WB
Инструкция K+3 (INT)				O	IF	O	ID	EX	MEM
Инструкция K+4 (INT)						O	IF	ID	EX

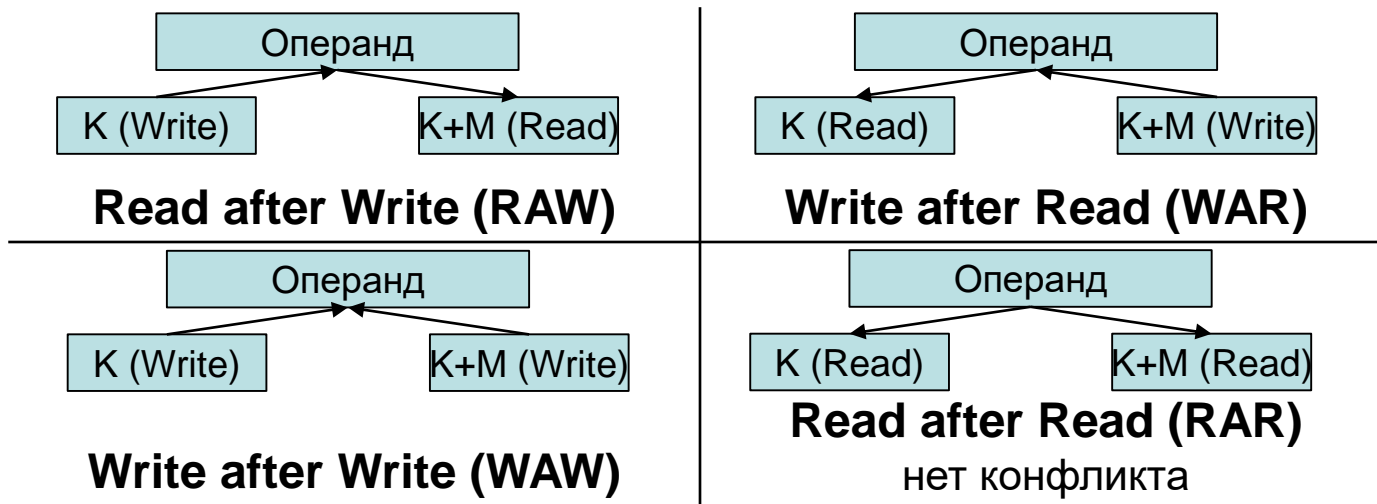
- ❑ Обычный способ разрешения таких конфликтов – простой конвейера (O)
- ❑ Для уменьшения структурных конфликтов можно наращивать количество аппаратных модулей, выполняющих одинаковые функции



# Конвейер команд

## Конфликт конвейера – конфликт данных

- Несколько инструкций могут использовать одни и те же данные на чтение и запись
- В таком случае различают 4 ситуации:



- Для решения конфликта данных необходимо организовывать простой конвейера для выполнения доступа к данным в нужном порядке

# Конвейер команд

## Конфликт конвейера – конфликт данных

□  $a=b+c$ ;

Инструкция\Время в тактах	1	2	3	4	5	6	7	8	9	10	11	12
LOAD [b], Rb	IF	ID	EX	MEM	WB							
LOAD [c], Rc		IF	ID	EX	MEM	WB						
ADD Rb, Rc, Ra			IF	O	O	ID	EX	MEM	WB			
STORE Ra, [a]				O	O	IF	O	O	ID	EX	MEM	WB

□ Для уменьшения потерь могут использоваться дополнительные механизмы, например, пересылка (forwarding)

# Конвейер команд

## Конфликт конвейера – конфликт управления

- Когда в коде выполняется условный переход, он может изменить номер следующей выполняемой инструкции

Инструкция\Время в тактах	1	2	3	4	5	6	7	8
Инструкция К (условный переход)	IF	ID	EX	MEM	WB			
Инструкция К+1 (выполняемая ветвь)		O	O	O	IF	ID	EX	MEM
Инструкция К+2 (выполняемая ветвь)						IF	ID	EX
Инструкция К+3 (выполняемая ветвь)							IF	ID
Инструкция К+4 (выполняемая ветвь)								IF

- В этом случае конвейер может считать следующую инструкцию только после определения направления перехода  
– в примере (MIPS) результат ветвления становится известным на стадии MEM

# Конвейер команд

## Параллелизм уровня инструкций...

---

- ❑ Параллелизм уровня инструкций (Instruction-Level Parallelism, ILP) – возможность одновременного выполнения нескольких команд
- ❑ Параллелизм уровня инструкций возможен, когда последовательные инструкции программы независимы и могут выполняться параллельно (с перекрытием)
- ❑ Конвейерная обработка команд использует ILP-потенциал кода
- ❑ ILP-потенциал кода зависит от структуры кода на языке высокого уровня и возможностей компилятора

# Конвейер команд

## Параллелизм уровня инструкций...

### □ Механизмы повышения ILP

- (1) Статическое планирование инструкций компилятором

Код программы на языке высокого уровня:

```
a=b+c;  
d=e+f;
```

Выполнение скомпилированного кода программы (если операции загрузки занимают один такт, то у исходного варианта будут простои из-за ожидания незагруженных операндов)

Исходный вариант кода (с простоями):

```
LOAD [b], Rb  
LOAD [c], Rc  
Простой  
ADD Rb, Rc, Ra  
STORE Ra, [a]  
LOAD [e], Re  
LOAD [f], Rf  
Простой  
ADD Re, Rf, Rd  
STORE Rd, [d]
```

Преобразованный (переупорядоченный) код:

```
LOAD [b], Rb  
LOAD [c], Rc  
LOAD [e], Re  
ADD Rb, Rc, Ra  
LOAD [f], Rf  
STORE Ra, [a]  
ADD Re, Rf, Rd  
STORE Rd, [d]
```

Почему нет простоя между последними инструкциями ADD и STORE?

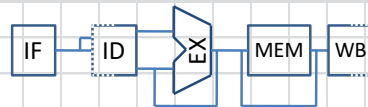
# Пример разбора выполнения программы на конвейере

1. Рассчитайте время выполнения программы и ее CPI

Количество стадий: 5. Пересылка: нет. Степень вычисления адреса перехода: EX (доступен после стадии MEM). Предсказание условного перехода: не производится.

```
int i = 2, Res = 1;
for( i = 2; i < 5; i ++ ){
    Res = Res + i;
}
```

1000	i=2
1004	5
1008	Res=1



CPI = 36/16 = 2.25

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42		
1	LD [1000], R1	IF	ID	EX	M	WB																																							
2	LD [1004], R2		IF	ID	EX	M	WB																																						
3	CMP R1, R2			IF	-	-	ID	EX	M	WB																																			
4	JGE [+32]				-	-	IF	ID	EX	M	WB																																		
5	LD [1000], R1						-	-	-	IF	ID	EX	M	WB																															
6	LD [1008], R3										IF	ID	EX	M	WB																														
7	ADD R1, R3, R1											IF	-	-	ID	EX	M	WB																											
8	ST R1, [1008]												-	-	IF	-	-	ID	EX	M	WB																								
9	LD [1000], R1															-	-	IF	ID	EX	M	WB																							
10	INC R1																		IF	-	-	ID	EX	M	WB																				
11	ST R1, [1000]																			-	-	IF	-	-	ID	EX	M	WB																	
12	JMP [0]																				-	-	IF	ID	EX	M	WB																		
13	LD [1000], R1																									IF	ID	EX	M	WB															
14	LD [1004], R2																										-	IF	ID	EX	M	WB													
15	CMP R1, R2																												-	IF	-	ID	EX	M	WB										
16	JGE [+32]																														-	IF	ID	EX	M	WB									
17	LD [1000], R1																															-	-	-	IF	ID									
18	LD [1008], R3																																											IF	

# Конвейер команд

## Параллелизм уровня инструкций

### □ Механизмы повышение ILP (1)

- (2) Увеличение размера **базового блока** (последовательность инструкций без ветвлений с единственными точками входа и выхода)
  - **развёртывание** циклов
- (3) Статическое предсказание переходов и спекулятивное выполнение команд
  - Кодируется в инструкциях перехода, требует поддержки ISA
    - при выполнении процессор загружает на конвейер инструкции из предсказанной ветви перехода, в случае неверного предсказания результат их исполнения не используется
  - Два основных метода для статического предсказания переходов
    - сбор информации о поведении программы при ее запусках и использование при перекомпиляции (профилирование)
    - эвристическое предсказание переходов на основе их направления или исходной инструкции языка высокого уровня; например, переходы назад помечаются как происходящие, а переходы вперед как не происходящие (предсказание перехода в цикле ошибается 1 раз при выходе)
  - Компилятор по возможности размещает наиболее вероятные ветви исполнения выше, что позволяет минимизировать сброс конвейера из-за условных переходов
    - при написании условных операторов рекомендуется размещать **наиболее вероятные** варианты в ветви if, а менее вероятные – в ветви else

# Конвейер команд

## Параллелизм уровня инструкций

### □ Механизмы повышение ILP (2)

- (4) Динамическое планирование выполнения инструкций CPU
- (5) Динамическое предсказание условных переходов CPU
  - Предположение о направлении перехода основывается на истории переходов
  - Пример: двухуровневый предсказатель с глобальной историей
    - Хранит результаты M последних выполненных инструкций условного перехода («история»)
    - Для каждой пары (история, инструкция) хранится статистика выполнения последних N переходов
    - Точность предсказания >90%
  - Процессор загружает на конвейер инструкции из предсказанной ветви перехода, в случае неверного предсказания результат их исполнения не используется
- (6) Одновременная многопоточность (SMT, Hyper-threading) – выполнение на одном физическом процессоре нескольких потоков; позволяет при простое ступеней конвейера использовать их для исполнения инструкций другого потока
  - Процессор дополняется средствами запоминания состояния потоков, схемами контроля одновременного выполнения нескольких потоков и т. д.
  - Одновременно выполняемые потоки конкурируют за исполнительные блоки единственного процессора, что приводит к возникновению структурных конфликтов



# СУПЕРСКАЛЯРНОСТЬ И ВЕКТОРИЗАЦИЯ

# Суперскалярность и векторизация

## Классификация Флинна

□ Классификация архитектур ЭВМ по признакам наличия параллелизма над инструкциями и данными

- SISD – работа с одинарными потоками инструкций и данных, как правило не является параллельной машиной (машина фон Неймана, конвейерные и суперскалярные процессоры)
- SIMD – применение одного потока инструкций ко множеству данных (векторные процессоры)
- MISD – применение множества инструкций к одним и тем же данным (представители отсутствуют)
- MIMD – одновременное использование множества инструкций и данных (многопоточные, многоядерные и многопроцессорные машины)

	Одиночный поток команд (single instruction)	Множество потоков команд (multiple instruction)
Одиночный поток данных (single data)	SISD	MISD
Множество потоков данных (multiple data)	SIMD	MIMD

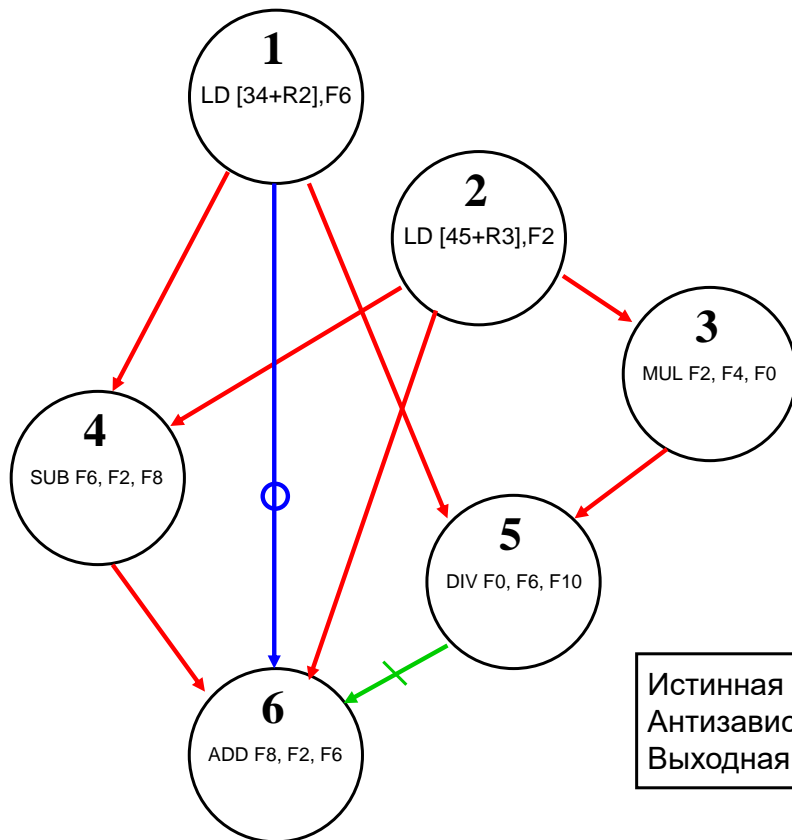
# Суперскалярность и векторизация

## Суперскалярность и внеочередное исполнение команд

- ❑ Наличие нескольких функциональных узлов (декодеры инструкций, исполнительные блоки и т.д.) позволяет обрабатывать несколько инструкций параллельно
- ❑ Суперскалярность – запуск на выполнение нескольких независимых инструкций за один такт для использования нескольких функциональных узлов на этапах конвейера
- ❑ Внеочередное исполнение (**Out-of-order execution**) – исполнение инструкций не в порядке их следования в программе, а в порядке готовности к исполнению
  - Позволяет избежать простоя процессора в тех случаях, когда данные, необходимые для выполнения очередной инструкции, недоступны
    - Требуется отслеживание зависимости между инструкциями (см. граф зависимостей)
  - Суперскалярность может быть реализована без поддержки внеочередного исполнения (Intel Pentium)

# Суперскалярность и векторизация

## Граф зависимостей



1 LD [34+R2], F6

2 LD [45+R3], F2

3 FMUL F2, F4, F0

4 FSUB F6, F2, F8

5 FDIV F0, F6, F10

6 FADD F8, F2, F6

Зависимости по данным:

(1, 4) (1, 5) (2, 3) (2, 4)




(2, 6) (3, 5) (4, 6)

Выходная зависимость:

(1, 6)

Антизависимость:

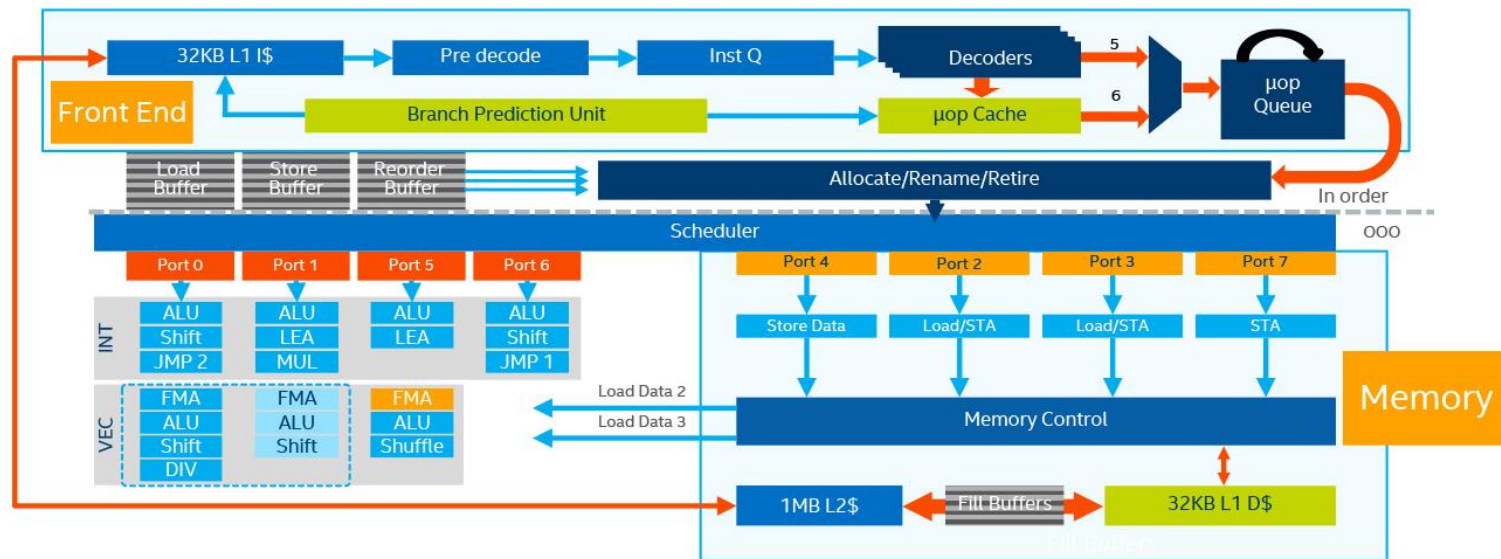
(5, 6)

Истинная зависимость данных (RAW)   
Антизависимость (WAR)   
Выходная зависимость (WAW) 

# Суперскалярность и векторизация

## Суперскалярность. Архитектура Skylake

- В каждом ядре современного процессора (Skylake):
  - 4 декодера инструкций + 1 сложный декодер инструкций
  - 22 исполнительных блока (Single: 4xALU (add, cmp, mov...), 1xDIV (div, sqrt, vsqrt, idiv ...), 2xShift, 1xSlow Int (mul, bit scanner ...), 2xBit Manipulation. Vector: 1xSIMD Misc, 1 FPxMov, 3xALU, 3xShift, 1xShuffle (shuf, pack, unpack...), 2xAdd, 2xMul)

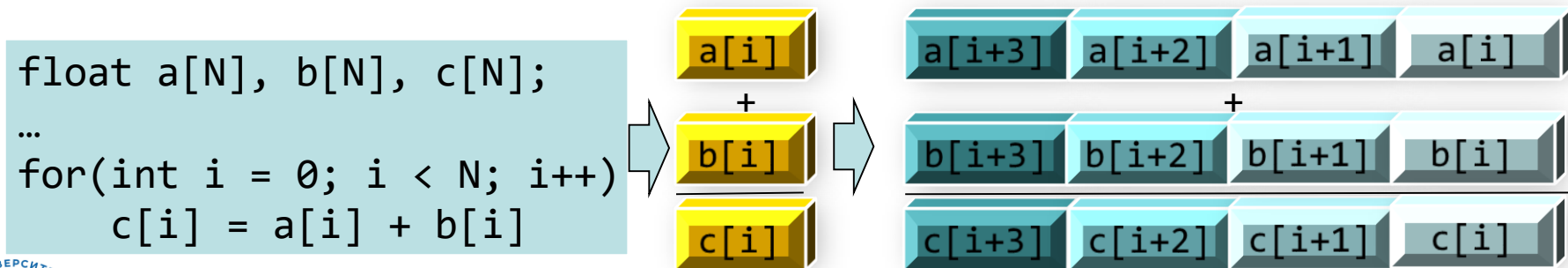


Источник: [https://en.wikichip.org/wiki/File:skylake\\_sp\\_buffer\\_windows.png](https://en.wikichip.org/wiki/File:skylake_sp_buffer_windows.png)

# Суперскалярность и векторизация

## Векторизация

- ❑ Векторизация – исполнение одной арифметической операции над множеством операндов **одновременно** (SIMD)
- ❑ Вектор – набор однотипных операндов, хранимых в векторном регистре. Длина векторного регистра определяется архитектурой:
  - SSE – 128 бит
  - AVX, AVX2 – 256 бит
  - AVX512 (Core, KNC, KNL) – 512 бит
- ❑ Векторизация может быть выполнена автоматически, с помощью директив компилятора, используя OpenMP > 4.0 или intrinsic



# Суперскалярность и векторизация

Intel® Intrinsic Guide (<https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html>)

## Intel® Intrinsic Guide

Updated  
12/06/2021

Version  
3.6.1

### Instruction Set

- ☐ MMX
- ☐ SSE
- ☐ SSE2
- ☐ SSE3
- ☐ SSE3.1
- ☐ SSE4.1
- ☐ SSE4.2
- ☐ AVX
- ☐ AVX2
- ☐ FMA
- ☐ AVX\_VNNI
- ☐ AVX-512
- ☐ KNC
- ☐ AMX
- ☐ SVML
- ☐ Other

### Categories

- ☐ Application-Targeted
- ☐ Arithmetic
- ☐ Bit Manipulation
- ☐ Control Flow
- ☐ Data Movement
- ☐ Floating Point
- ☐ Hashing
- ☐ Memory Access
- ☐ Memory Management
- ☐ SIMD
- ☐ Streaming SIMD Extensions
- ☐ Vector

The Intel® Intrinsic Guide contains reference information for Intel intrinsics, which provide access to Intel instructions such as Intel® Streaming SIMD Extensions (Intel® SSE), Intel® Advanced Vector Extensions (Intel® AVX), and Intel® Advanced Vector Extensions 2 (Intel® AVX2).

- For information about how Intel compilers handle intrinsics, view the [Intel® C++ Compiler Classic Developer Guide and Reference](#).
- For questions about Intel intrinsics, visit the [Intel® C++ Compiler board](#).

\_mm\_search

```
void _mm_2intersect_epi32 (__m128i a, __m128i b, __mmask8* k1, __mmask8* k2) vp2intersectd
void _mm256_2intersect_epi32 (__m256i a, __m256i b, __mmask8* k1, __mmask8* k2) vp2intersectd
void _mm512_2intersect_epi32 (__m512i a, __m512i b, __mmask16* k1, __mmask16* k2) vp2intersectd
void _mm_2intersect_epi64 (__m128i a, __m128i b, __mmask8* k1, __mmask8* k2) vp2intersectq
void _mm256_2intersect_epi64 (__m256i a, __m256i b, __mmask8* k1, __mmask8* k2) vp2intersectq
void _mm512_2intersect_epi64 (__m512i a, __m512i b, __mmask8* k1, __mmask8* k2) vp2intersectq
__m512i _mm512_4dpwssd_epi32 (__m512i src, __m512i a0, __m512i a1, __m512i a2, __m512i a3, __m128i * b) vp4dpwssd
__m512i _mm512_mask_4dpwssd_epi32 (__m512i src, __mmask16 k, __m512i a0, __m512i a1, __m512i a2, __m512i a3, __m128i * b) vp4dpwssd
__m512i _mm512_maskz_4dpwssd_epi32 (__mmask16 k, __m512i src, __m512i a0, __m512i a1, __m512i a2, __m512i a3, __m128i * b) vp4dpwssd
__m512i _mm512_4dpwssds_epi32 (__m512i src, __m512i a0, __m512i a1, __m512i a2, __m512i a3, __m128i * b) vp4dpwssds
__m512i _mm512_mask_4dpwssds_epi32 (__m512i src, __mmask16 k, __m512i a0, __m512i a1, __m512i a2, __m512i a3, __m128i * b) vp4dpwssds
__m512i _mm512_maskz_4dpwssds_epi32 (__mmask16 k, __m512i src, __m512i a0, __m512i a1, __m512i a2, __m512i a3, __m128i * b) vp4dpwssds
__m512 _mm512_4fmadd_ps (__m512 src, __m512 a0, __m512 a1, __m512 a2, __m512 a3, __m128 * b) vf4fmaddps
__m512 _mm512_mask_4fmadd_ps (__m512 src, __mmask16 k, __m512 a0, __m512 a1, __m512 a2, __m512 a3, __m128 * b) vf4fmaddps
```

# ИЕРАРХИЯ ПАМЯТИ



# Иерархия памяти

Кэш-память:

- L1 (на ядро):  
~32 КБ данных,  
~32 КБ инструкций,  
Доступ 2-3 такта  
ПС 8-12 ТБ/сек

- L2 (на ядро): 128-1024 КБ,  
Доступ 12-14 тактов  
ПС 4-6 ТБ/сек

- L3 (общий): 4-80 МБ,  
Доступ 50-70 тактов  
ПС 800-1000 ГБ/сек

Регистры процессора (L0):  
32-1024 байта Прямой доступ

Быстрый доступ

ЭВМ

CPU

Регистры

Кэш-память

Оперативная память

Долговременная память  
(Жесткие диски, твердотельные накопители)

Сторонние устройства  
(Оптические диски, магнитные ленты)

Дальше от CPU:

- Дешевле
- Больше задержка
- Меньше пропускная способность (ПС)
- Больше объем

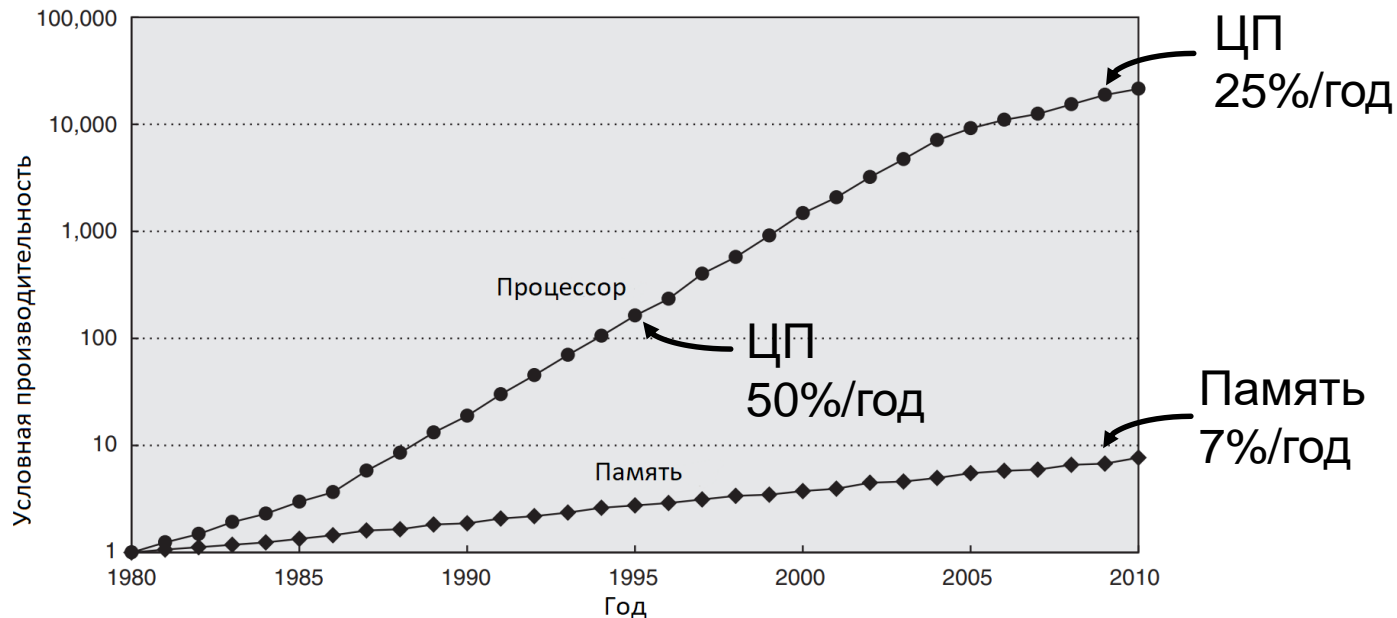
Оперативная память (RAM): 4-3096 ГБ (DDR4),  
Доступ 100-500 тактов, ПС до 500 ГБ/сек

Долговременная память: 1-64 ТБ,  
ПС 100-1000 МБ/сек

# Иерархия памяти

## Развитие памяти...

- ❑ Развитие памяти происходит медленнее, чем развитие вычислительной мощности ЭВМ
- ❑ Чаще всего именно использование памяти является узким местом в программах

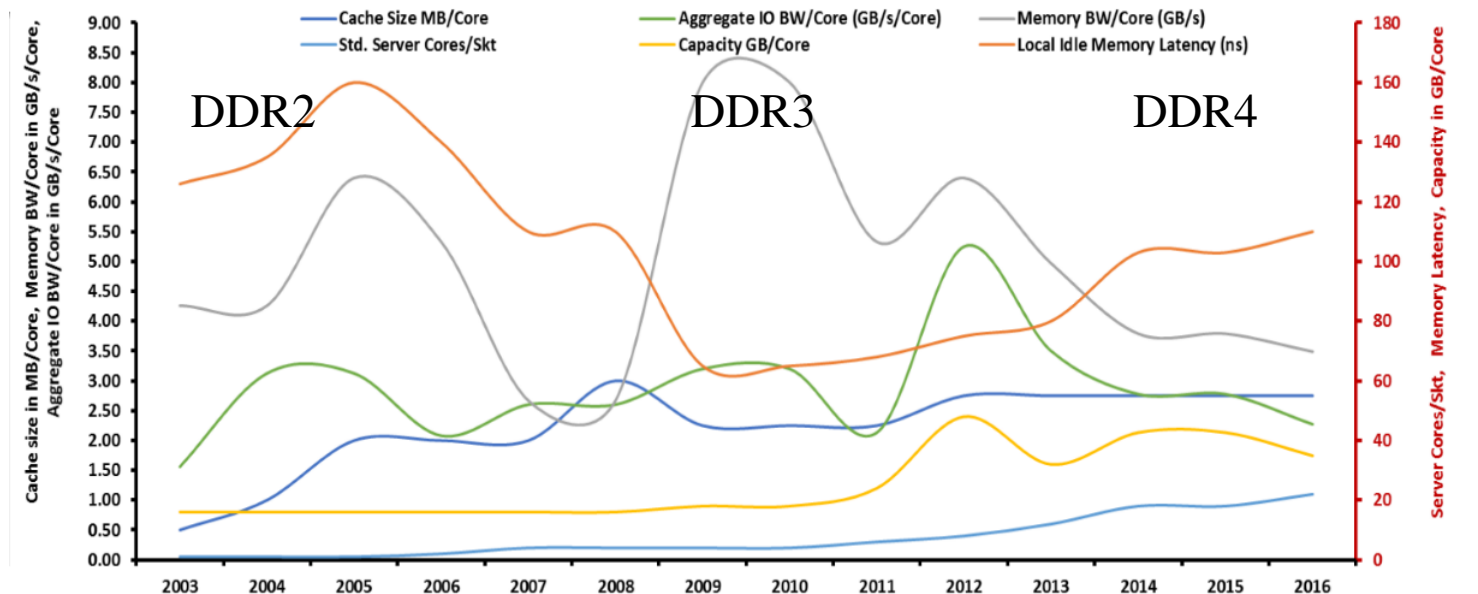


Источник: Hennessy J. L., Patterson D. A. Computer architecture: a quantitative approach. – Elsevier, 2011.

# Иерархия памяти

## Развитие памяти

- В 2010 году контроллер (северный мост), отвечающий за работу CPU с памятью, ускорителями и южным мостом, заменяется на аналог, находящийся на подложке процессора



Источник: <https://www.delltechnologies.com/en-us/blog/memory-centric-architecture-vision/>

# Иерархия памяти

## Работа кэш-памяти

- ❑ **Кэш-память** – промежуточный буфер памяти с быстрым доступом, содержащий информацию, которая может быть запрошена с **наибольшей вероятностью**
- ❑ Как размещать блоки памяти в кэше?
  - Политика размещения блоков и организация кэша: полностью ассоциативный, наборно-ассоциативный, кэш с прямым отображением
- ❑ Как проверять наличие памяти в кэше (попадание/промах)?
  - Идентификация блока – сравнение тегов
- ❑ Какой блок заменять в случае промаха?
  - Алгоритмы замещения: случайный, FIFO, LRU, pseudo-LRU, ...
- ❑ Что происходит при записи?
  - Политика записи кэша: кэш со сквозной записью (write-through, моментально сохранять все изменения), кэш с обратной записью (write-back, запись в основную память может быть отложена)

# Иерархия памяти

## Архитектуры кэш-памяти

- ❑ Кэш-память по политике включения делится на инклюзивную и эксклюзивную
- ❑ **Инклюзивная** архитектура кэш-памяти подразумевает дублирование данных в кэше всех уровней
  - Небольшие накладные расходы если кэши разных уровней сильно отличаются по объему
- ❑ **Эксклюзивная** архитектура кэш-памяти предполагает уникальность хранящейся в кэш-памяти информации
  - Более сложная схема реализации
  - Более эффективное использование кэш-памяти
- ❑ По организации кэш-память делится на **кэш прямого отображения** и **наборно-ассоциативный кэш**

# Иерархия памяти

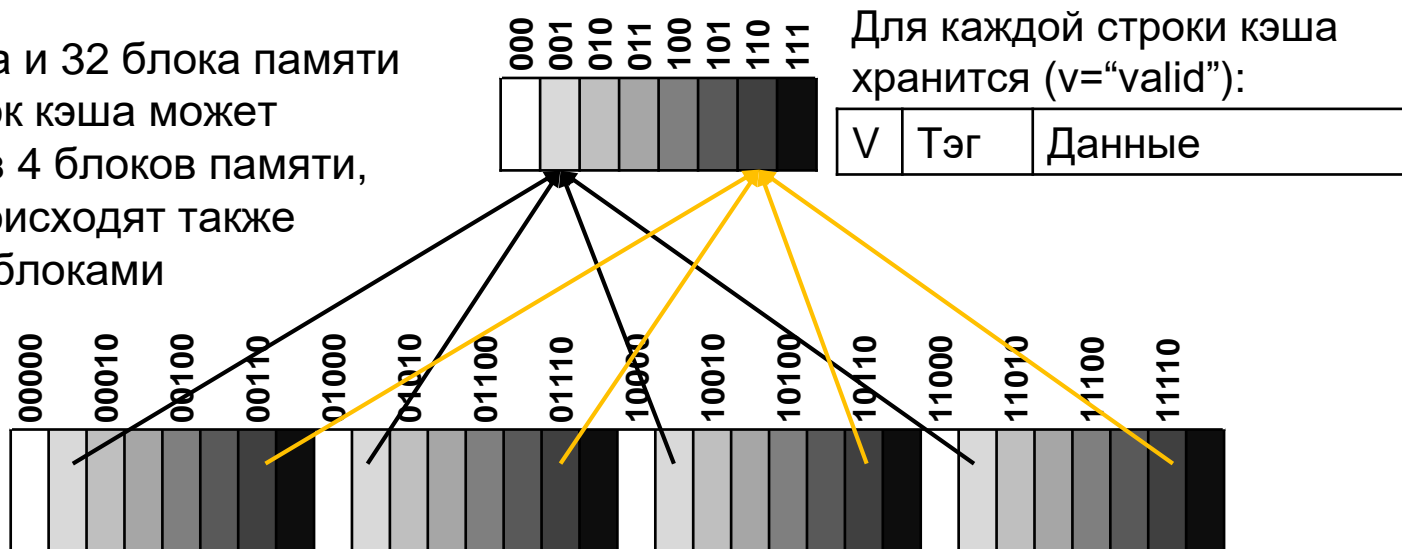
## Кэш прямого отображения

- ❑ **Кэш прямого отображения** – адрес в памяти, по которому происходит обращение, однозначно определяет строку кэша, в которой могут находиться кэшируемые данные

$$(\text{Адрес в кэше}) = (\text{Адрес памяти}) \% (\text{Число блоков в кэше})$$

- ❑ **Пример:**

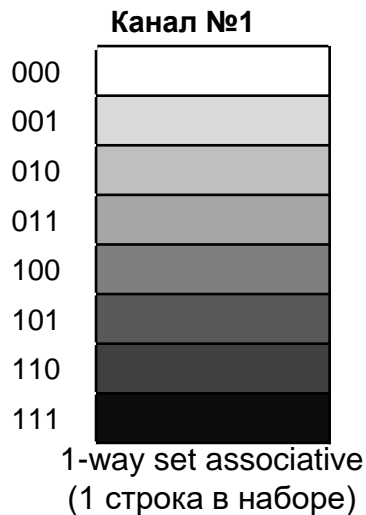
- 8 блоков кэша и 32 блока памяти
- В каждый блок кэша может попасть один из 4 блоков памяти, вытеснения происходят также между этими 4 блоками



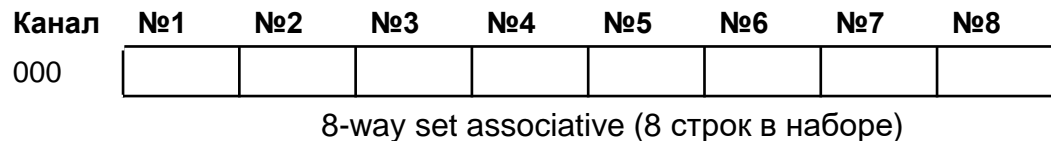
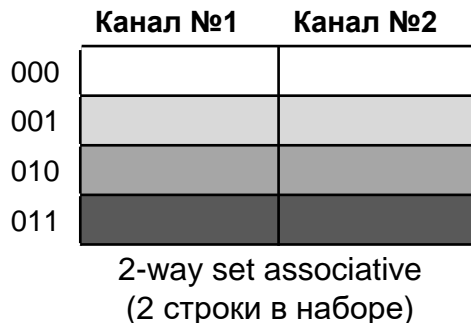
# Иерархия памяти

## Наборно-ассоциативный кэш и степень ассоциативности

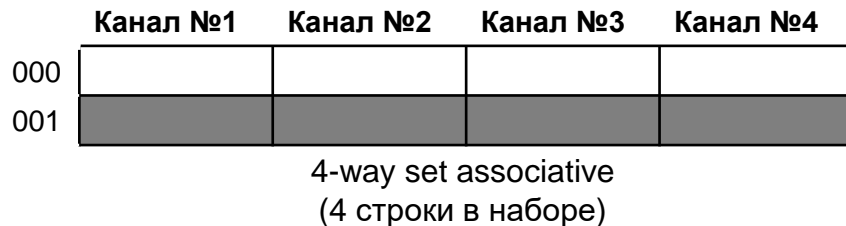
□ Рассмотрим подробнее кэш с 8 строками:



Кэш прямого отображения



Полностью ассоциативный кэш (ячейка памяти претендует на любую кэш-линию)



□ В наборно-ассоциативном кэше каждая ячейка памяти претендует на некоторое количество кэш-линий (блоков кэша). Позволяет уменьшить число промахов в результате уменьшения числа конфликтов между блоками памяти, которые были бы спроецированы в одну и ту же строку в кэше прямого отображения

# Иерархия памяти

## Intel Cascade Lake caches

---

- ❑ L1I Cache – 32 KiB/core
  - 8-way set associative, 64 sets, 64 B line size
- ❑ L1D Cache – 32 KiB/core
  - 8-way set associative, 64 sets, 64 B line size
  - Latency 4 cycles for simple pointer accesses, 5 cycles for complex addresses
- ❑ L2 Cache – 1 MiB/core
  - 16-way set associative, 64 B line size
  - Inclusive
  - Latency 14 cycles
- ❑ L3 Cache – 1.375 MiB/core
  - 11-way set associative, 2 048 sets, 64 B line size
  - Non-inclusive victim cache
  - Latency 50-70 cycles



# Иерархия памяти

## Виды промахов

- ❑ Промах при чтении инструкции – дает наибольшую задержку, так как текущий поток исполнения не может продолжить работу до загрузки инструкции в кэш
- ❑ Промах при чтении данных – дает среднюю задержку, так как текущий поток исполнения может продолжить выполнять независимые инструкции, а после загрузки данных возобновить выполнение инструкции, ожидавшей данные
- ❑ Промах при записи данных – дает наименьшую задержку, так как запись ставится в очередь, и продолжается выполнение независимых инструкций

# Иерархия памяти

## Категории промахов

- ❑ Compulsory misses – промахи, вызванные первым обращением к запрошенному адресу. Размеры и ассоциативность кэшей не влияют на количество данных промахов. Можно уменьшить предвыборкой или увеличением размера кэш-линий.
  - Существуют программная предвыборка и аппаратная предвыборка
- ❑ Capacity misses – промахи, вызванные исключительно конечным размером кэша, происходящие вне зависимости от степени ассоциативности или размера кэш-линии. Можно уменьшить, используя более локальную по памяти программу.
- ❑ Conflict misses – промахи, вызванные конфликтом. Разделяются на промахи отображения и замещения. Можно уменьшить использованием других алгоритмов замещения.

# Иерархия памяти

## Возможности по оптимизации

### □ Программная оптимизации и возможности компилятора

#### – Инструкции

- Расположение процедур в памяти так, чтобы сократить промахи из-за конфликтов
- Профилирование для обнаружения конфликтов

#### – Данные

- Слияние массивов – увеличение пространственной локальности
- Согласование вложенности циклов для обработки в соответствии с порядком хранения данных в памяти
- Слияние циклов – составление из нескольких независимых циклов, обрабатывающих одни данные, одного
- Поблочная обработка массивов – улучшение временной локальности за счет многократной обработки блоков данных

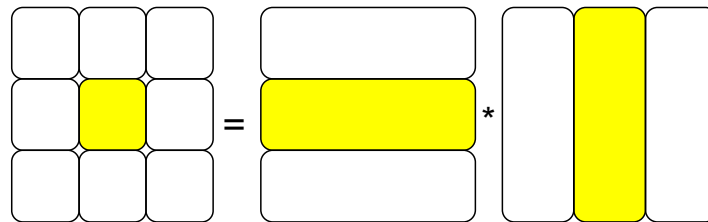
# Иерархия памяти

## Возможности по оптимизации – умножение матриц

```
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1){
    r = 0.0;
    for (k = 0; k < N; k = k+1)
      r = r + A[i][k]*B[k][j];
    C[i][j] = r;
  };
```

2 вложенных цикла читают все  $N$  элементов одной строки  $A$  многократно. Промехи из-за ограниченной емкости зависят от  $N$  и размера кэша:  $2N^3 + N^2$

```
for (jj = 0; jj < N; jj = jj+B)
  for (kk = 0; kk < N; kk = kk+B)
    for (i = 0; i < N; i = i+1)
      for (j = jj; j < min(jj+B-1,N); j=j+1){
        r = 0;
        for (k = kk; k < min(kk+B-1,N); k = k+1)
          r = r + A[i][k]*B[k][j];
        C[i][j] = C[i][j] + r;
      }
```

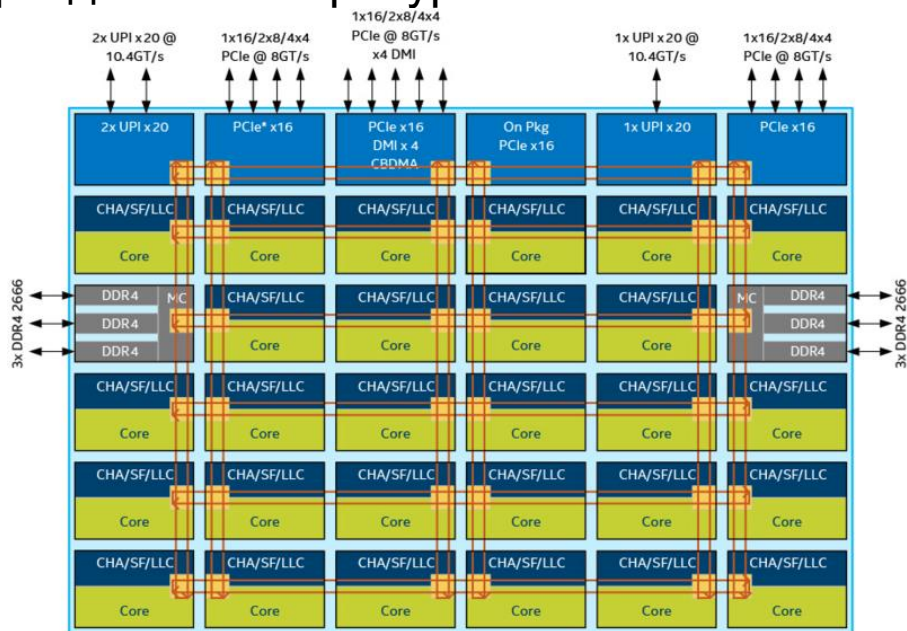


$B$  – фактор блочности.  
Промехи из-за ограниченной емкости:  $2N^3/B + N^2$   
Что будет с промахами из-за конфликтов?

# NUMA АРХИТЕКТУРА

# Многоядерные системы

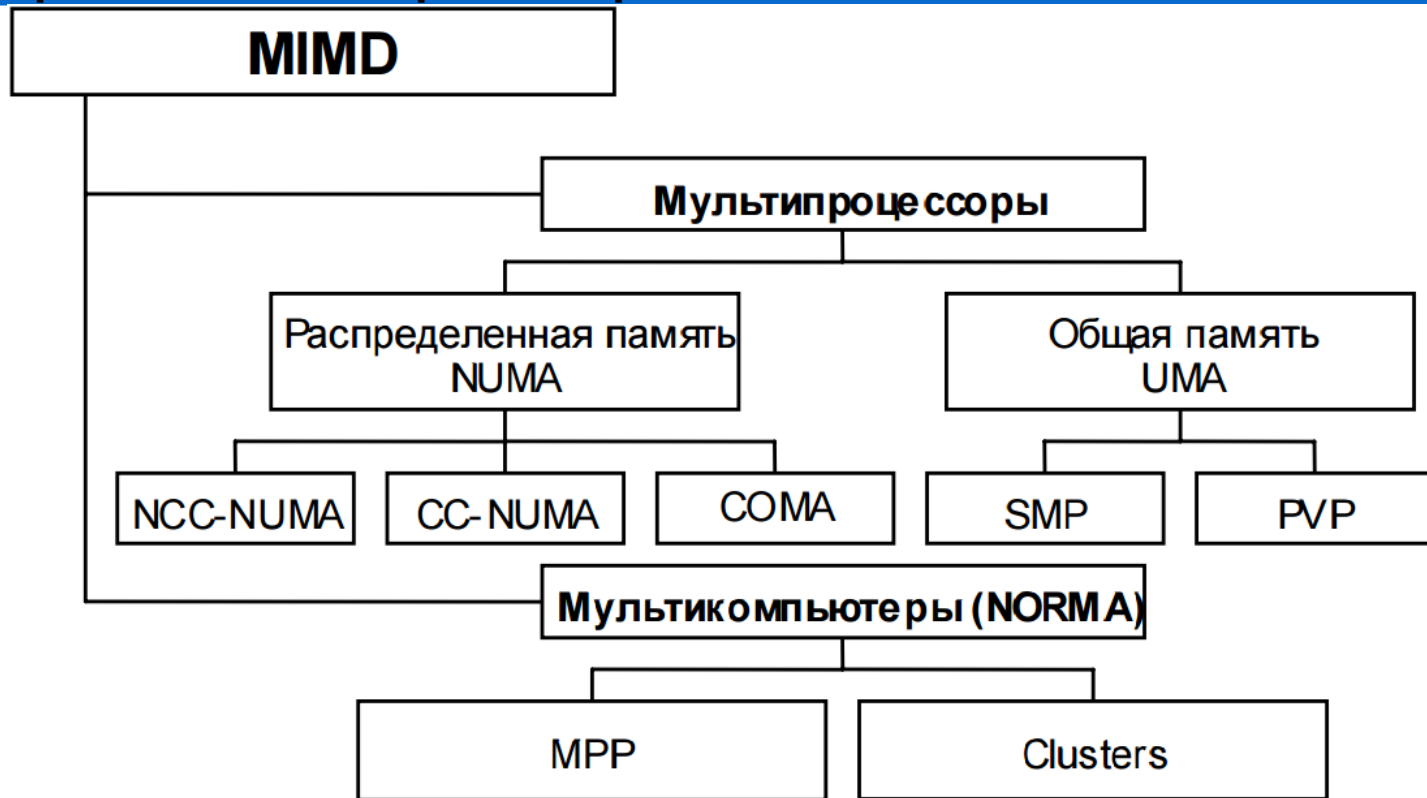
- ❑ Многоядерные процессоры – центральные процессоры содержащие 2+ ядер
- ❑ Как правило, ядра процессора содержат общие участки кэшей (L3 – общий, L2 – разделяемый), некоторые разделяемые ресурсы и имеют связь с другими ядрами
- ❑ Архитектура Skylake
  - Доступ к памяти других ядер неравномерный (Sub-NUMA Clustering – SNC) и зависит от расстояния между ядрами
  - Доступ к разделяемым ресурсам неравномерный



Источник: [https://en.wikichip.org/wiki/File:skylake\\_sp\\_xcc\\_die\\_config.png](https://en.wikichip.org/wiki/File:skylake_sp_xcc_die_config.png)

# Многопроцессорные системы

## Классификация многопроцессорных систем



Источник: <https://www.intuit.ru/studies/courses/1156/190/lecture/4942?page=4>

# Многопроцессорные системы

## Многопроцессорность...

□ Многопроцессорность разделяют на 2 больших типа по строению памяти (1)

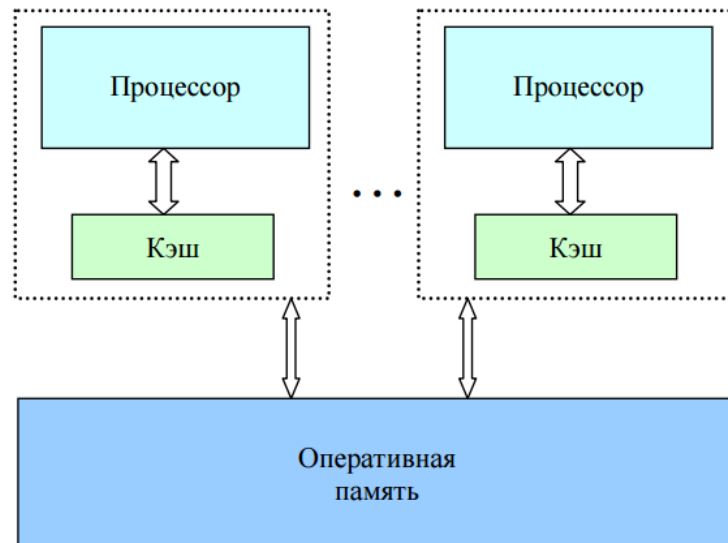
– Общая память (UMA) – однородный доступ ко всей памяти:

- Симметричные мультипроцессоры (SMP) – все процессоры имеют одинаковый доступ к общей разделяемой памяти.

Основными минусами является определение состояния кэш-памяти, что влечет постоянные синхронизации данных.

Как следствие, такие системы обладают плохой масштабируемостью.

- Векторные параллельные процессоры (PVP) – являются SMP системой, где процессоры являются векторно-конвейерными





# Многопроцессорные системы

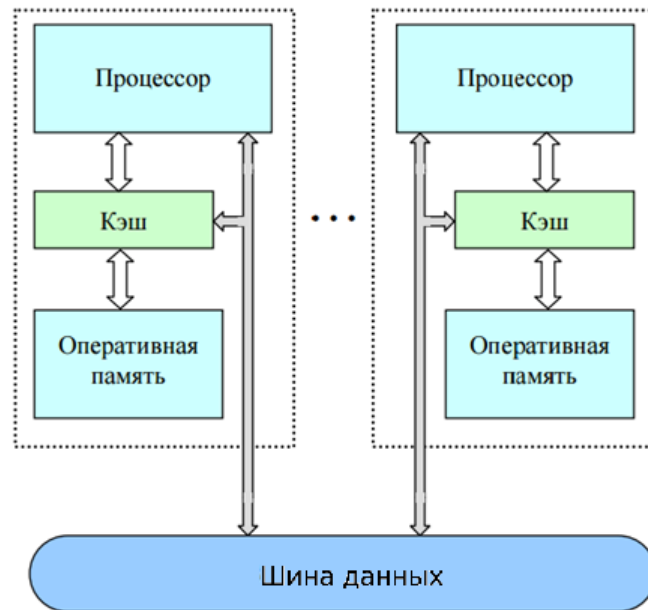
## Многопроцессорность...

- ❑ Насколько все плохо? Предположим, мы выполняем скалярное умножение вещественных векторов двойной точности на системе с длиной вектора 512
- ❑ CPU 16 ядер, 2.3 ГГц:  
 $P_{\max} = 2.3 * 10^9 * 16 * 8 * 2 = 294,4 \text{ GFLOPS}$   
– Частота \* ЧислоЯдер \* ШиринаВектора \* FMA
- ❑ Пропускная способность памяти:  
 $B = 50 \text{ ГБ/с}$
- ❑ Арифметическая интенсивность:  
 $AI = 2 \text{ FLOP/16 Б} = 0.125 \text{ FLOP/Б}$   
 $AI * B = 6,25 \text{ GFLOPS}$
- ❑ Фактическая производительность  
 $P = \min(P_{\max}, I * B) = 6,25 \text{ GFLOPS}$

# Многопроцессорные системы

## Многопроцессорность

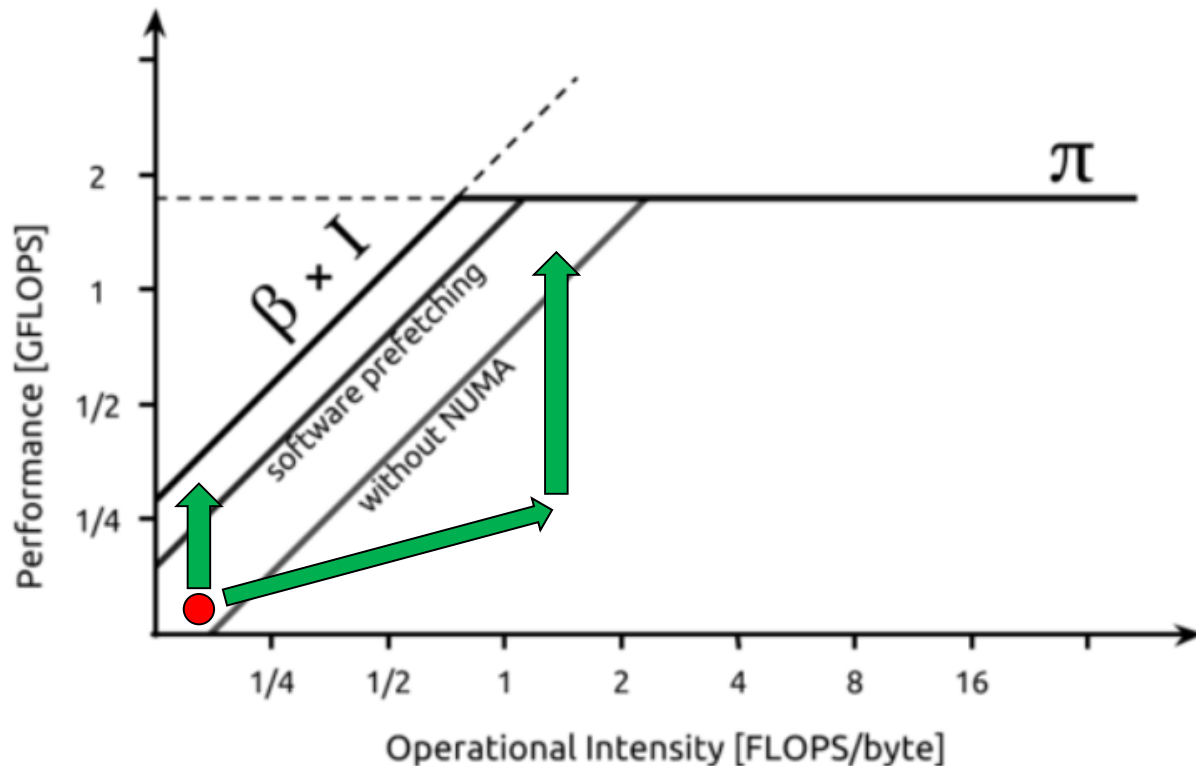
- ❑ Многопроцессорность разделяют на 2 больших типа по строению памяти (2)
  - Распределенная память (NUMA) – неоднородный доступ к памяти:
    - COMA – система с доступом только к кэш-памяти. На сегодняшний момент крайне мало представителей
    - CC-NUMA – система, обеспечивающая когерентность локальных кэшей разных процессоров
    - NCC-NUMA – система, обеспечивающая доступ к локальной памяти без аппаратной поддержки когерентности кэшей разных процессоров
  - NUMA архитектура обладает хорошей масштабируемостью, но при этом доступ к разным участкам памяти занимает разное время
  - Большинство современных компьютеров имеют NUMA архитектуру памяти (ccNUMA)



# Многопроцессорные системы

## Roofline Model – ограничение производительностью памяти

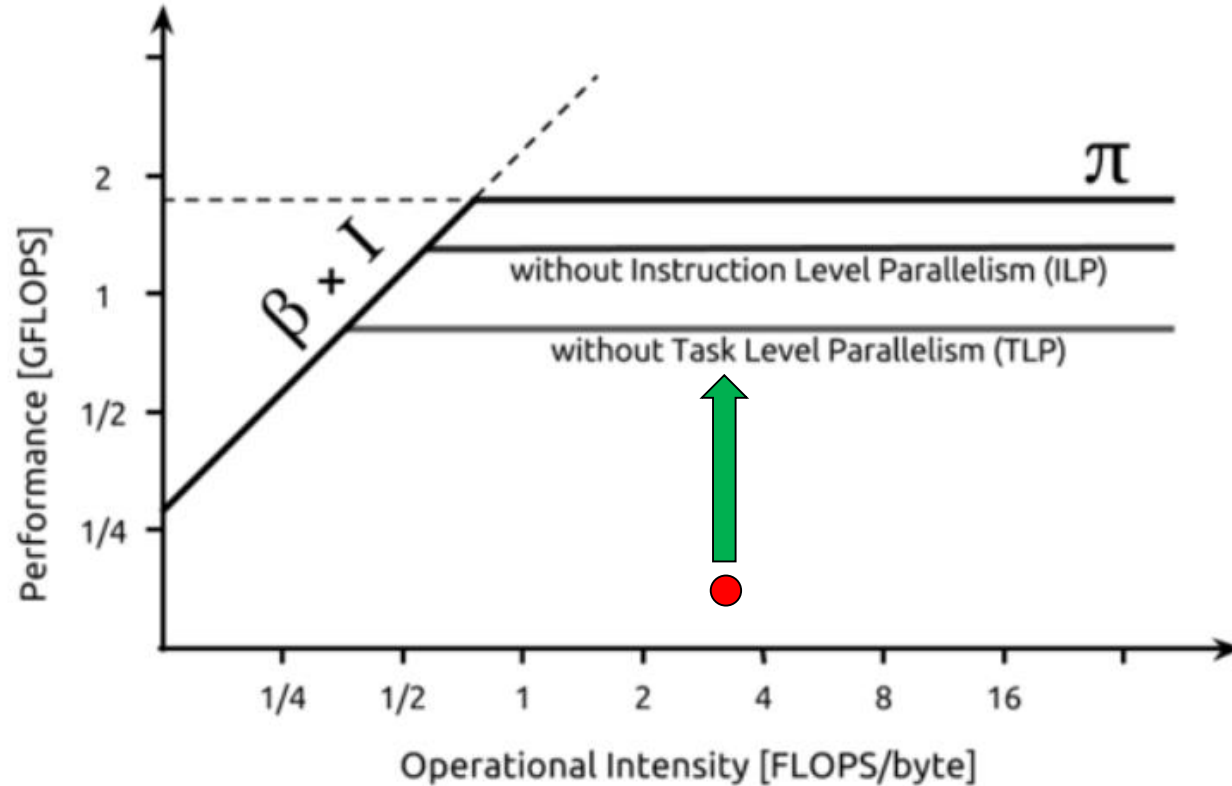
- Roofline Model – инструмент анализа производительности программ
  - позволяет определить арифметическую интенсивность и факторы, наиболее ограничивающие скорость работы



[https://en.wikipedia.org/wiki/Roofline\\_model](https://en.wikipedia.org/wiki/Roofline_model)

# Многопроцессорные системы

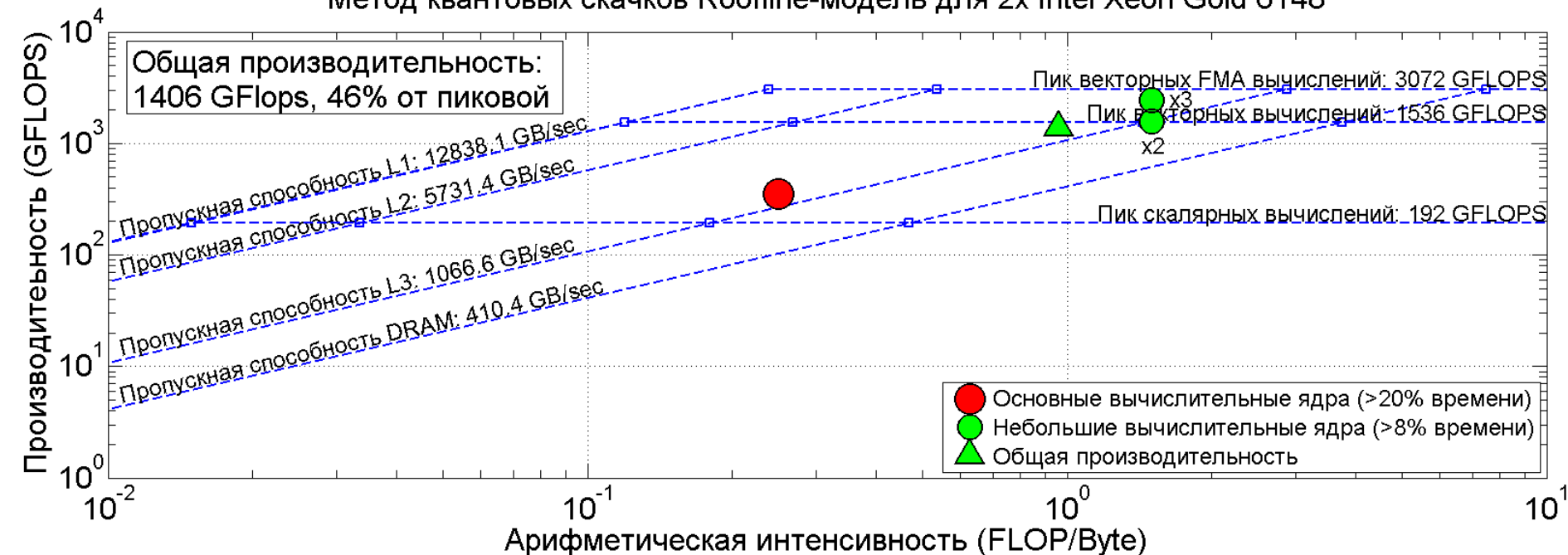
## Roofline Model – ограничение производительностью CPU



# Многопроцессорные системы

## Roofline Model – пример реальной программы

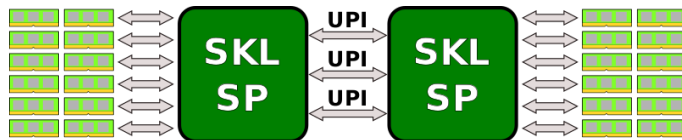
Метод квантовых скачков Roofline-модель для 2x Intel Xeon Gold 6148



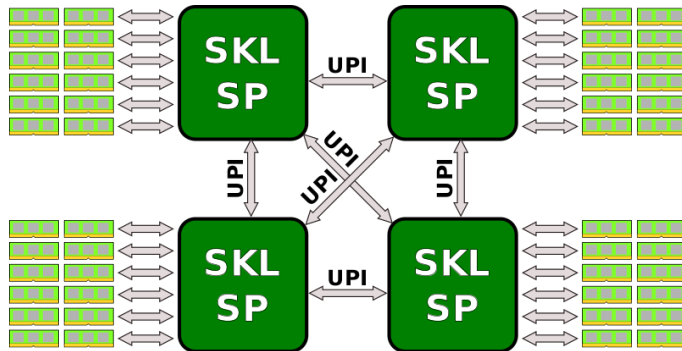
# Многопроцессорные системы

## Многопроцессорность в современных системах (CC-NUMA)

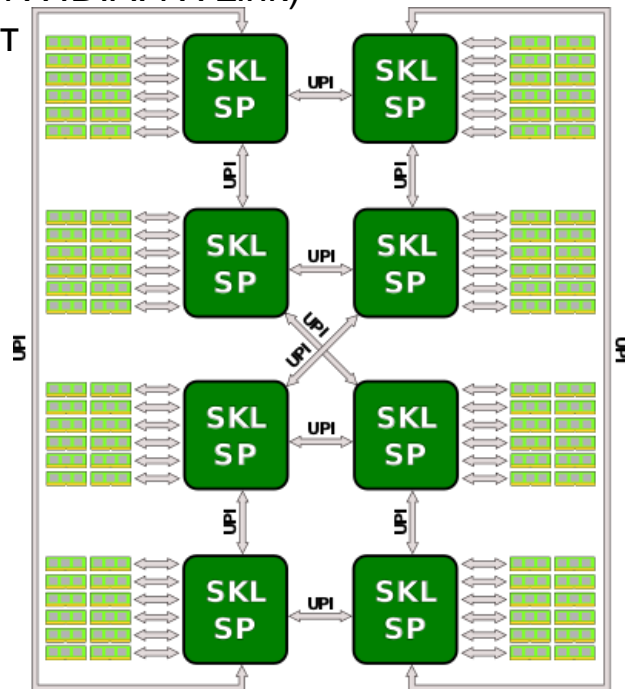
- ❑ В современных архитектурах процессоры соединены специальными шинами – интерконнектор (Intel: QPI, UPI; AMD: HyperTransport; NVIDIA: NVLink)
- ❑ Как правило, скорость интерконнектора превосходит скорость одного канала RAM



2 сокета с 3 каналами UPI



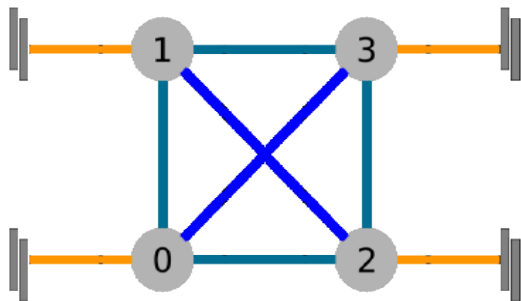
4 сокета с 3 каналами UPI



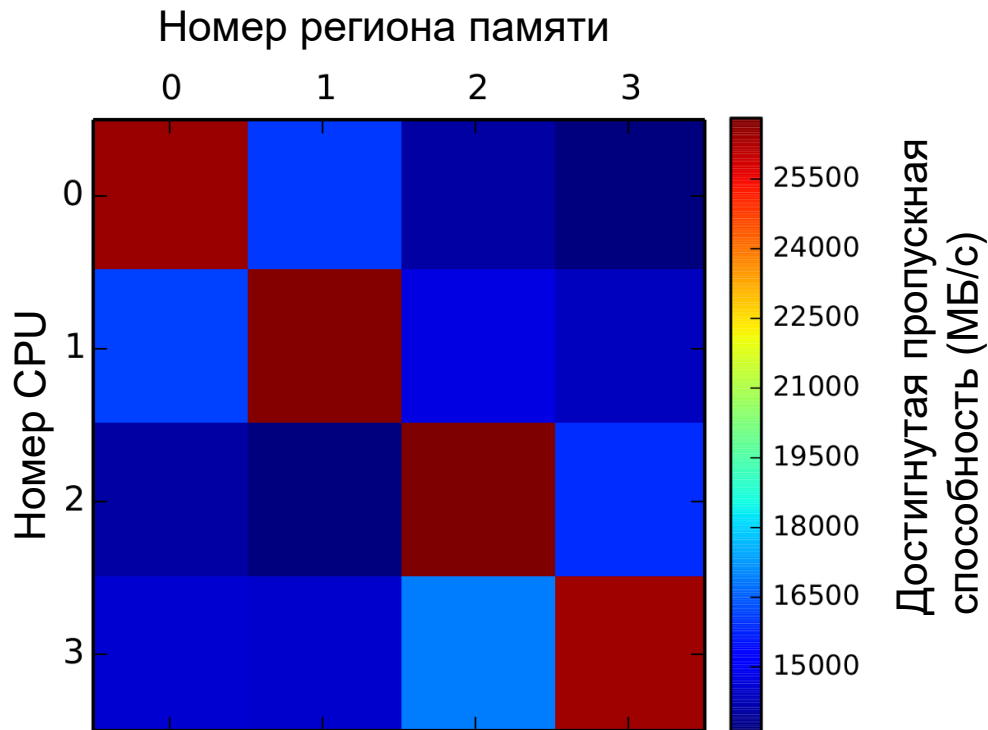
8 сокетов с 3 каналами UPI

# Многопроцессорные системы

## ccNUMA – пропускная способность между доменами

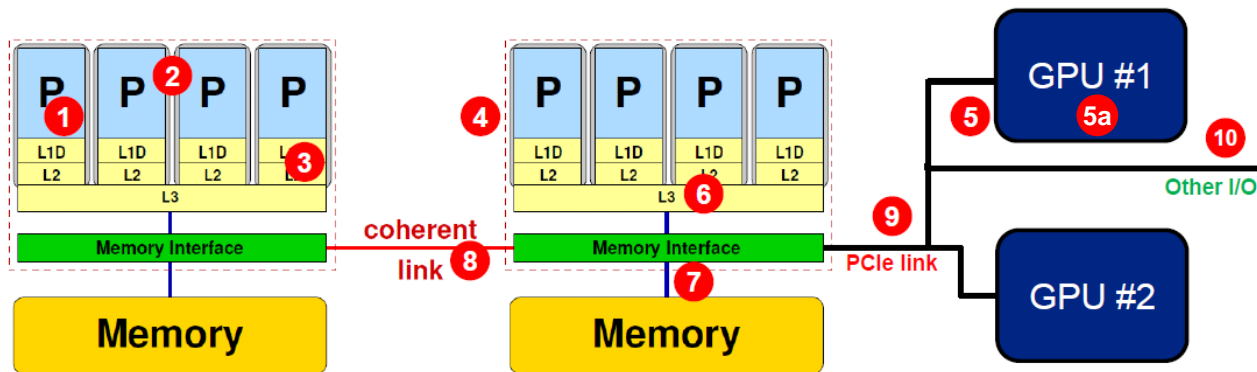


- Нужно обеспечивать локальность данных
  - libnuma
  - Параметры VirtualAlloc
  - Другие способы (порядок инициализации)



# Многопроцессорные системы

## Варианты/уровни параллелизма



Параллельные ресурсы

1. SIMD (векторы)
2. Ядра CPU
3. Локальные кеши
4. Процессоры/домены с NUMA
5. Ускорители
- 5a. Параллелизм внутри ускорителей

Разделяемые ресурсы

6. Общий кеш CPU/домена
7. Шина памяти CPU/домена
8. Интерконнектор
9. Шины PCIe
10. Ресурсы ввода-вывода



# Многопроцессорные системы

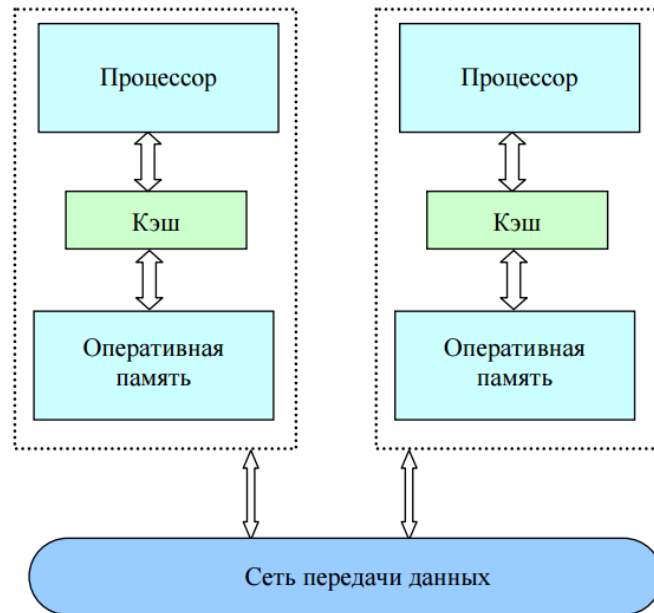
## О чем нужно помнить

- ❑ Не выполняется ли лишняя работа
- ❑ Равномерность загрузки (FLOPS, число инструкций, ПС памяти)
- ❑ Насыщение использования локальной памяти (кешей)
- ❑ FLOPS и арифметическая интенсивность
- ❑ Векторизация
- ❑ Характеристика CPI (Cycles Per Instruction)
- ❑ Распределение типов инструкций, доля условных переходов и успешность их предсказания

# Многопроцессорные системы

## Системы с распределенной памятью

- ❑ Мультикомпьютеры – не могут обеспечить общий доступ ко всей памяти. Компьютер использует свою локальную память, а для доступа к другой осуществляется явная передача данных между компьютерами
- ❑ Выделяют два типа построения:
  - Массивно-параллельная система (МРР) – система из однородных узлов, соединенных быстрой магистралью для передачи сообщений
  - Кластер – система из компьютеров (узлов) общего назначения, соединённых в единую сеть с использованием стандартных сетевых технологий на базе коммутаторов



# ЗАКЛЮЧЕНИЕ

# Заключение

---

В данной лекции рассмотрены:

- ❑ Основные архитектурные механизмы, влияющие на производительность программ
- ❑ Конвейерная обработка инструкций и параллелизм уровня инструкций (ILP)
- ❑ Механизмы векторизации (параллелизм по данным)
- ❑ Иерархия памяти и неоднородность доступа к ней в современных архитектурах
- ❑ Классификация и структура многопроцессорных систем

# Литература

---

1. Паттерсон Д., Хеннеси Д. Архитектура компьютера и проектирование компьютерных систем. – 2012.
2. Хеннеси Д. Л., Паттерсон Д. А. Компьютерная архитектура. Количественный подход. Издание 5-е //М.: ТЕХНОСФЕРА. – 2016.
3. Хэррис Д. М., Хэррис С. Л. Цифровая схемотехника и архитектура компьютера. – 2015.
4. Таненбаум Э. Архитектура компьютера //Питер. – Питер, 2015.
5. Гергель В. П. Теория и практика параллельных вычислений. – 2007.
6. Carvalho C. The gap between processor and memory speeds //Proc. of IEEE International Conference on Control and Automation. – 2002.

# Авторский коллектив

---

- ❑ Мееров Иосиф Борисович, к.т.н., доцент, зам. зав. каф. МОСТ
- ❑ Сысоев Александр Владимирович, к.т.н., доцент каф. МОСТ
- ❑ Линев Алексей Владимирович, зав. лаб. интернета вещей, каф. ПРИН
- ❑ Волокитин Валентин Дмитриевич, программист лаборатории СТиВВ, каф. МОСТ
- ❑ Козинев Евгений Александрович, к.т.н., преподаватель каф. МОСТ
- ❑ Панова Елена Анатольевна, инженер лаборатории СТиВВ, каф. МОСТ

# Контакты

---

Нижегородский государственный университет

<http://www.unn.ru>

Центр компетенций oneAPI в ННГУ

<http://hpc-education.unn.ru/ru/центр-компетенций-oneapi-в-ннгу>

Институт информационных технологий, математики и механики

<http://www.itmm.unn.ru>