МО ВВС ИВМиМГ СО РАН

Программирование многоядерных архитектур

(слайды для лекции 2013/04/20)

Киреев С.Е., Маркова В.П., Остапкевич М.Б., Перепелкин В. А.

2013

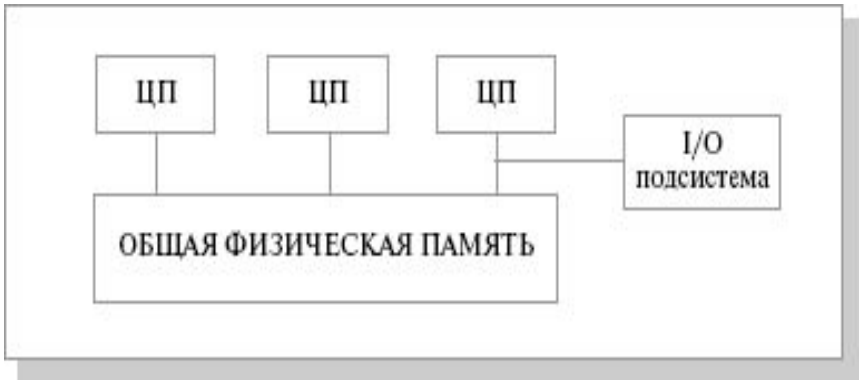
План презентации

- кратко о параллельных ЭВМ;
- введение в многопоточность;
- интерфейсы работы с потоками;
- OpenMP:
  - организация многопоточности,
  - синхронизация,
  - что замедляет вычисления;
- цель работы, задание

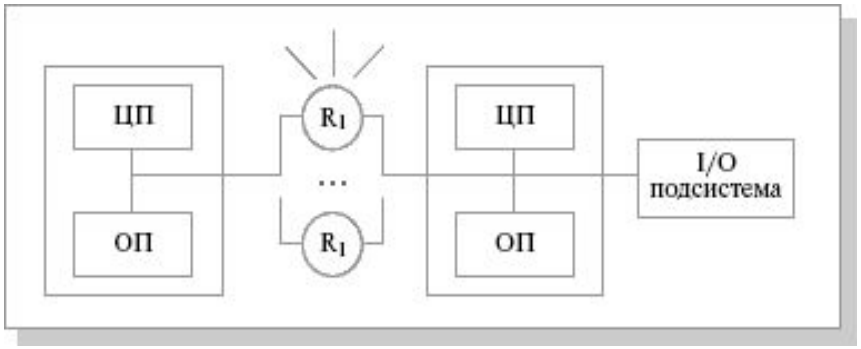
Кратко о параллельных ЭВМ

Основные классы параллельных ЭВМ

SMP — *Sequent (1991), HP 9000 V-class*
Tilera (2010)



MPP — *IBM RS/6000 SP2*
CRAY T3E



NUMA — *SGI Origin2000*

PVP — *CRAY-1, CRAY X1, Fujitsu VPP*

COW — *Beowulf*

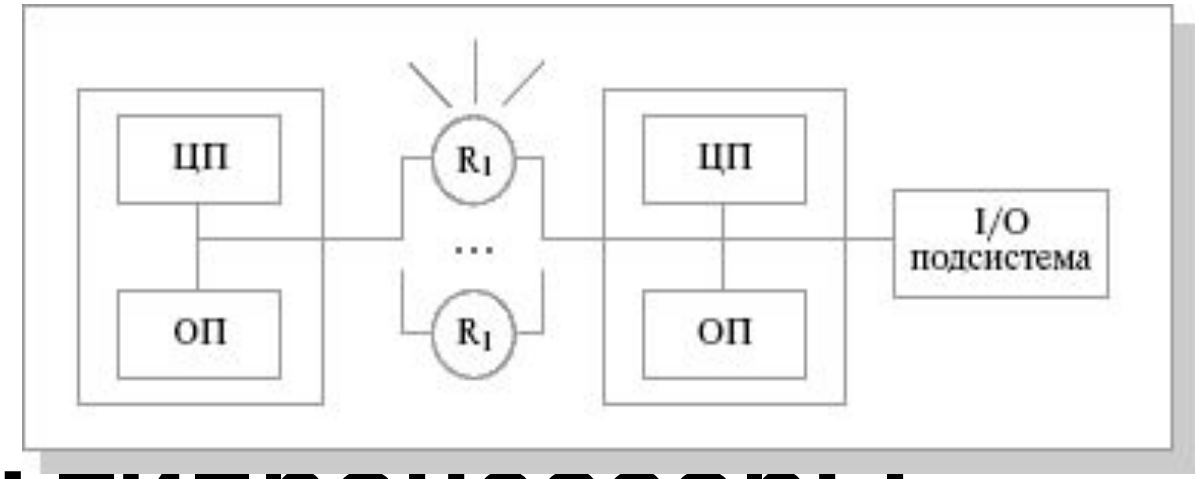
NOW

GPGPU — *NVIDIA, ATI, S3*

FPGA — *Xilinx, Altera*

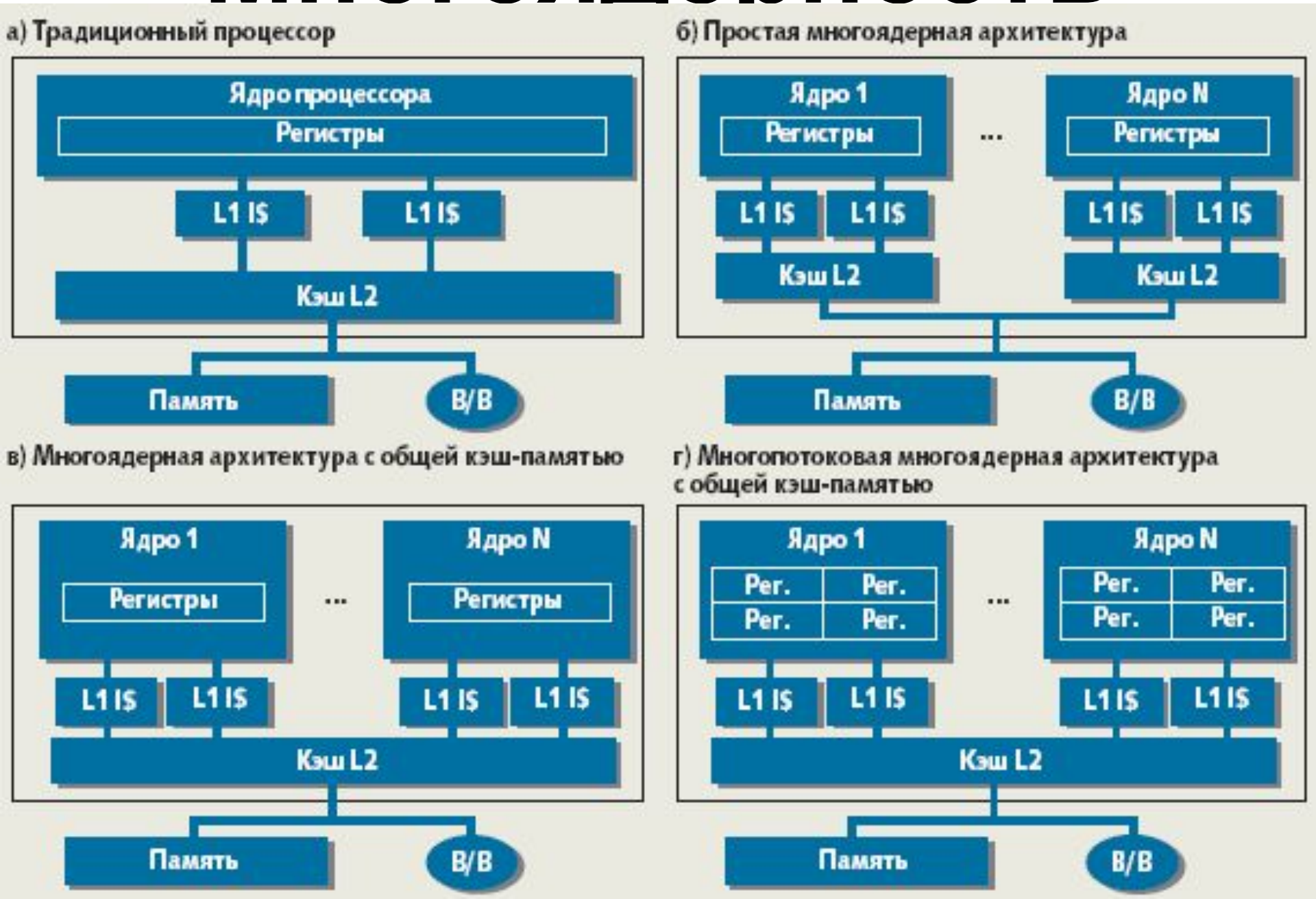
Основные виды параллельных ЭВМ

- **Мультикомпьютеры**



- **Мультипроцессоры**

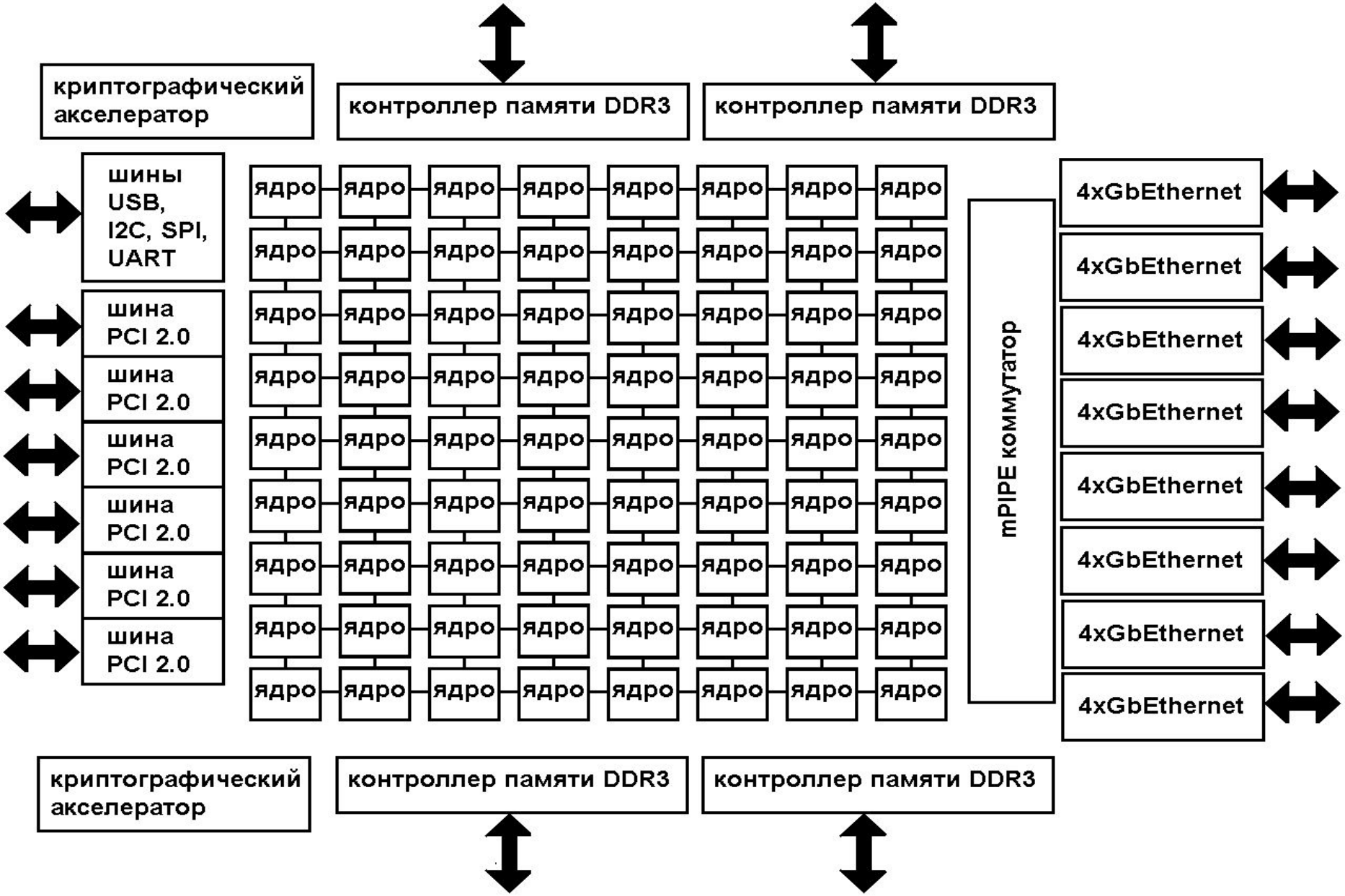
Многоядерность



Примеры многоядерных процессоров

- Intel XEON Phi 60 ядер
- Tilera TILE-Gx8072 72 ядра
- GreenArrays GA144A12 144 ядра

Процессор Tilera TILE-Gx8072



Введение в многопоточность

- процесс,
- поток,
- многозадачность,
- многопоточная программа.

Процесс

Процесс - экземпляр запущенной на исполнение программы.

Имеет собственные:

- **состояние, код возврата, переменные окружения, идентификаторы себя, владельца и родителя, приоритет;**
- **отдельное адресное пространство/виртуальная память и/или физическая память;**
- **данные в памяти; исполняемый код в памяти;**
- **один или несколько потоков исполнения**

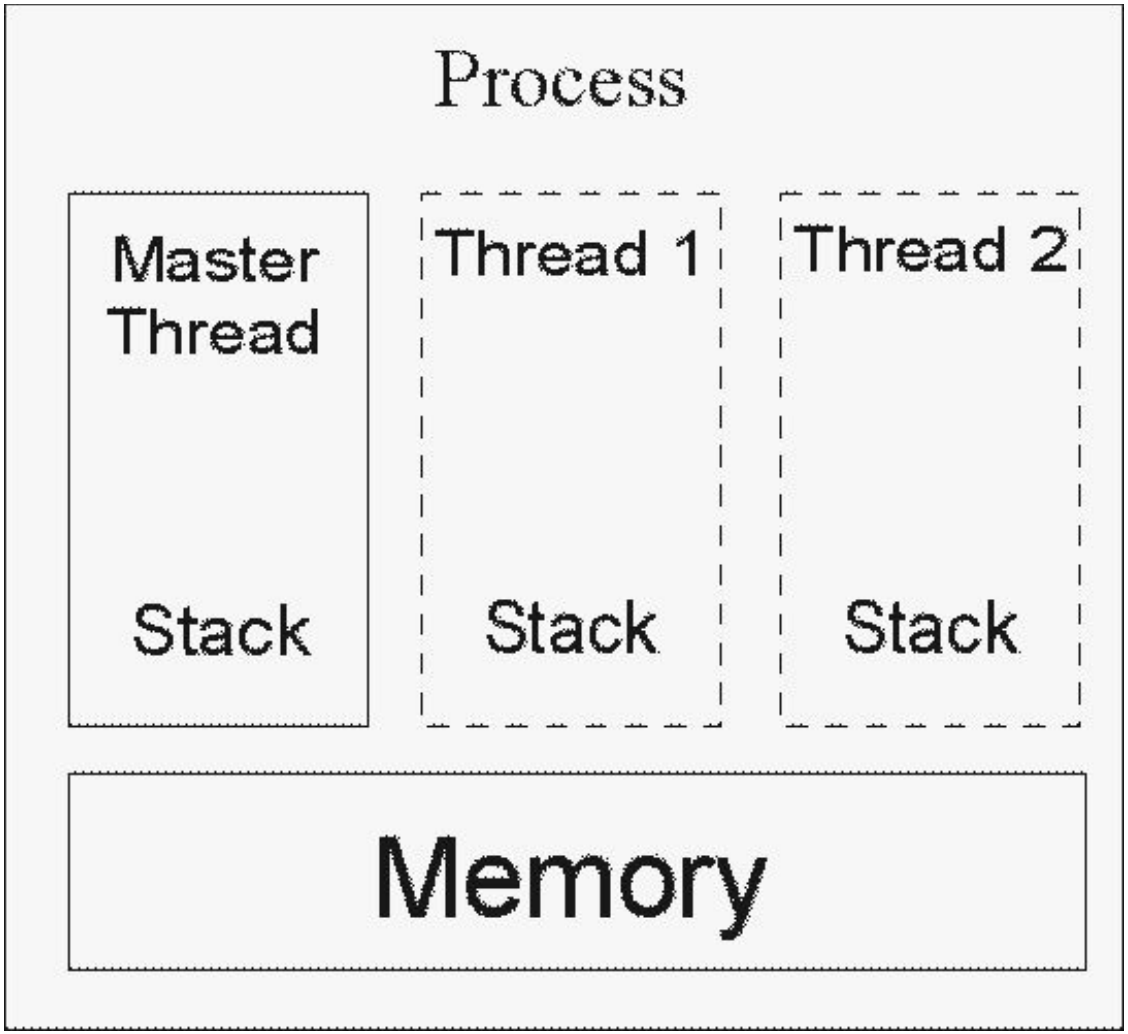
Поток

Разделяет общие:

- **адресное пространство/ память,**
- **данные в памяти,**
- **исполняемый код.**

Имеет собственные:

- **ID, приоритет;**
- **<u>Регистры</u>:**
- **поток команд,**



Многозадачность

Формы многозадачности:

- **разделение времени;**
- **параллельное исполнение.**

Виды планирования многозадачности:

- **вытесняющая (Win32, Win64, Linux);**
- **кооперативная (Win16).**

Интерфейсы по работе с потоками

Интерфейс	Windows	Windows+ MinGW, Cygwin	UNIX
Win32 Threads	+	+	
POSIX Threads		+	+
OpenMP	+	+	+

OpenMP

- организация многопоточности;
- синхронизация;
- что замедляет вычисления.

Что такое OpenMP?

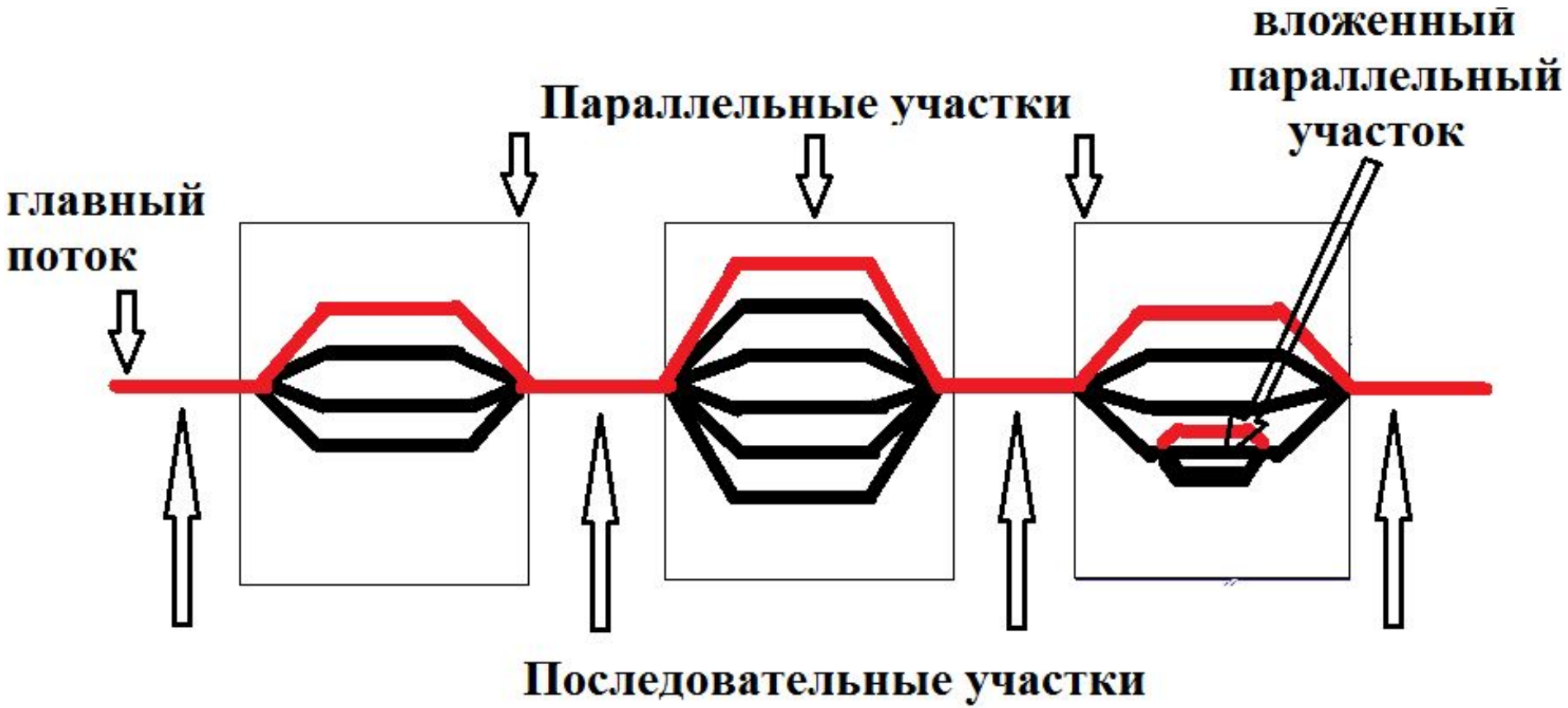
Стандарт OpenMP определяет:

- **интерфейс (API);**
- **набор прагм компилятора;**
- **переменные окружения**

для построения многопоточных приложений на мультипроцессорных системах в модели общей памяти.

Архитектура OpenMP

Модель исполнения Fork-join



OpenMP программа 1

Напишем многопоточную программу, в которой:

- каждый поток печатает свой номер,
- один поток печатает количество запущенных потоков в

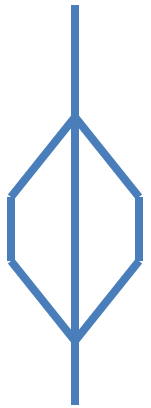
Шаги создания OpenMP программы

1. **Подключение заголовочного файла:**

```
#include <omp.h>
```

2. **Создание параллельно исполняемого блока в функции main():**

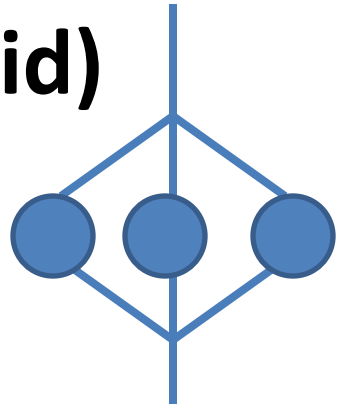
```
void main(){
    #pragma omp parallel
    {
    }
}
```



Шаги создания OpenMP программы

3. Декларация переменной, собственную копию которой имеет каждый поток:

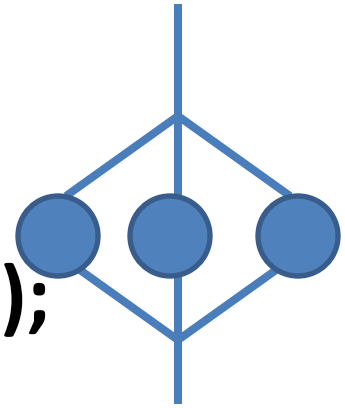
```
#include <omp.h>
void main(){
    int tid;
    #pragma omp parallel private(tid)
    {
    }
}
```



Шаги создания OpenMP программы

3. Получение и печать номера потока:

```
void main(){
    int tid;
    #pragma omp parallel private(tid)
    {
      tid = omp_get_thread_num();
      printf("thread num=%d\n", tid);
    }
}
```

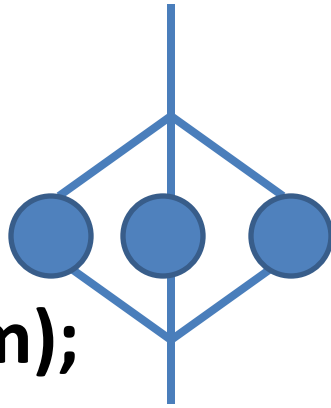


Шаги создания OpenMP программы

4. Декларация переменной, общей для всех потоков.

5. Получение и печать количества потоков:

```
void main(){
    int tid, tnum;
    #pragma omp parallel private(tid)
    { tid = omp_get_thread_num();
     printf("thread num=%d\n", tid);
      tnum= omp_get_num_threads();
      printf("number of threads=%d\n", tnum);
    }
}
```



Шаги создания OpenMP программы

6. Исполнение кода только одним потоком:

```
#pragma omp parallel private(tid)
{
    tid = omp_get_thread_num();
    printf("thread num=%d\n", tid);
    if(tid == 0){
        tnum= omp_get_num_threads();
        printf("number of threads=%d\n", tnum);
    }
}
```

Шаги создания OpenMP программы

7. Компиляция OpenMP программы:

В Windows + Microsoft Visual C++:

```
cl.exe myprog.c /openmp
```

В UNIX, Linux, Windows+MinGW, Windows+CygWin

```
gcc –o myprog myprog.c –fopenmp
```

OpenMP программа 2

Напишем многопоточную программу, для поэлементного суммирования двух векторов способами:

1) используя конструкцию распределения работы,
2) используя секции.

OpenMP программа 2

1. Пишем общую часть

```
#include <omp.h>
void main ()
{  int i;
   float a[N], b[N], c[N];
   for (i=0; i < N; i++)
    a[i] = b[i] = 1.0;
}
```

Способ 1. Распределение работы между потоками

2. Добавляем параллельную часть

```
#pragma omp parallel shared(a,b,c) private(i)
{
  #pragma omp for schedule(dynamic, 100) nowait
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];
}
```

Ограничения в for

- число итераций известно до входа в цикл,
- итерации независимы
- Индекс меняется только следующим образом:

  i++, i--, i += c, i -= c.

Способ 2. С использованием секций

2. Добавляем параллельную часть

```
#pragma omp parallel shared(a,b,c) private(i)
{
#pragma omp sections nowait
 {
 #pragma omp section
  for (i=0; i < N/2; i++)c[i] = a[i] + b[i];
 #pragma omp section
  for (i=N/2; i < N; i++) c[i] = a[i] + b[i];
  }
}
```

OpenMP программа 3

Напишем многопоточную программу вычисления скалярного произведения векторов.

OpenMP программа 3

1. Напишем последовательную часть.

```
#include <omp.h>
#define N     1000


void main()
{  int i;
   float a[N], b[N], res;
   for (i=0; i < N; i++){a[i] = 2.0l; b[i] = 1.0l;}
       res = 0.0l;
}
```

OpenMP программа 3

1. Напишем параллельную часть.

```
#pragma omp parallel for \
    default(shared) private(i) \
    schedule(static, 100) reduction(+:res)
 for(i = 0; i < N; i++)
   res = res + (a[i] * b[i]);
```

OpenMP программа 4

Напишем многопоточную программу, для увеличения счетчика на единицу:

1) наивно;
2) с синхронизацией.

Вариант 1. Наивный

```
#include <omp.h>
void main(){
   int x;  x = 0;
   #pragma omp parallel for shared(x)
   for(int i = 0; i < 1000000; i++){
    x = x + 1;
   }
   printf(“x=%d\n”, x);
}
```

Вариант 1. Наивный

```
$ ./plus1
  x=977177
$ ./plus1
  x=975883
$ ./plus1
  x=980608
$ ./plus1
  x=979246
```

Получился неверный результат.

Вариант 1. Наивный

Поток 1	Поток 2
Получить X.	
	Получить X.
	Добавить 1.
	Записать в X.
Добавить 1.	
Записать в X.	

Вариант 2. С синхронизацией

```
#include <omp.h>

void main()
{   int x;
    x = 0;
    #pragma omp parallel for shared(x)
    for(int I = 0; I < 1000000; i++){
     #pragma omp critical
     x = x + 1;
    }
    printf(“x=%d”, x)
}
```

Вариант 2. С синхронизацией

```
$ ./plus2
x=1000000
```

Вариант 2. С синхронизацией

Поток 1	Поток 2
Получить X.	
Добавить 1.	
Записать в X.	
	Получить X.
	Добавить 1.
	Записать в X.

Что замедляет вычисления?

- избыточное число порождений потока -> если можно, выносим параллельный блок из цикла;
- одновременный доступ двух потоков к одному участку памяти;
- излишняя синхронизация.

Цель работы

- **Написание многопоточного приложения с использованием интерфейса OpenMP.**

Задание

В соответствии с вариантом задания реализовать алгоритм с использованием интерфейса OpenMP.

Варианты:

1) **скалярное произведение двух векторов (20 баллов),**
2) **умножение матрицы на вектор (22 балла),**
3) **Умножение матрицы на матрицу (25 баллов),**
4) **Решение системы линейных алгебраических уравнений методом Гаусса (30 баллов).**