Параллельное и многопоточное программирование

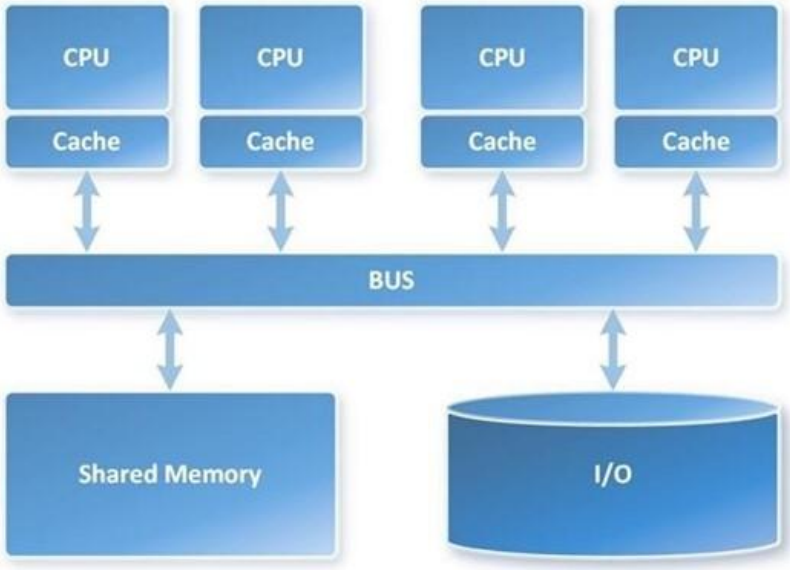
лекция 6

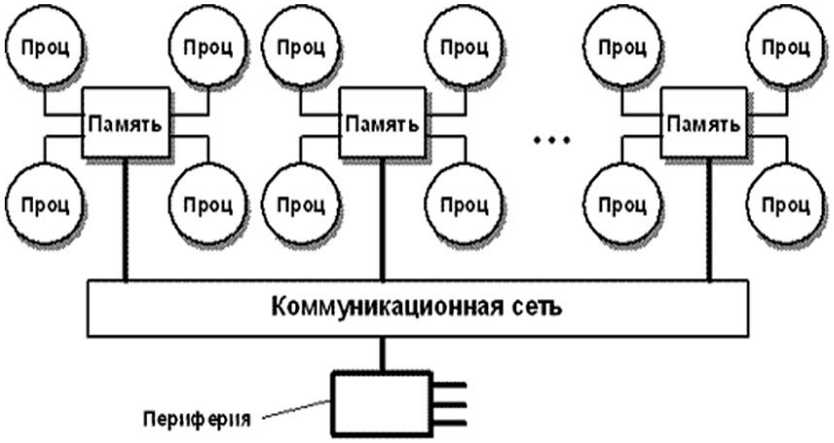
OpenMP

- **OpenMP** (Open Multi-Processing) — открытый стандарт для распараллеливания программ на языках Си, Си++ и Фортран. Дает описание совокупности директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью.

Обзор технологии OpenMP

- Интерфейс OpenMP задуман как стандарт параллельного программирования для многопроцессорных систем с общей памятью (SMP, ccNUMA, …)





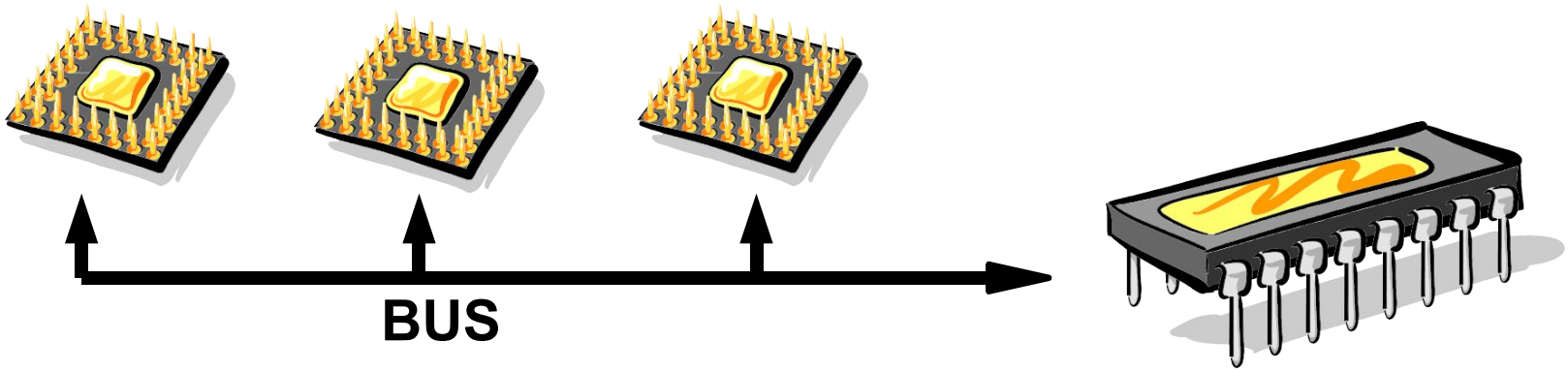
OpenMP сегодня

- Модель OpenMP мощный, но в тоже время компактный
- Стандарт de-facto для программирования систем с общей памятью
- Текущая версия - 4.0
- Спецификация от Июля 2013

Цели OpenMP

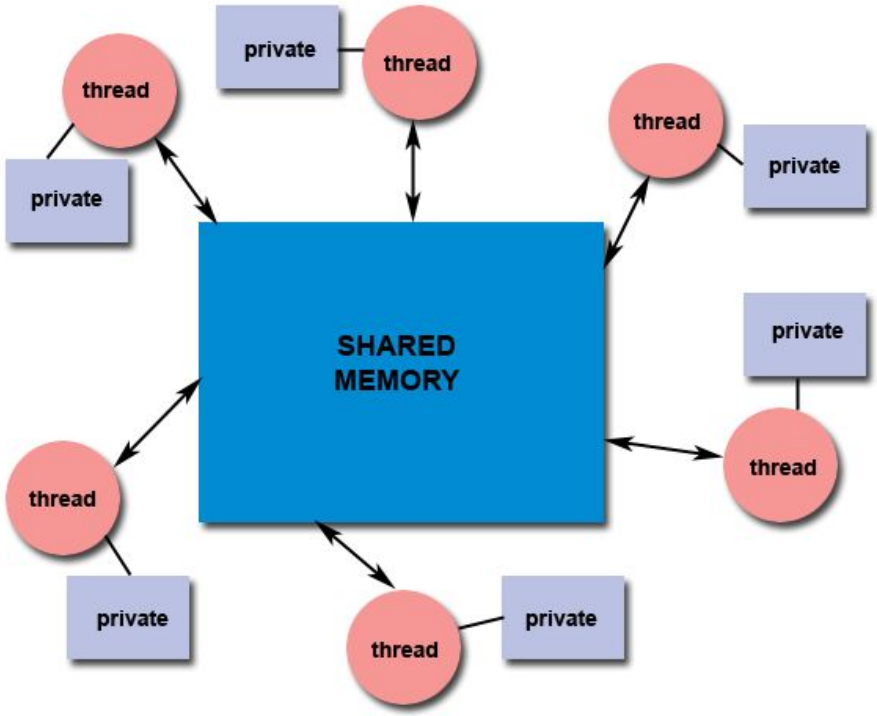
- Быть стандартом для различных архитектур и платформ с распределенной памятью
- Дать простой, но ограниченный набор директив для параллелизации программы.
- Обеспечивать совместимость и возможность инкрементальной параллелизации программы.
- Дать возможность как для мелкозернистого распараллеливания, так и для крупнозернистого.
- Поддержка Fortran (77, 90 и 95), C, и C++

SMP системы



Модель с разделяемой памятью

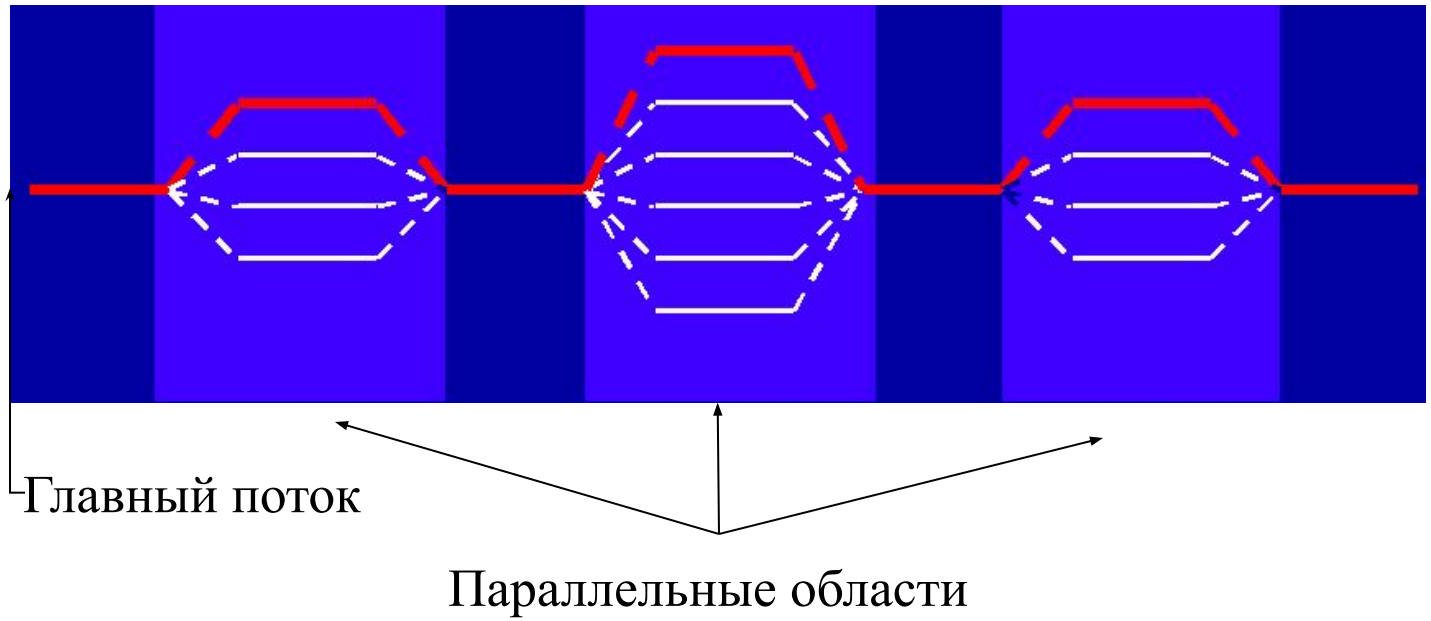
- Все потоки имеют доступ к глобальной разделяемой памяти
- Данные могут быть разделяемые и приватные
- Разделяемые данные доступны всем потокам
- Приватные — только одному
- Синхронизация требуется для доступа к общим данным



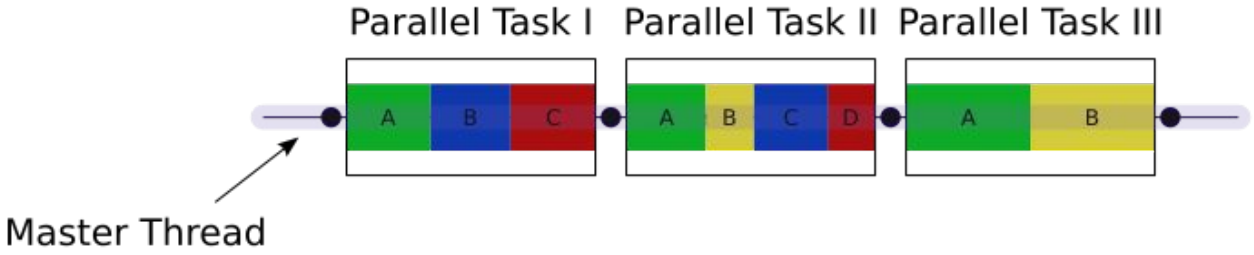
Обзор технологии OpenMP

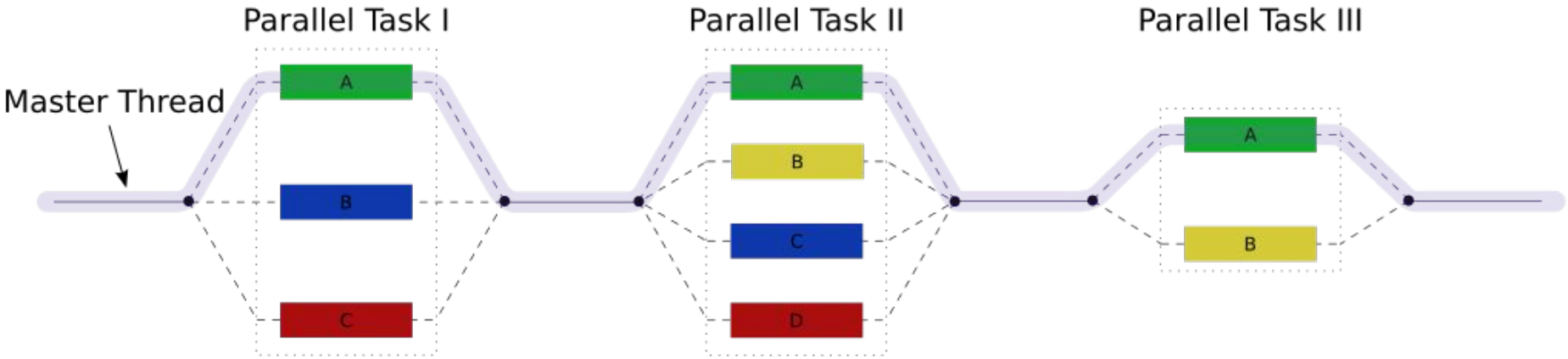
Принцип организации параллелизма

- Использование потоков (общее адресное пространство)
- Пульсирующий (“вилочный”, fork-join) параллелизм

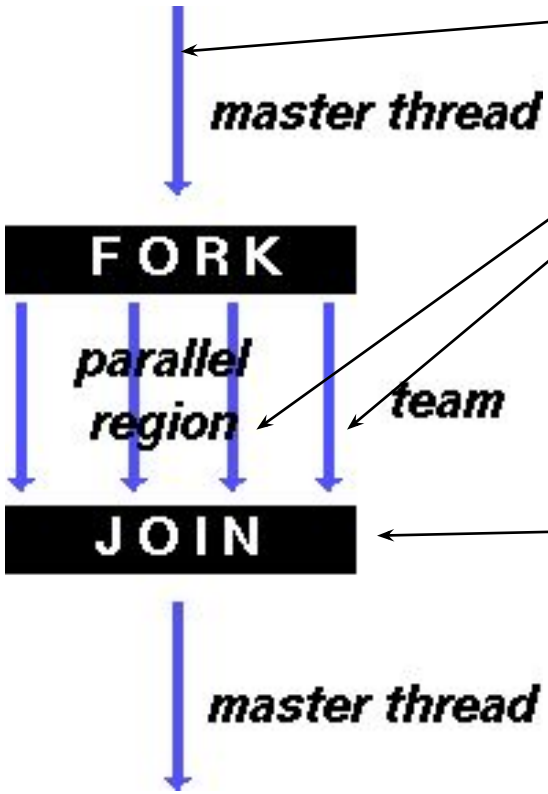


Модель выполнения OpenMP





Модель выполнения OpenMP



Нити (они потоки, они же threads)

Здесь по умолчанию происходит синхронизация. Главная нить не выйдет, пока не сработают остальные. Но это можно отключить клаузой **nowait**

Терминология

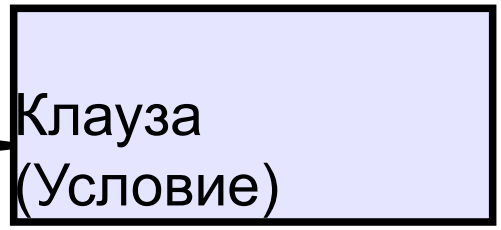
- OpenMP Team := Master + Workers
- Параллельный регион — блок кода, который всеми потоками исполняется одновременно
- Поток мастер (master thread) имеет ID 0
- Все потоки синхронизируются при выходе из параллельного региона
- Параллельные регионы могут быть вложены, но поведение зависит от реализации
- Работа в параллельном регионе распределяется между всеми потоками

Ещё чуть терминологии

- Важные элементы OpenMP:
  - Функции
  - Директивы
  - Клаузы

Параллелизация цикла с помощью OpenMP

```
#pragma omp parallel shared(a,b)
{
#pragma omp for private(i)
  for(i=0; i<10000; i++)
    a[i] = a[i] + b[i];
}
```



Функции

Функции OpenMP носят скорее вспомогательный характер, так как реализация параллельности осуществляется за счет использования директив. Однако в ряде случаев они весьма полезны и даже необходимы.

Функции можно разделить на три категории: функции исполняющей среды, функции блокировки/синхронизации и функции работы с таймерами. Все эти функции имеют имена, начинающиеся с omp_, и определены в заголовочном файле omp.h.

Директивы

Конструкция #pragma в языке Си/Си++ используется для задания дополнительных указаний компилятору.
С помощью этих конструкций можно указать как осуществлять выравнивание данных в структурах, запретить выдавать определенные предупреждения и так далее.

Форма записи:
#pragma *директивы*

Директива omp

Использование специальной ключевой директивы «omp» указывает на то, что команды относятся к OpenMP. Таким образом директивы #pragma для работы с OpenMP имеют следующий формат:

#pragma omp <*директива*> [*клауза* [[,] к*лауза*]…]

Как и любые другие директивы pragma, они игнорируются теми компиляторами, которые не поддерживают данную технологию. При этом программа компилируется без ошибок как последовательная.

Это особенность позволяет создавать хорошо переносимый код на базе технологии OpenMP. Код содержащий директивы OpenMP может быть скомпилирован Си/Си++ компилятором, который ничего не знает об этой технологии. Код будет выполнятся как последовательный,

OpenMP поддерживает директивы private, parallel, for, section, sections, single, master, critical, flush, ordered и atomic и ряд других, которые определяют механизмы разделения работы или конструкции синхронизации.

Директива parallel

Самой главной можно пожалуй назвать директиву parallel. Она создает параллельный регион для следующего за ней структурированного блока, например:

#pragma omp parallel [другие директивы]
    структурированный блок

Директива parallel указывает, что структурный блок кода должен быть выполнен параллельно в несколько **потоков (нитей, threads)**. Каждый из созданных потоков выполнит одинаковый код содержащийся в блоке, но не одинаковый набор команд. В разных потоках могут выполняться различные ветви или обрабатываться различные данные, что зависит от таких операторов как if-else или использования директив распределения работы.

Формат записи директив и клауз OpenMP

- Формат записи

```
#pragma omp имя_директивы [clause,…]
```

- Пример

```
#pragma omp parallel default(shared)private(beta,pi)
```

Директива

Клаузы (clause)

Директива – описывает, что делать

Клауза – это что-то вроде настроек директив

А ещё бываю функции – это просто команды делающие что-то полезное, типа получения номера нити, но это не директивы.

Пример

Чтобы продемонстрировать запуск нескольких потоков, распечатаем в распараллеливаемом блоке текст:

```
#pragma omp parallel
{
 printf(“OpenMP Test\n”);
 }
```

На 4-х ядерной машине мы можем ожидать увидеть следующей вывод:

```
OpenMP Test
OpenMP Test
OpenMP Test
OpenMP Test
```

Директива for

Рассмотренный нами выше пример демонстрирует наличие параллельности, но сам по себе он бессмыслен. Теперь извлечем пользу из параллельности. Пусть нам необходимо извлечь корень из каждого элемента массива и поместить результат в другой массив:

```
void VSqrt(double *src, double *dst, int n)
 {
      for (int i = 0; i < n; i++)
            dst[i] = sqrt(src[i]);
 }
```

Если мы напишем:

```
#pragma omp parallel
{
for (int i = 0; i < n; i++)
      dst[i] = sqrt(src[i]);
 }
```

То мы вместо ускорения впустую проделаем массу лишней работы. Мы извлечем корень из всех элементов массива в каждом потоке.

Директива for

Для того, чтобы распараллелить цикл нам необходимо использовать директиву разделения работы «for». Директива #pragma omp for сообщает, что при выполнении цикла for в параллельном регионе итерации цикла должны быть распределены между потоками группы:

```
#pragma omp parallel
{
	#pragma omp for
	for (int i = 0; i < n; i++)
		dst[i] = sqrt(src[i]);
}
```

Теперь каждый создаваемый поток будет обрабатывать только отданную ему часть массива. Например, если у нас 8000 элементов, то на машине с четырьмя ядрами работа может быть распределена следующим образом. В первом потоке переменная i принимает значения от 0 до 1999. Во втором от 2000 до 3999. В третьем от 4000 до 5999. В четвертом от 6000 до 7999. Теоретически мы получаем ускорение в 4 раза. На практике ускорение будет чуть меньше из-за необходимости создать потоки и дождаться их завершения. В конце параллельного региона выполняется неявная (мы её специально не писали») барьерная синхронизация. Иначе говоря, достигнув конца региона, все потоки блокируются до тех пор, пока последний поток не завершит свою работу.

Директива for

Можно использовать сокращенную запись, комбинируя несколько директив в одну управляющую строку. Приведенный выше код будет эквивалентен:

```
#pragma omp parallel for
for (int i = 0; i < n; i++)
	dst[i] = sqrt(src[i]);
```

Некоторые функции OpenMP

omp_get_thread_num(); - возвращает номер нити типом int. Вне параллельной секции всегда вернёт 0

omp_get_num_threads(); - возращает общее количество нитей типом int. Вне параллельной секции всегда вернёт 1.

omp_get_wtime(); - возращает время в секундах типа double с момента некого «момента в прошлом». «Момент в прошлом» является произвольным, но он гарантировано не меняется с момента зпуска программы. Т.е. для подсчёта времени нужно вычесть одно время из другого.

Пример

Последовательный код

```
void main(){  double
x[1000];
for(i=0; i<1000; i++){
calc_smth(&x[i]);
}
}
```

Параллельный код

```
void main(){  double x[1000];
#pragma omp  parallel for
for(i=0; i<1000; i++){
calc_smth(&x[i]);
}
}
```

Условие reduction - Пример

- Пример

```
#pragma ompparallel
{
  #pragma for shared(x, sum) private(i)
     for(i=0; i<10000; i++)
                sum+ x[i];
  sum=
  }
```

- Нужна осторожность при работе с переменной
- SUM

  При использовании условия «reduction» компилятор заботится о синхронизации доступа к SUM

Клауза reduction

- reduction (operator : list)
  - Редукционные переменные должны быть разделяемыми (shared)

```
#pragma ompparallel
{
#pragma forshared(x)                    private(i)  reduction(+:sum)
for(i=0; i<10000;                       i++)
sum += x[i];
}

#pragma ompparallel
{                                       private(i)  reduction(min:gmin)
#pragma forshared(x)
      for(i=0; i<10000; i++)
          gmin = min(gmin, x[i]);
      }
```