Computer Science МФТИ

Intro to OpenMP

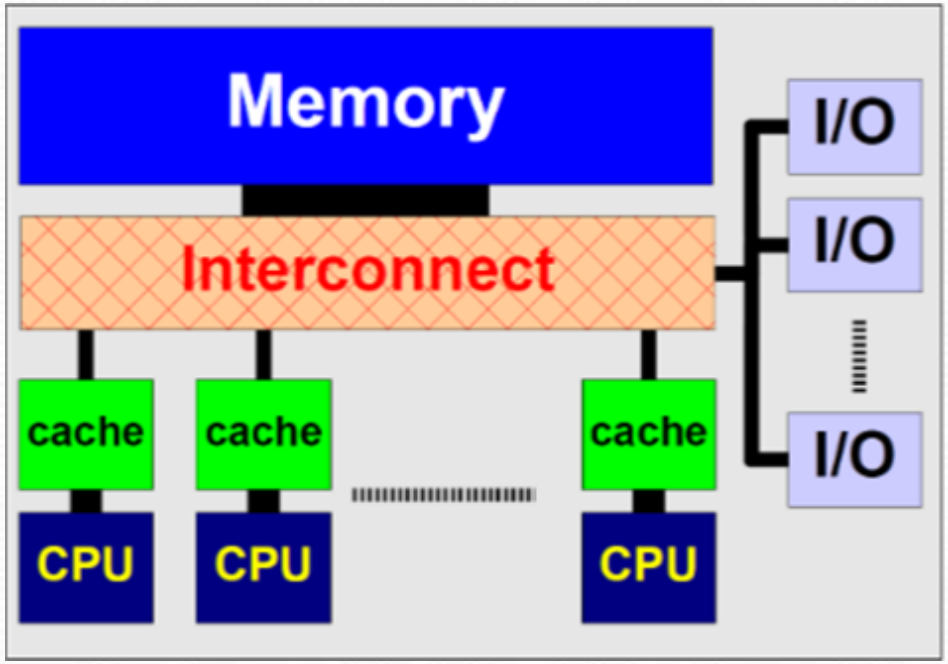
Субботина Анна
МФТИ

Содержание лекции

- Архитектура параллельных вычислительных систем
- Модели параллельного программирования
- Технология OpenMP
  - История
  - Модель данных
  - Модель исполнения
  - Директивы

Параллельные архитектуры

- Uniform Memory Access (UMA) или Symmetric Multi Processor (SMP)



КЕШ-когерентна

Преимущества:

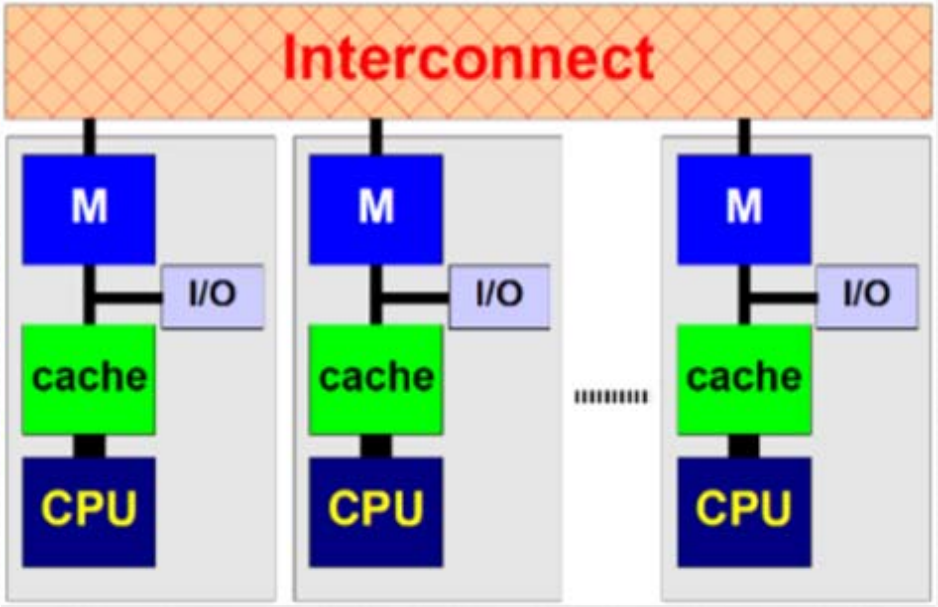
- Простая логика использования и управления
- Эффективное использование ресурсов
- Быстрый доступ к памяти (шина)

Недостатки:

- Дорого
- Тяжело Масштабируема

Параллельные архитектуры

- Non Uniform Memory Access (NUMA) или "Distributed Memory" или NORMA (No Remote Memory Access)



Не КЕШ-когерентна

Преимущества:

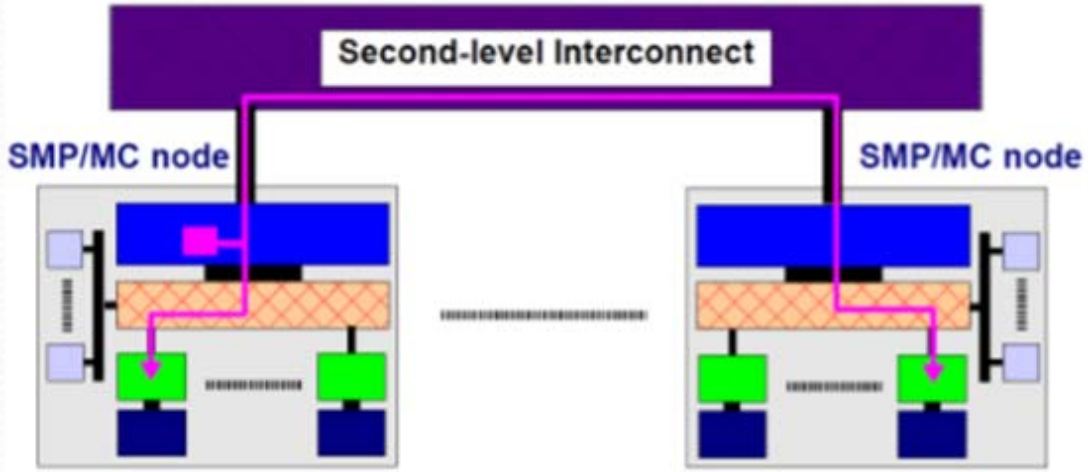
- Недорого
- Легко масштабируема

Недостатки:

- Не так легко использовать и управлять
- Не настолько эффективное использование ресурсов, как в UMA
- Доступ к памяти медленный (Ethernet, Infiniband)

Параллельные архитектуры

- Гибридная архитектура (Hybrid Architecture)



Second level interconnect:

- Ethernet, Infiniband

Преимущества и недостатки обеих архитектур:

- UMA в рамках одного узла
- NUMA между узлами

Самая распространенная архитектура современности
Типичный представитель – вычислительный кластер

Модели параллельного программирования

Последовательная модель:

- Последовательная программа для автоматического распараллеливания компилятором или специальными программными средствами.
  - Преимущество – знакомая парадигма программирования
  - Недостаток – ограниченные возможности автоматического распараллеливания

- Пример: Intel Parallel Studio

Модели параллельного программирования

Модель передачи сообщений:

- Работающее приложение состоит из набора процессов с различными адресными пространствами. Процессы обмениваются сообщениями с помощью явных send-receive операций.
  - Преимущество – полный контроль над выполнением
  - Недостаток – сложность и кропотливость программирования
- Пример: MPI

Модели параллельного программирования

Модель разделяемой памяти:

- Приложение состоит из набора thread'ов, использующих разделяемые переменные и примитивы синхронизации
- Явные нити исполнения: программирование с использованием библиотечных или системных вызовов для thread'ов
  - Преимущество – переносимость
  - Недостаток – сложность программирования
  - Пример: средства System V IPC
- Программирование на языке высокого уровня с применением прагм.
  - Преимущество – легкость программирования
  - Недостаток – сложность контроля над выполнением
  - Пример: OpenMP

Модели параллельного программирования

Модель разделенных данных:

- Приложение состоит из набора процессов, каждый работает со своим набором данных, обмена информацией при работе нет.
  - Преимущество – легкость реализации
  - Недостаток – производительность зависит от конкретной реализации, малый круг задач

- Пример: задачи дешифрования и рендеринга

Что такое OpenMP?

- Открытый стандарт (API) для распараллеливания многопоточных приложений на многопроцессорных системах с общей памятью
- Состоит из трех основных компонент API:
  - директив компилятора
  - библиотечных функций/процедур
  - переменных окружения

Что такое OpenMP?

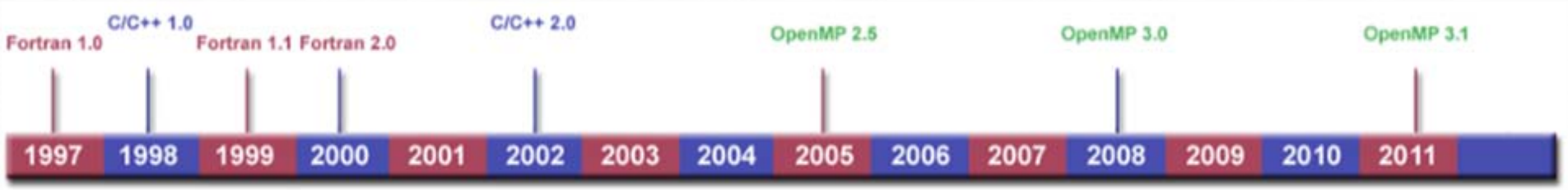
- Портируемый:
  - определен для языков C (C++) и Fortran
  - мульти-платформный, поддерживается на большинстве стандартных ОС (Unix/Linux и Windows NT)
- Широко распространенный:
  - разрабатывается и поддерживается группой крупных производителей вычислительной техники и программного обеспечения

Какие ограничения?

OpenMP

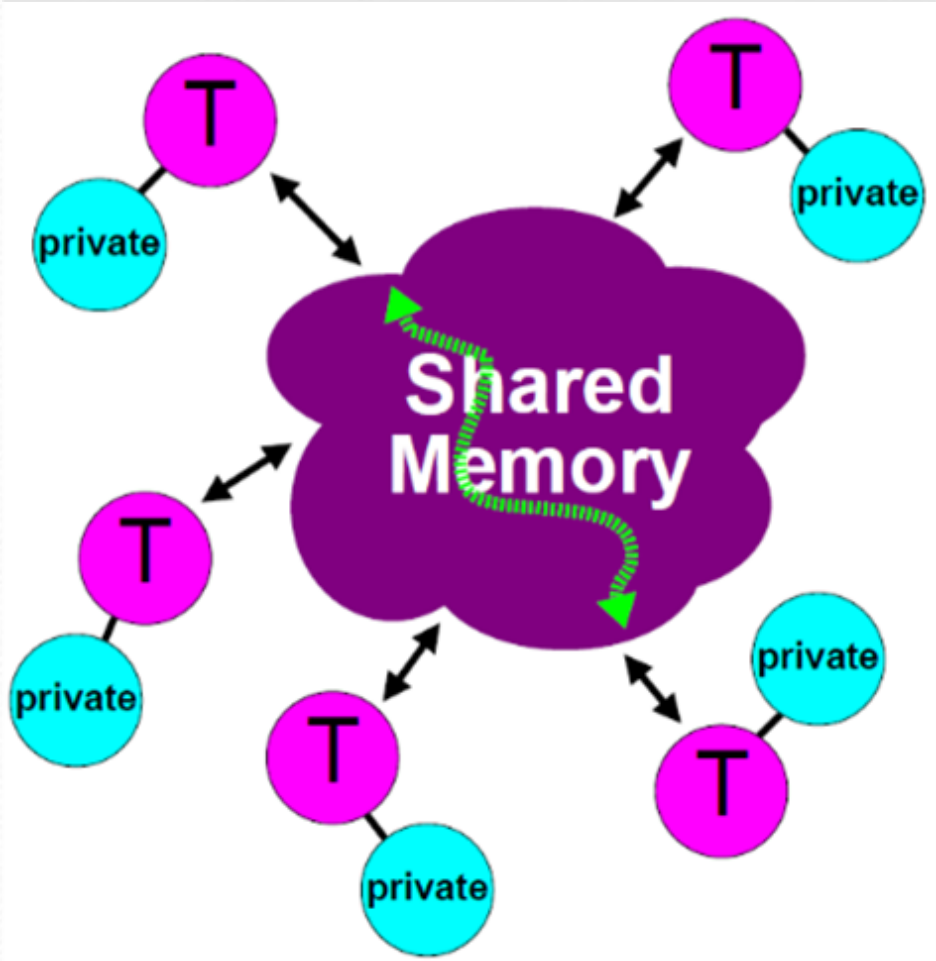
- Не подходит для систем с распределенной памятью
- Не дает автоматического распараллеливания при компиляции
- Не гарантирует защиту от зависимостей и конфликтов данных, условий гонки и тупиков
- Не гарантирует синхронного ввода/вывода в файл, синхронизизацию должен обеспечить программист

История OpenMP



- 90-е – начало массового появления SMP машин, различные производители разрабатывали свои системы программирования
- 94-й – начали выделять общее во всех подходах, и представить в качестве стандарта
- 97-й – можно считать годом рождения OpenMP

Модель данных OpenMP

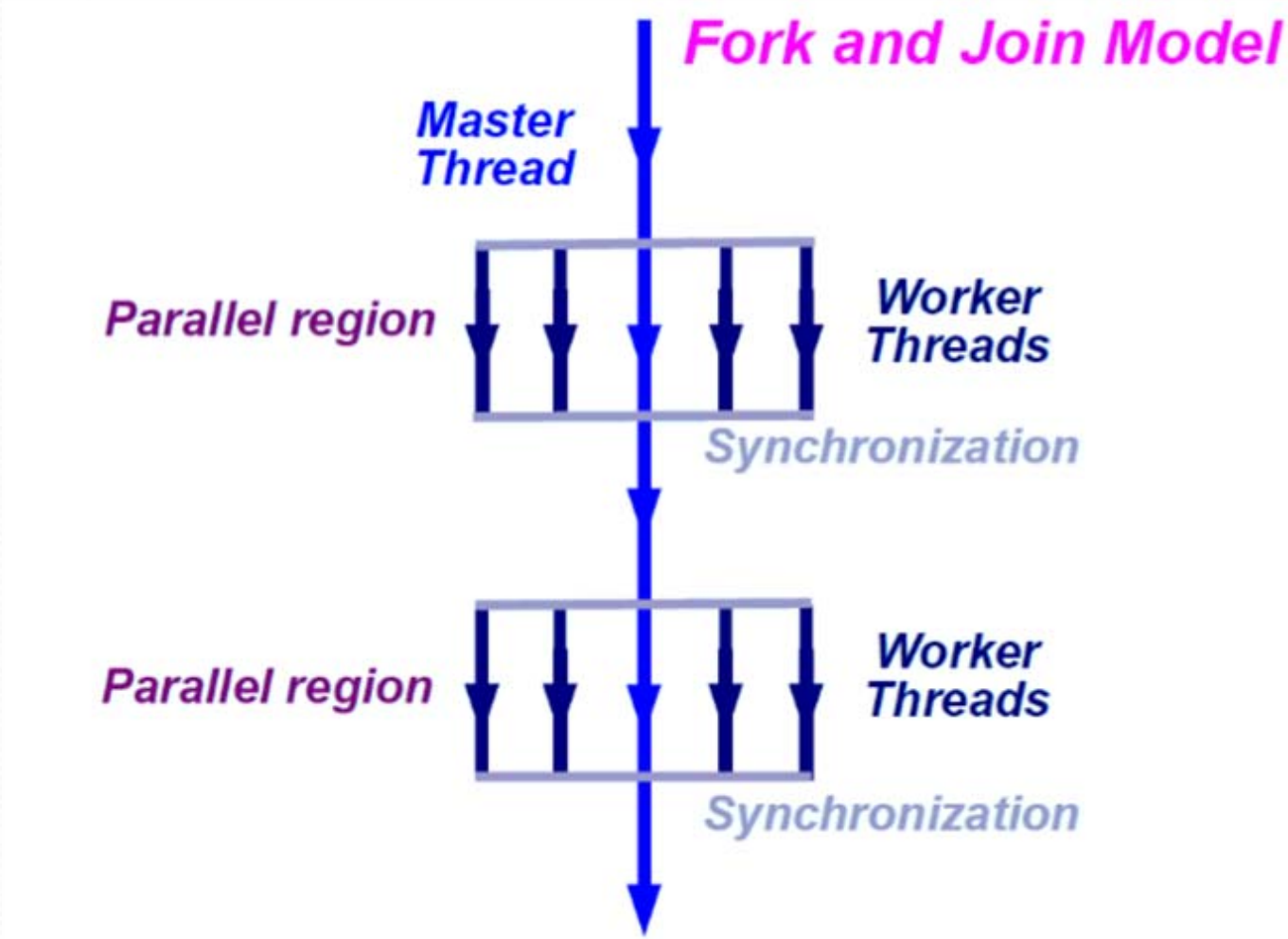


- Все потоки имеют доступ к одной, общей для всех, разделяемой памяти
- Данные могут быть разделяемые и локальные (private)
- Разделяемые данные доступны всем потокам
- Локальные – только потоку-владельцу этих данных
- Транспорт данных прозрачен для программиста
- Имеет место неявная синхронизация

Атрибуты данных

- В OpenMP данные должны иметь тип:
- Изначально, существует два типа данных:
  - Разделяемые – один экземпляр данных
    - Все потоки могут читать/писать данные одновременно, если данные не защищены специальным конструктом OpenMP
    - Все изменения над данными видны всем потокам
- Локальные – у каждого потока своя копия
  - Другие потоки не имеют прав доступа к данным
  - Изменения над данными видны только потоку-владельцу данных

Модель исполнения OpenMP



Пример параллельного исполнения

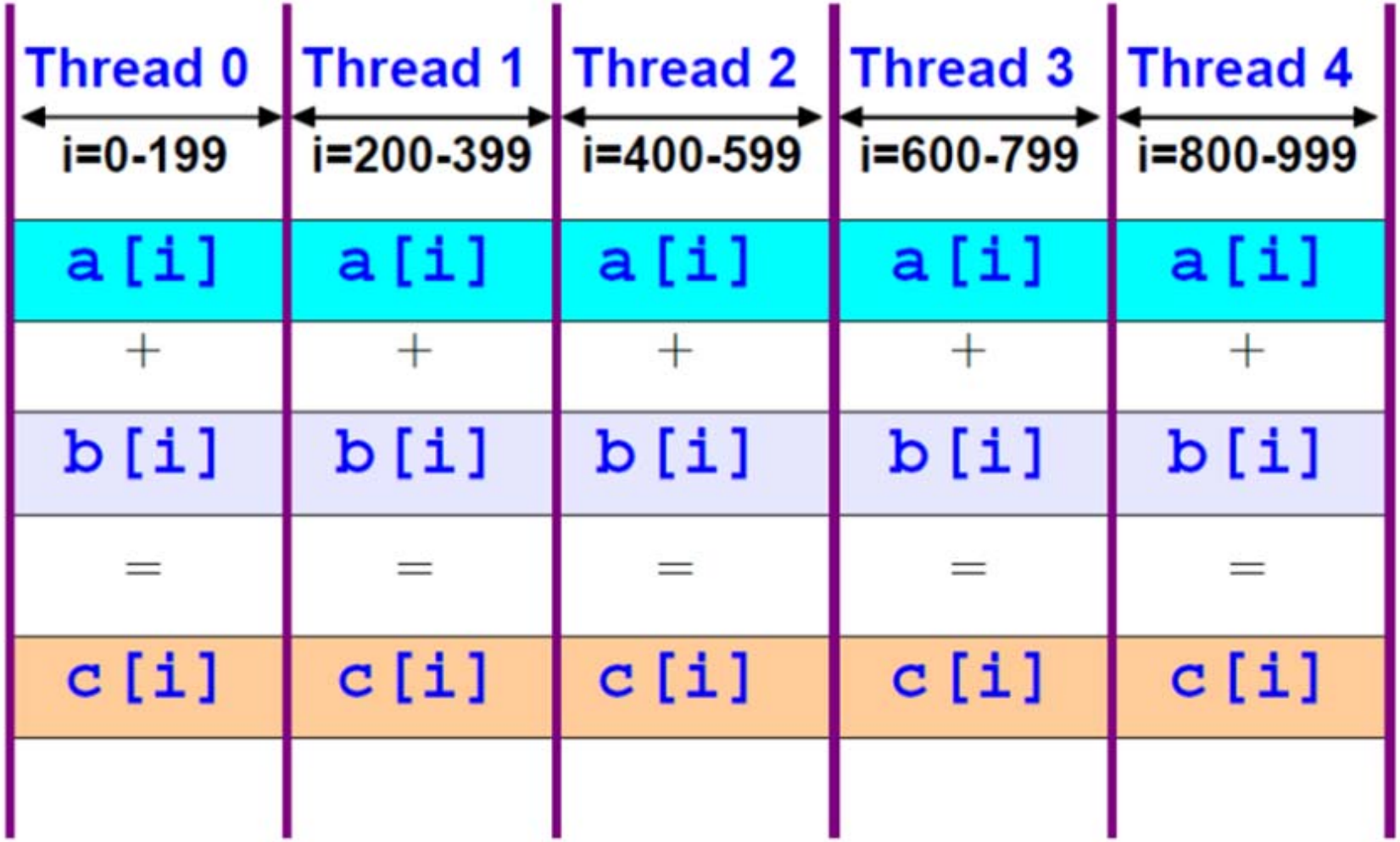
for-цикл последовательного исполнения итераций

```
for (int i=0; i<n; i++)
        c[i] = a[i] + b[i];
```

for-цикл параллельного исполнения итераций с помощью OpenMP pragma

```
#pragma omp parallel for
for (int i=0; i<n; i++)
        c[i] = a[i] + b[i];
```

Пример параллельного исполнения



Параллелизм в OpenMP

- OpenMP Team := Master + Workers
- Parallel Region (параллельный блок) – это блок кода исполняемый всеми потоками одновременно
  - ID master thread всегда равен нулю
  - Параллельные блоки могут быть вложенными, но их поддержка зависит от реализации
  - Параметр “if” может использоваться для защиты параллельного блока; в случае, если условие ложное, код исполняется последовательно

Параллелизм в OpenMP

- При выполнении обычного кода (вне параллельных областей) программа исполняется одним потоком (master thread)
- При появлении директивы #parallel происходит создание “команды” (team) потоков для параллельного выполнения вычислений
- После выхода из области действия директивы #parallel происходит синхронизация, все потоки, кроме master, уничтожаются
- Продолжается последовательное выполнение кода (до очередного появления директивы #parallel)

Формат директив

- C: директивы чувствительны к регистру
  - Синтаксис: **#pragma omp directive [clause ...]**

- Fortran: директивы не чувствительны к регистру
  - Синтаксис: **sentinel directive [clause ...]**
  - sentinel directive – одна из:
    - !$OMP или C$OMP или *$OMP

Параметры OpenMP

- Большинство директив OpenMP поддерживают параметры (clause)
  - Данные параметры обеспечивают поддержку дополнительной информации к директивам
- Например: параметр "private(a)" для директивы "for":
  - #pragma omp for private(a)
- Использование различных параметров зависит от директив

Параметр "if"

if (scalar expression)

✓Исполнение параллельно, если условие верно
✓ В противном случае - исполнение последовательное

```
#pragma omp parallel if (n > some_threshold) \
shared(n,x,y) private(i)
{
        #pragma omp for
        for (i=0; i<n; i++)
                x[i] += y[i];
} /*-- End of parallel region --*/
```

Барьер

Предположим мы запустили оба цикла параллельно:

```
for (i=0; i < N; i++)
        a[i] = b[i] + c[i];
```

```
for (i=0; i < N; i++)
        d[i] = a[i] + b[i];
```

Результат может оказаться неправильным (однажды)

Почему?

Барьер

Необходимо обновить все элементы массива a[ ] прежде, чем использовать их в качестве входных данных

```
for (i=0; i < N; i++)
      a[i] = b[i] + c[i];
```

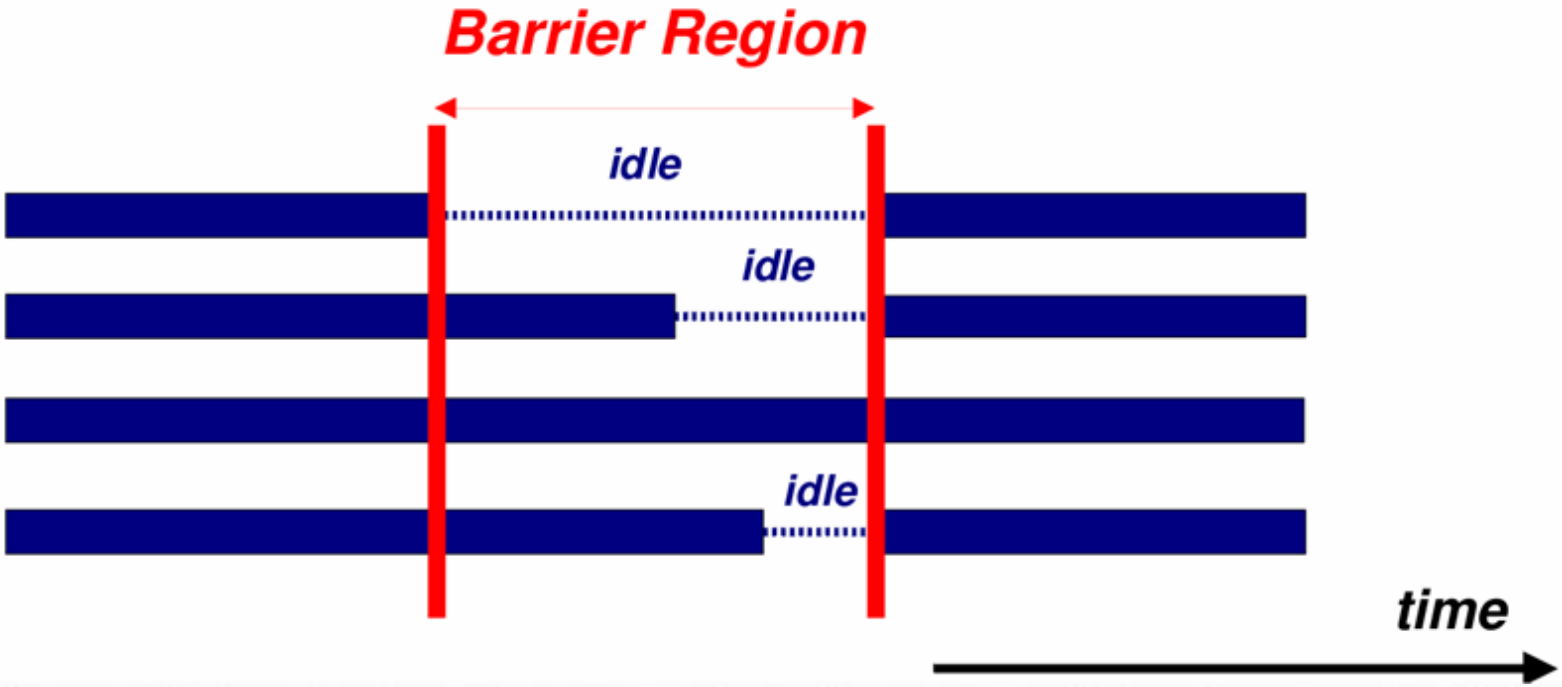
ждем!

барьер

```
for (i=0; i < N; i++)
      d[i] = a[i] + b[i];
```

Все потоки ждут в точке брьера друг друга и продолжат исполнение только когда ВСЕ потоки достигнут барьера

Барьер



Синтаксис барьеров:

```
#pragma omp barrier
```

```
!$omp barrier
```

Когда использовать барьеры?

- Если обновление данных асинхронно и целостность данных подвергается риску
- Например:
  - при чтении и записи одних и тех же общих данных
  - при решении уравнений - после каждого посчитанного шага
- К сожалению, барьеры дорого обходятся в смысле производительности

Параметр "nowait"

- Используется для минимизации синхронизации, некоторые директивы OpenMP поддерживают параметр "nowait"
- При данном параметре, потоки не сихронизируются (ждут друг друга) в конце текущего блока
- В Fortran-е параметр "nowait" добавляется при закрытии блока
- В C, "nowait" - параметр директивы

```
#pragma omp for nowait
{
.
}
```

```
!$omp do
.
.
!$omp end do nowait
```

Пример

```
#pragma omp parallel if (n>limit) default(none) \
shared(n,a,b,c,x,y,z) private(f,i,scale)
{
        f = 1.0;
#pragma omp for nowait
        for (i=0; i<n; i++)
                a[i] = b[i] + c[i];
#pragma omp for nowait
        for (i=0; i<n; i++)
                z[i] = x[i] + y[i];
        ....
#pragma omp barrier

        scale = sum(a,0,n) + sum(z,0,n) + f;
        ....
} /*-- End of parallel region --*/
```

Исполняется всеми потоками

параллельный цикл (раделение работы)

параллельный цикл (раделение работы)

синхронизация

Исполняется всеми потоками

Вопросы?