Уральский федеральный университет

Параллельные вычисления

Технология OpenMP

Созыкин Андрей Владимирович

к.т.н.
Заведующий кафедрой высокопроизводительных компьютерных технологий
Институт математики и компьютерных наук

OpenMP

OpenMP – Open Multi-Processing

Технология параллельного программирования для систем с общей памятью

Отличительная особенность – автоматизация распараллеливания

Основные компоненты:

- Директивы компилятора
- Функции
- Переменные окружения

Автоматизация распараллеливания

Разрабатывать параллельные программы сложно

Автоматическое распараллеливание:

- Возможно только для простых случаев
- Компилятор не всегда может найти участки кода которые можно распараллелить
- Не всегда получается автоматически определить, безопасно ли распараллеливать код

Подсказки компилятору:

- Программист указывает компилятору с помощью директив, какой код нужно распараллелить
- Компилятор распараллеливает автоматически то, что ему показали

Основные компоненты OpenMP

Директивы компилятора

Функции

Переменные окружения

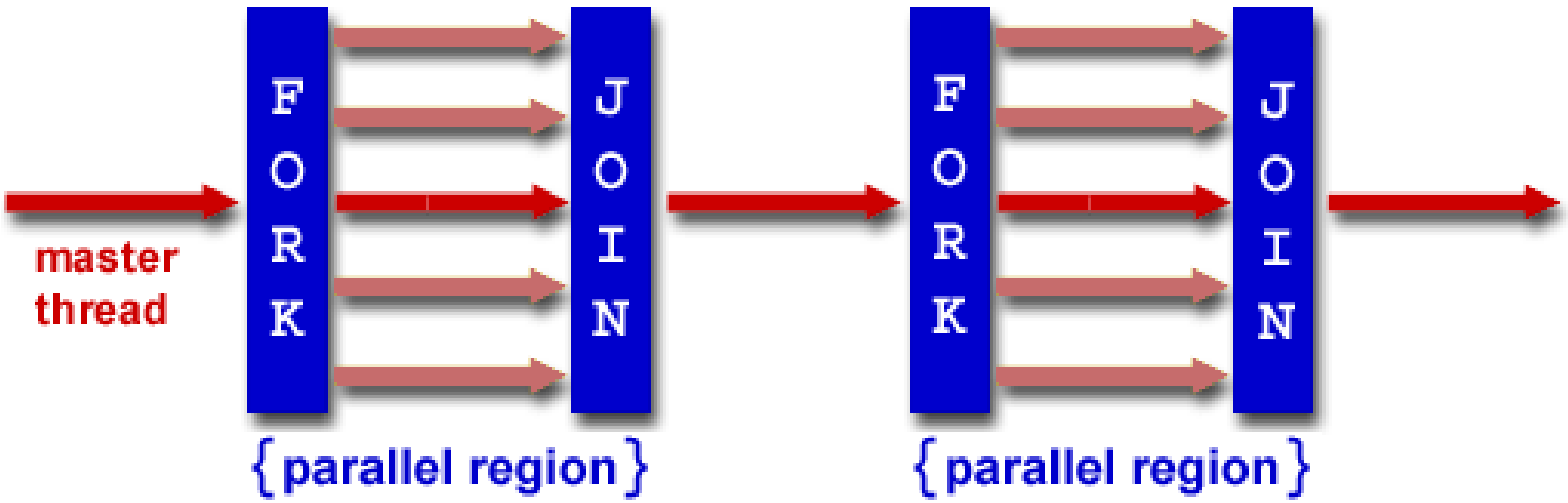
Hello, world!

```
int main(){
    #pragma omp parallel
    {
        int nthread = omp_get_num_threads();
        int thread_id = omp_get_thread_num();
        std::cout << "Hello, world from thread "
           << thread_id << " of "
           << nthread << std::endl;
    }
}
```

Hello, world!

```
Hello, world from thread 0 of 8
Hello, world from thread 1 of 8
Hello, world from thread 2 of 8
Hello, world from thread 7 of 8
Hello, world from thread 3 of 8
Hello, world from thread 4 of 8
Hello, world from thread 6 of 8
Hello, world from thread 5 of 8
```

Модель Fork-Join



http://www.techdarting.com/2013/07/openmp-getting-started.html

Как компилировать

GCC:

- g++ -fopenmp

Intel:

- icpc -openmp

LLVM:

- clang++ -fopenmp

MS Visual Studio:

- Project → Properties → Configuration Properties → C/C++ → Language → OpenMP Support → Generate Parallel Code

Без ключа OpenMP

Предупреждение:

```
reduction.cpp:26:0: warning: ignoring #pragma omp
parallel [-Wunknown-pragmas]
 #pragma omp parallel for shared(a) reduction(+:sum)
```

Программа будет выполнять последовательно

Параллельная OpenMP программа – это также корректна последовательная программа

- Если не используются функции OpenMP

Сколько потоков запуститься?

По-умолчанию OpenMP выбирает автоматически

- Количество виртуальных процессоров в ОС

Переменная окружения OMP_NUM_THREADS

- `export OMP_NUM_THREADS=N`

Функция OpenMP

- `omp_set_num_threads(N);`

Опция директивы компилятора num_threads

- `#pragma omp parallel num_threads(N)`

Сколько потоков запуститься?

```
#pragma omp parallel num_threads(4)
{
    int nthread = omp_get_num_threads();
    int thread_id = omp_get_thread_num();
    std::cout << "Hello, world from thread " << thread_id
        << " of " << nthread << std::endl;
}
```

Сколько потоков запуститься?

```
omp_set_num_threads(4);
#pragma omp parallel
{
    int nthread = omp_get_num_threads();
    int thread_id = omp_get_thread_num();
    std::cout << "Hello, world from thread " << thread_id
        << " of " << nthread << std::endl;
}
```

Рекомендации

Не задавайте количество потоков в программе в продуктивном коде!

- При переносе на более мощный компьютер не нужно будет менять программу

Если необходимо запустить на разном количестве потоков, используйте переменную окружения OMP_NUM_THREADS

Распределение нагрузки между потоками

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

Распределение нагрузки между потоками

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

Распределение нагрузки между потоками

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

Каждый поток выполнит все итерации цикла!

Сокращенная запись

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

Распределение нагрузки между потоками

Как итерации цикла распределяются между потоками?

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

Schedule

```
#pragma omp parallel for schedule(type, chunk_size)
```

`schedule(static, chunk_size)`

- Статическое распределение порций цикла

`schedule(dynamic, chunk_size)`

- Динамическое распределение порций цикла

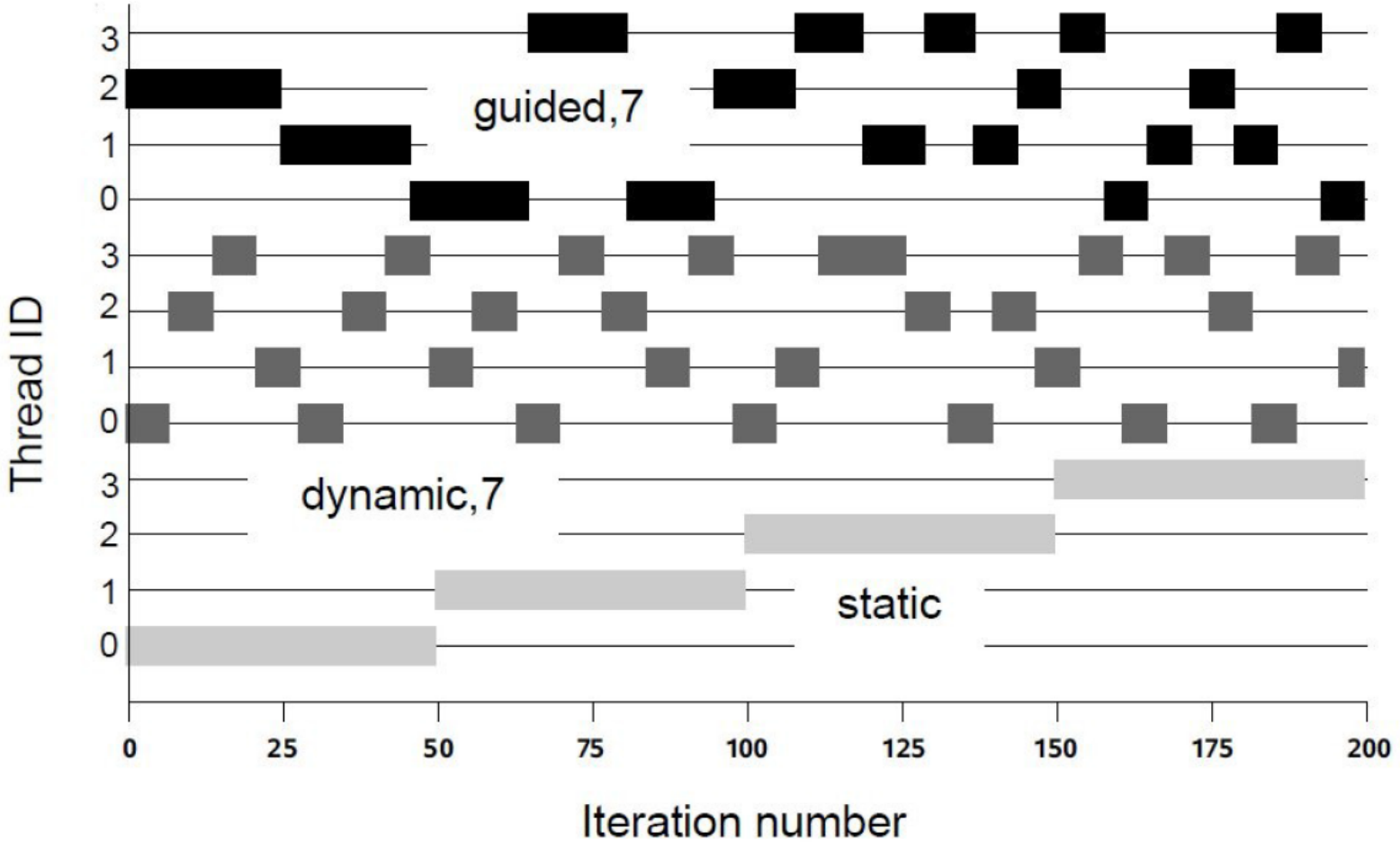
`schedule(guided, chunk_size)`

- Динамическое распределение порций цикла, размер порции постоянно уменьшается

`schedule(runtime)`

- Используется значение из переменной окружения OMP_SCHEDULE

Schedule


guided,7
dynamic,7
static
Thread ID
Iteration number
0
25
50
75
100
125
150
175
200
3
2
1
0

Schedule

Какое распределение лучше использовать?

Schedule

Какое распределение лучше использовать?

Протестировать разные варианты и выбрать лучший!

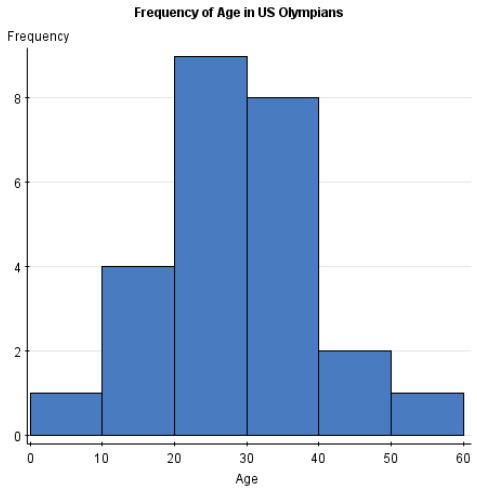
Зависимости

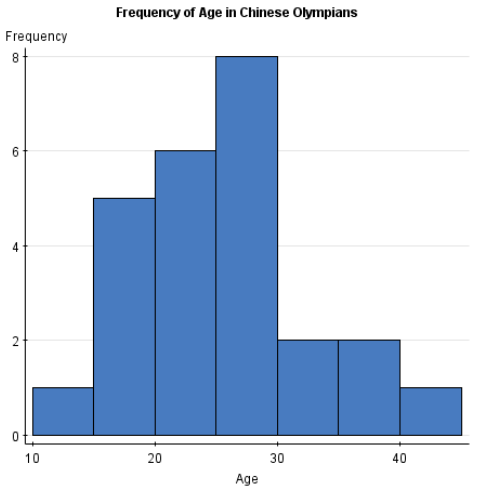
Компилятор не проверяет зависимости при распараллеливании

- За безопасность распараллеливания отвечает программист!

Построим гистограмму возрастов победителей Олимпийских игр

```
for (int i = 0; i < N; i++)
   hist_age[age[i]]++;
```





Типы зависимостей

Read-after-write

- flow dependency
- ```
  for (j=1; j<MAX; j++)
      A[j]=A[j-1]+1;
  ```

Write-after-read

- anti-dependency
- ```
  for (j=1; j<MAX; j++)
      A[j-1]=A[j]+1;
  ```

Write-after-write

- output dependency
- ```
  for( i=0; i<N; i++)
      a[i%2] = b[i] + c[i];
  ```
  ```

Редукция

Какое распараллелить такой цикл?

```
double sum = 0;
for (int i = 0; i < N; ++i) {
    sum += a[i] * b[i];
}
```

Редукция

```
double sum = 0;
#pragma omp parallel for reduction(+:sum)
for (int i = 0; i < N; ++i) {
    sum += a[i] * b[i];
}
```

Допустимые операторы в редукции

Operator	Initialization value
+	0
*	1
-	0
&	~0
|	0
^	0
&&	1
||	0

Опции для переменных

shared(list)

- Задает список общих для всех потоков переменных

private(list)

- Задает список частных для потока переменных

firstprivate(list)

- Задает список частных для потока переменных, которые инициализируются значениями из однопоточной части

lastprivate(list)

- Задает список частных для потока переменных, которые сохраняют значение после завершения параллельной секции

Опции для переменных

По умолчанию все переменные общие (shared)

- Возможны условия гонок

Счетчик цикла в директиве for автоматически становится частным

Переменные, объявленные в параллельной секции, являются частными

Условия гонок

```
int nthread, thread_id;
#pragma omp parallel
{
    nthread = omp_get_num_threads();
    thread_id = omp_get_thread_num();
    std::cout << "Hello, world from thread "
      << thread_id << " of " << nthread << std::endl;
}
```

Переменные private

```
int nthread, thread_id;
#pragma omp parallel private(nthread, thread_id)
{
    nthread = omp_get_num_threads();
    thread_id = omp_get_thread_num();
    std::cout << "Hello, world from thread "
      << thread_id << " of " << nthread << std::endl;
}
```

Переменные внутри секции

```
#pragma omp parallel
{
    int nthread = omp_get_num_threads();
    int thread_id = omp_get_thread_num();
    std::cout << "Hello, world from thread "
      << thread_id << " of " << nthread << std::endl;
}
```

Оценка времени выполнения

Функция omp_get_wtime();

- Время в секундах

Пример использования:

```
double start =  omp_get_wtime();
...
double stop = omp_get_wtime();
std::cout << stop – start << " seconds.";
```

Необходимо подключить заголовочный файл:

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

Другие директивы

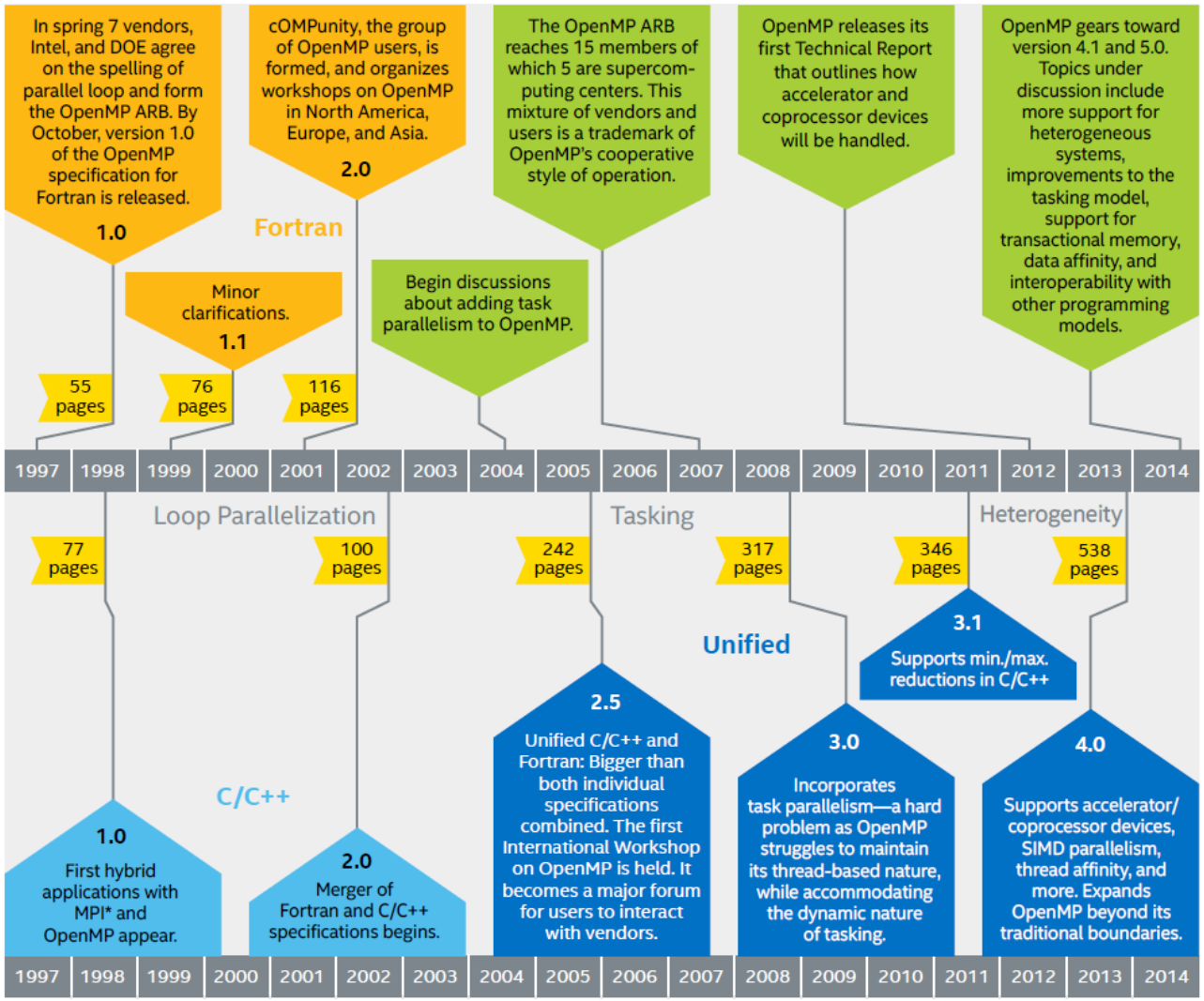
```
#pragma omp single

#pragma omp master

#pragma omp critical

#pragma omp atomic
```

Версии OpenMP



Итераторы C++

Распараллеливание возможно для итераторов C++

- random access iterators
- Появилось в версии OpenMP 3.0

```
std::vector<double> vec;
std::vector<double>::iterator it;
#pragma omp parallel for shared(vec)
for (it = vec.begin(); it < vec.end(); ++it) {
  int thread_id = omp_get_thread_num();
  std::cout << "Thread id = " << thread_id <<
  ", vector element = " << *it << std::endl;
}
```

Векторизация

OpenMP 4.0 включает директиву для использования векторизации:

```
#pragma omp parallel simd
```

Возможно совместное использование многопоточности и векторизации:

```
#pragma omp parallel for simd reduction(+:sum)
for (int i = 0; i < N; ++i) {
    sum += a[i] * b[i];
}
```

Связанный список

Как распараллелить такую программу:

```
p=head;
while (p) {
    process(p);
    p = p->next;
}
```

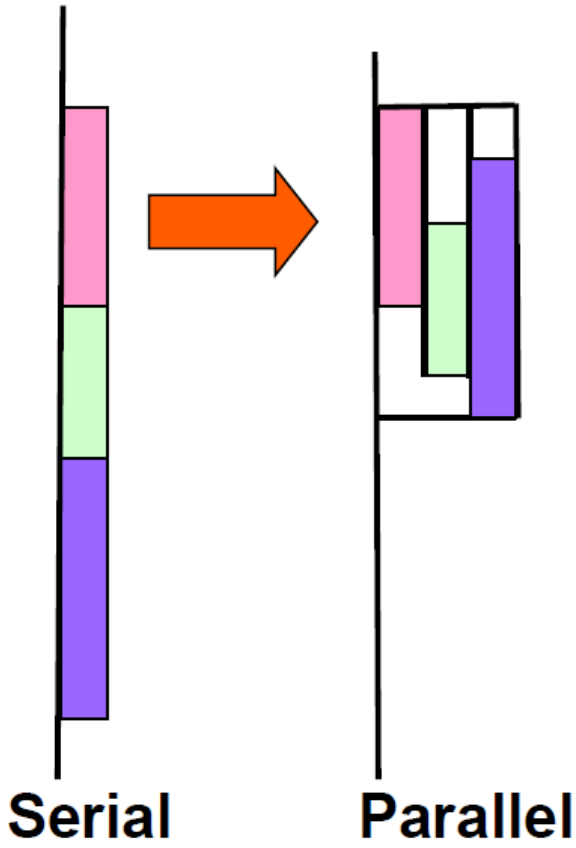
Директива task

Директива распределения работы в OpenMP

Создает отдельную задачу:

- Задача ставится в очередь
- Может быть выполнена параллельно другим потоком
- Может быть выполнена последовательно одним потоком
- Аналог async в C++11

Обязательно использование директивы parallel!



Связанный список и task

```
#pragma omp parallel
{
    #pragma omp single
    {
        p = head;
        while (p) {
            #pragma omp task firstprivate(p)
            process(p);
            p = p->next;
        }
    }
}
```

Привязка потоков к ядрам

До OpenMP 3.1 зависела от компилятора

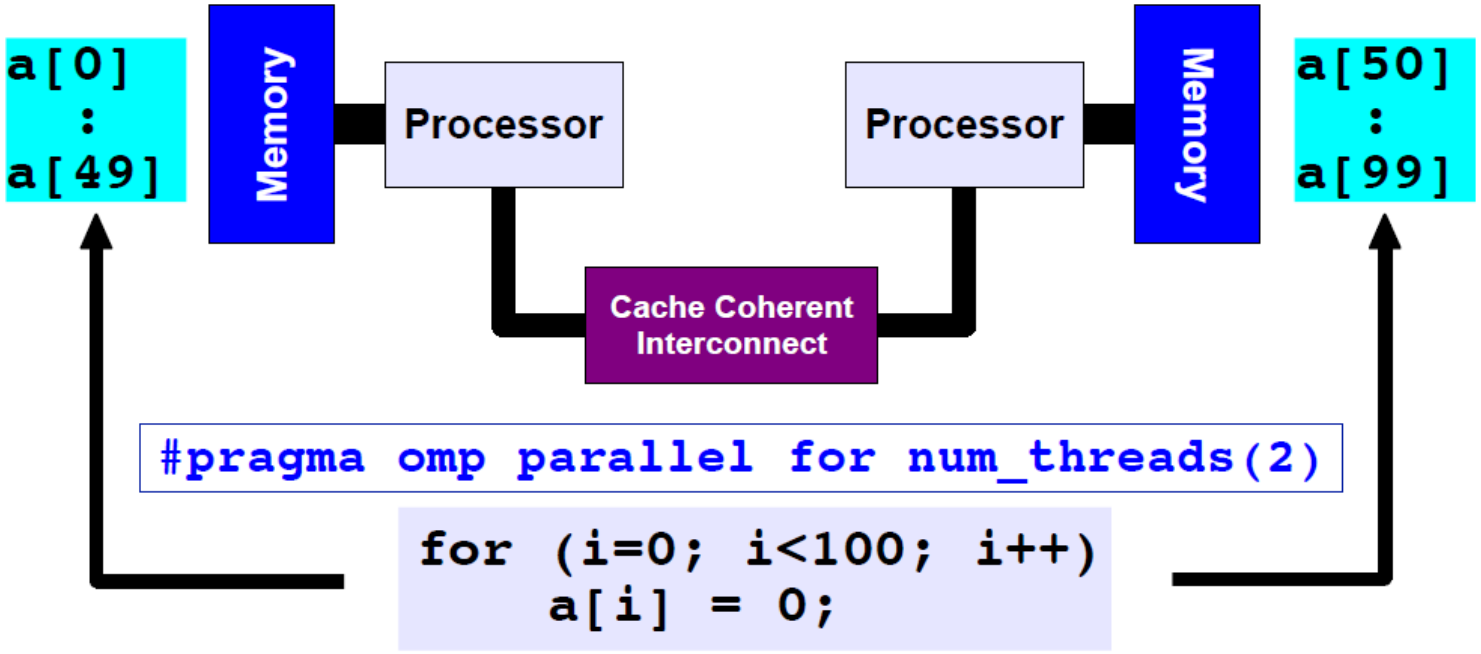
В OpenMP 3.1

- Переменная окружения OMP_PROC_BIND (true – есть привязка, false – нет привязки)

В OpenMP 4

- Переменная окружения OMP_PLACES – явно указывает, где разместить потоки
- Абстрактные значения: `threads, cores, sockets`
- Явный перечень:
  `OMP_PLACES=0,8,1,9,2,10,3,11,4,12,5,13,6,14,7,15`

Размещение данных в памяти NUMA



First Touch – появилось в OpenMP 4.0

Домашнее задание

Реализовать параллельную версию алгоритма K-Means с помощью OpenMP

- Будет предоставлена последовательная версия
- Используйте потоки и векторизацию

Измерьте время выполнения:

- В зависимости от количества потоков
- В зависимости от размера задачи

Прокомментируйте полученные результаты в отчете

Дополнительное чтение:

- Structured Parallel Programming by Michael McCool, Arch Robison, James Reinders (http://parallelbook.com/)

Вопросы?