

# Введение в технологию OpenMP

## Семинар по использованию вычислительного кластера

4 апреля 2010 г.

# Классификация Флинна

## Классификация параллельных архитектур (М. Флинн)

**SIMD** (*Single Instruction, Multiple Data streams*) — один поток команд, множество потоков данных.

**MIMD** (*Multiple Instruction, Multiple Data streams*) — множество потоков команд, множество потоков данных.

# Классификация по типу используемой памяти

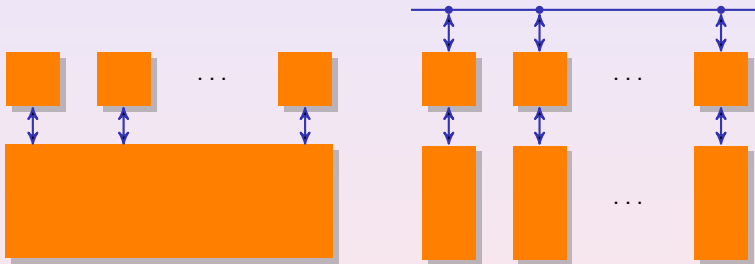


Рис. 1: Архитектуры с общей и распределённой памятью

# Многопоточный параллелизм

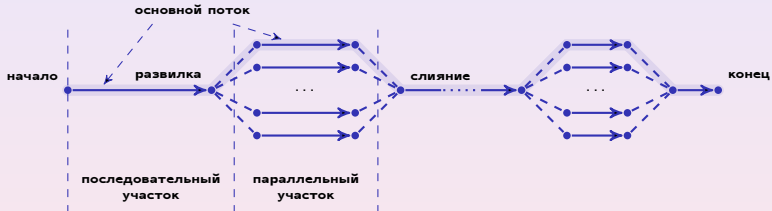


Рис. 2: Концепция многопоточного параллелизма

# Закон Амдала

## Оценка ускорения (G. Amdahl)

$$a \leq \frac{1}{(1 - p) + \frac{p}{n}}$$

- $a$  — ускорение программы;
- $p$  — распараллеливаемая часть программы (доля общего времени выполнения);
- $n$  — количество используемых процессоров.

## Асимптотика при росте количества ядер

$$a \rightarrow \frac{1}{1 - p} \quad \text{при } n \rightarrow +\infty$$

# Закон Амдала

## Оценка ускорения (G. Amdahl)

$$a \leq \frac{1}{(1 - p) + \frac{p}{n}}$$

- $a$  — ускорение программы;
- $p$  — распараллеливаемая часть программы (доля общего времени выполнения);
- $n$  — количество используемых процессоров.

## Асимптотика при росте количества ядер

$$a \rightarrow \frac{1}{1 - p} \quad \text{при } n \rightarrow +\infty$$

# Закон Амдала

## Оценка ускорения (G. Amdahl)

$$a \leq \frac{1}{(1 - p) + \frac{p}{n}}$$

- $a$  — ускорение программы;
- $p$  — распараллеливаемая часть программы (доля общего времени выполнения);
- $n$  — количество используемых процессоров.

## Асимптотика при росте количества ядер

$$a \rightarrow \frac{1}{1 - p} \quad \text{при } n \rightarrow +\infty$$

# Закон Амдала

## Оценка ускорения (G. Amdahl)

$$a \leq \frac{1}{(1 - p) + \frac{p}{n}}$$

- $a$  — ускорение программы;
- $p$  — распараллеливаемая часть программы (доля общего времени выполнения);
- $n$  — количество используемых процессоров.

## Асимптотика при росте количества ядер

$$a \rightarrow \frac{1}{1 - p} \quad \text{при } n \rightarrow +\infty$$



# Закон Амдала

## Оценка ускорения (G. Amdahl)

$$a \leq \frac{1}{(1 - p) + \frac{p}{n}}$$

- $a$  — ускорение программы;
- $p$  — распараллеливаемая часть программы (доля общего времени выполнения);
- $n$  — количество используемых процессоров.

## Асимптотика при росте количества ядер

$$a \rightarrow \frac{1}{1 - p} \quad \text{при } n \rightarrow +\infty$$

## Закон Амдала (окончание)

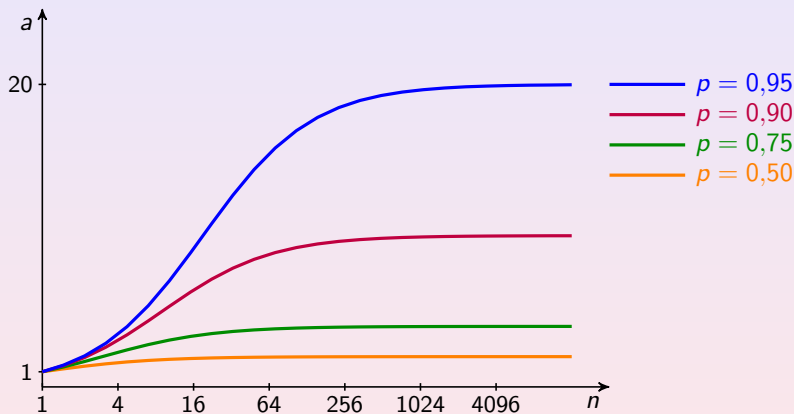


Рис. 3: Примеры графиков

# Организация многопоточности вручную

## Пример (использование библиотеки POSIX Threads)

```
int main()
{
    int i;
    pthread_t ahThreads[MY_NUM_THREADS];
    for (i = 0; i < MY_NUM_THREADS; ++ i)
        pthread_create(
            &ahThreads[i], NULL, g_afnThreadProcs[i], NULL);
    // ...
    for (i = 0; i < MY_NUM_THREADS; ++ i)
        pthread_join(ahThreads[i], NULL);
    // ...
}
```

# Назначение OpenMP

## Определение

**OpenMP** (*Open Multi-Processing*) — программный интерфейс для создания многопоточных приложений на многопроцессорных системах с разделяемой (общей) памятью.

## Компоненты

- директивы компилятора — расширения языков C/C++ и Fortran;
- библиотечные функции;
- переменные окружения.

# Краткая история OpenMP

Год	Событие
1998	OpenMP 1.0 (Fortran)
1999	OpenMP 1.0 (C/C++)
2000	OpenMP 2.0 (Fortran)
2002	OpenMP 2.0 (C/C++)
2005	OpenMP 2.5 (C/C++/Fortran)
2008	OpenMP 3.0 (C/C++/Fortran)

Таблица 1: Основные этапы развития OpenMP

# Настройка среды Microsoft Visual Studio 2008

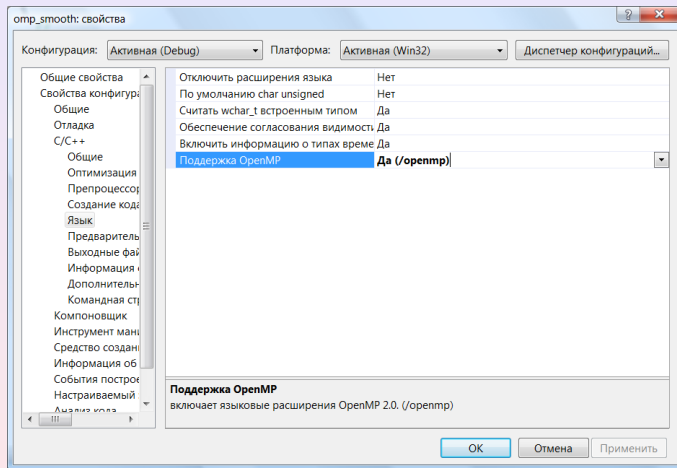


Рис. 4: Настройки компилятора в среде Microsoft Visual Studio 2008

# Настройка среды Code::Blocks

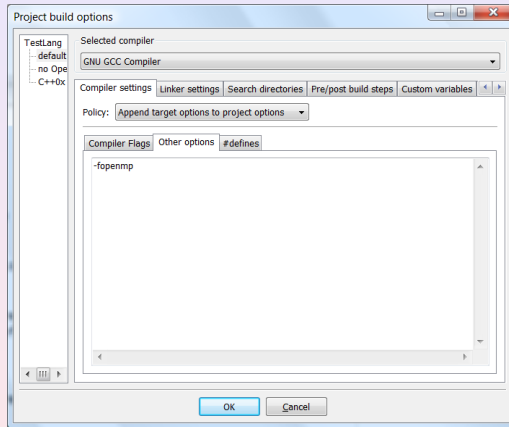


Рис. 5: Настройки компилятора в среде Code::Blocks

# Настройка среды Code::Blocks (окончание)

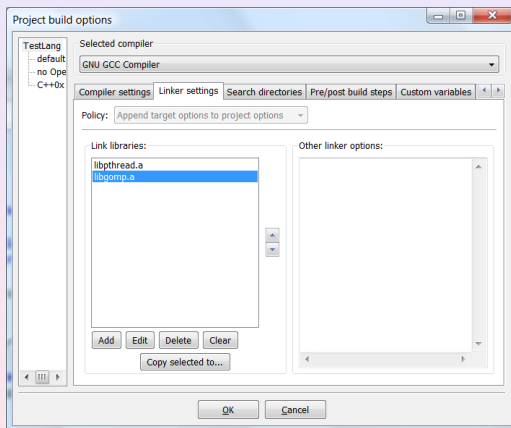


Рис. 6: Настройки редактора связей в среде Code::Blocks

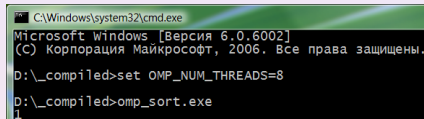


# Использование командной строки

Пример (включение OpenMP в командной строке)

```
stu003@tplatform1:~/omp_for> gfortran -fopenmp -lgomp omp_for.f90
```

# Настройка переменных окружения



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Версия 6.0.6002]
(C) Корпорация Майкрософт, 2006. Все права защищены.

D:\_compiled>set OMP_NUM_THREADS=8

D:\_compiled>omp_sort.exe
1
```

Рис. 7: Настройка переменной окружения OMP\_NUM\_THREADS

# Директивы OpenMP

## Определения

**Исполняемая директива OpenMP** — директива OpenMP, имеющая связанный с ней исполняемый пользовательский код.

**Декларативная директива OpenMP** — директива OpenMP, имеющая связанные с ней пользовательские объявления, но не исполняемый код (директива `threadprivate`).

# Директивы OpenMP

## Определения

**Исполняемая директива OpenMP** — директива OpenMP, имеющая связанный с ней исполняемый пользовательский код.

**Декларативная директива OpenMP** — директива OpenMP, имеющая связанные с ней пользовательские объявления, но не исполняемый код (директива `threadprivate`).

# Структурные блоки и конструкции OpenMP

## Определения

**Структурный блок программы** (*structured block*) — исполняемый оператор (возможно, составной) с единственной точкой входа в начале и единственной точкой выхода в конце, либо **конструкция OpenMP**.

**Конструкция OpenMP** (*construct*) — исполняемая директива OpenMP вместе со связанным структурным блоком (если он есть), не включая кода вызываемых подпрограмм.

# Структурные блоки и конструкции OpenMP

## Определения

**Структурный блок программы** (*structured block*) — исполняемый оператор (возможно, составной) с единственной точкой входа в начале и единственной точкой выхода в конце, либо **конструкция OpenMP**.

**Конструкция OpenMP** (*construct*) — **исполняемая директива OpenMP** вместе со связанным структурным блоком (если он есть), не включая кода вызываемых подпрограмм.

# Проверка версии OpenMP

## Пример (получение версии OpenMP)

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

int main()
{
#ifdef _OPENMP
    std::cout
        << "OpenMP Version: "
        << _OPENMP / 100 << " (" << _OPENMP % 100 << ")" << std::endl;
#else
    std::cout << "Sequential Version" << std::endl;
#endif
}
```

# Формат директивы OpenMP

## Формат директивы (C/C++)

**#pragma omp** *имя* [*предложение* {[*предложение*]}] *↵*

- |                    |   |   |
|--------------------|---|---|
| <i>имя</i>         | — | имя директивы;  |
| <i>предложение</i> | — | конструкция, задающая дополнительную информацию и зависящая от директивы; |
| <i>↵</i>           | — | конец строки (в дальнейшем не указывается).                               |



# Формат директивы OpenMP

## Формат директивы (C/C++)

**#pragma omp** *имя* [*предложение* {[*,*] *предложение*}] *↵*

- имя* — имя директивы;
- предложение* — конструкция, задающая дополнительную информацию и зависящая от директивы;
- ↵* — конец строки (в дальнейшем не указывается).

## Формат директивы (Fortran)

*!\$OMP* *имя* [*предложение* {[*,*] *предложение*}] *↵*  
*C\$OMP* *имя* [*предложение* {[*,*] *предложение*}] *↵*  
*\*\$OMP* *имя* [*предложение* {[*,*] *предложение*}] *↵*

# Директива parallel

## Формат директивы (C/C++)

```
#pragma omp parallel [⟨предложения⟩]  
    ⟨структурный блок⟩
```

# Директива parallel

## Формат директивы (C/C++)

```
#pragma omp parallel [⟨предложения⟩]  
    ⟨структурный блок⟩
```

## Формат директивы (Fortran)

```
!$omp parallel [⟨предложения⟩]  
    ⟨структурный блок⟩  
!$omp end parallel
```

# Директива parallel (окончание)

## Действие директивы

- Поток, встречающий конструкцию parallel, создаёт команду потоков, становясь для неё основным.
- Потокам команды присваиваются уникальные целые номера, начиная с 0 (основной поток).
- Каждый поток исполняет код, определяемый структурным блоком, в конце которого неявно устанавливается барьер.

# Директива parallel (окончание)

## Действие директивы

- Поток, встречающий конструкцию parallel, создаёт команду потоков, становясь для неё основным.
- Потокам команды присваиваются уникальные целые номера, начиная с 0 (основной поток).
- Каждый поток исполняет код, определяемый структурным блоком, в конце которого неявно устанавливается барьер.

# Директива parallel (окончание)

## Действие директивы

- Поток, встречающий конструкцию parallel, создаёт команду потоков, становясь для неё основным.
- Потокам команды присваиваются уникальные целые номера, начиная с 0 (основной поток).
- Каждый поток исполняет код, определяемый структурным блоком, в конце которого неявно устанавливается **барьер**.

# Барьер

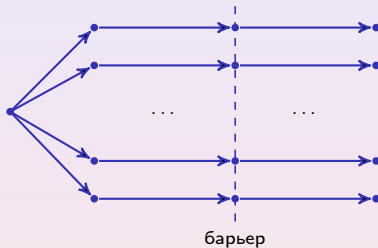


Рис. 8: Концепция барьера

# Пример использования директивы parallel

## Пример (C++)

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

int main()
{
    #pragma omp parallel
    std::cout << " Inside parallel block" << std::endl;
}
```



# Пример использования директивы parallel (окончание)

## Пример (Fortran)

```
program omp_first
  !$omp parallel
  print *, "Inside parallel block"
  !$omp end parallel
end
```

# Модель памяти OpenMP

## Виды переменных

- **Общие** (shared).
- **Локальные** (private).

# Область

## Определения

**Область** (*region*) — весь код, исполнявшийся при выполнении заданной конструкции или функции OpenMP (т. е., область включает все вызываемые подпрограммы, а также неявный код, добавленный реализацией OpenMP).

**Связанная область** (*binding region*) — для заданной области — охватывающая её область, которая определяет для неё контекст исполнения и определяет границы её воздействия.

# Область

## Определения

**Область** (*region*) — весь код, исполнявшийся при выполнении заданной конструкции или функции OpenMP (т. е., область включает все вызываемые подпрограммы, а также неявный код, добавленный реализацией OpenMP).

**Связанная область** (*binding region*) — для заданной области — охватывающая её область, которая определяет для неё контекст исполнения и определяет границы её воздействия.

## Функция `omp_get_num_threads()`

### Объявление (C/C++)

```
int omp_get_num_threads(void);
```

### Объявление (Fortran)

```
integer function omp_get_num_threads()
```

## Функция `omp_get_max_threads()`

### Объявление (C/C++)

```
int omp_get_max_threads(void);
```

### Объявление (Fortran)

```
integer function omp_get_max_threads()
```

## Функция `omp_get_thread_num()`

### Объявление (C/C++)

```
int omp_get_thread_num(void);
```

### Объявление (Fortran)

```
integer function omp_get_thread_num()
```

# Поиск максимального значения в массиве

## Пример

В заданном массиве вещественных чисел найти максимальное.



Рис. 9: Параллелизм по данным



## Поиск максимального значения в массиве (продолжение)

### Пример (C++)

```
Vector::value_type max(const Vector &rcV)
{
    if (rcV.empty())
        return 0;
    //
    Vector v_max(omp_get_max_threads(), rcV[0]);
```

# Поиск максимального значения в массиве (продолжение)

## Пример (C++, продолжение)

```
#pragma omp parallel
{
    int nSize = omp_get_num_threads();
    int nRank = omp_get_thread_num();
    size_t uChunkSize = rcV.size() / nSize;
    size_t uStart = nRank * uChunkSize;
    size_t uEnd =
        (nRank == nSize - 1 ?
         rcV.size() : (nRank + 1) * uChunkSize);
    v_max[nRank] = *max_element(
        rcV.begin() + uStart, rcV.begin() + uEnd);
}
```

## Поиск максимального значения в массиве (окончание)

### Пример (C++, окончание)

```
//  
return *max_element(v_max.begin(), v_max.end());  
}
```

## Список литературы



Савельев В. А.

*Методические указания на тему: «Параллельное программирование: OpenMP API».*

Изд-во РГУ, Ростов-на-Дону, 2006.

Available from: <http://open-edu.sfedu.ru/pub/1931/>.



Антонов А. С.

*Параллельное программирование с использованием технологии OpenMP: Учебное пособие.*

Изд-во МГУ, М., 2009.

Available from: <http://parallel.ru/info/parallel/openmp/>.



*OpenMP Application Program Interface / Version 3.0 May 2008.*

OpenMP Architecture Review Board, 2008.

Available from: <http://openmp.org/wp/openmp-specifications/>.