HPC-2

Общая память: OpenMP

перпендикулярный взгляд

И.Н. Коньшин[1,2,3,4]

[1]Институт вычислительной математики им. Г.И. Марчука, РАН
[2]Московский физико-технический институт
[3]Сеченовский университет
[4]Университет Сириус

План

Ситуация:

- параллельные вычисления гораздо быстрее последовательных...
- // результатов много, а оценок // эффективности почти нет
- // моделей достаточно, оценки тоже встречаются,
  но либо слишком абстрактные, либо слишком подробные

Цель:

- конструктивные оценки //-ной эффективности для различных алгоритмов

План:

- классификация компьютеров
- общая память
- получение оценок //-ной эффективности
- примеры программ
- результатов численных экспериментов

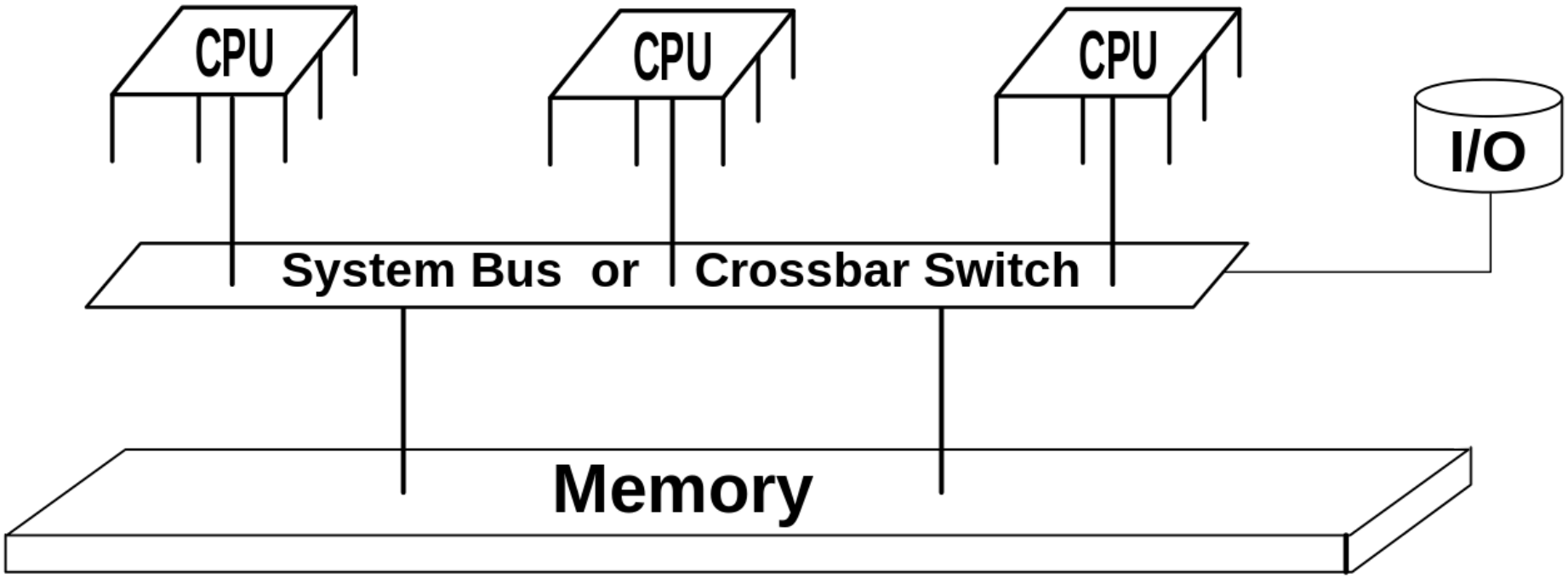
Классификация компьютеров

- ***Общая*** память (рабочие станции, персональные компьютеры, ноутбуки, телефоны, ...)
- ***Распределенная*** память (суперкомпьютеры, кластеры, ...)

- *'Разделяемая'* $\longleftrightarrow$ *'Разделенная'*

¡ Давайте начнем с ***'Общей'*** *памяти !*

Общая память



OpenMP $\longleftarrow$ общая память

OpenMP (Open Multi-Processing) для C, C++, Fortran

OpenMP состоит из набора:

- *макросов*,
- *директив препроцессора*,
- библиотечных функций и
- переменных окружения.

1997: v.1.0
2018: v.5.0

Оценка //-ной эффективности (общая память)

Закон Амдала [Amdahl's law, 1967]

- p — количество процессов (потоков)
- $T(p)$ — время выполнения программы для p процессов
- σ — доля последовательных (не распараллеленных) операций

Ускорение:

$$S(p) = \frac{T(1)}{T(p)} = \frac{T(1)}{\sigma T(1) + \frac{(1-\sigma)T(1)}{p}} = \frac{p}{1+\sigma(p-1)}$$

¡ *Пример:* $S(p=1) = 1$, $S(\sigma=0) = p$, $S(\sigma=0.01, p\to\infty) = 100$ *!*

// (параллельная) эффективность:

$$E(p) = \frac{S(p)}{p} = \frac{1}{1+\sigma(p-1)}$$

¡ *Напрямую применимо к программам на OpenMP !*

Условия наилучшей применимости закона Амдала

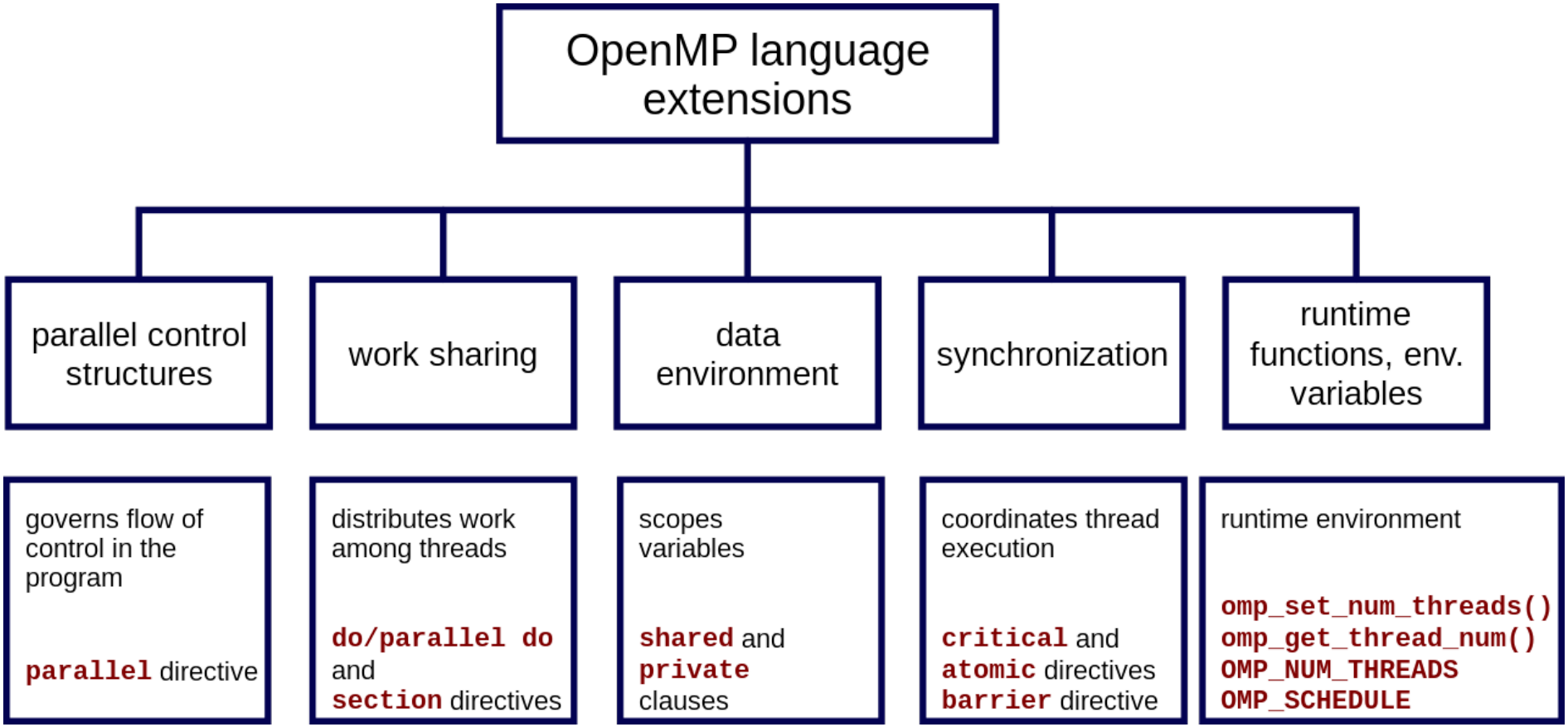
- арифметические операции достаточно ***однородны***;
- ***все*** доступные нити участвуют в вычислениях параллельной части кода;
- ***сбалансированность*** вычислений для всех нитей;
- ***масштабируемость*** работы нитей, т.е. скорость работы нитей не зависит от количества нитей (или, другими словами, время выполнения параллельной части действительно уменьшается в p раз при использовании p нитей).

Традиционный взгляд на OpenMP:

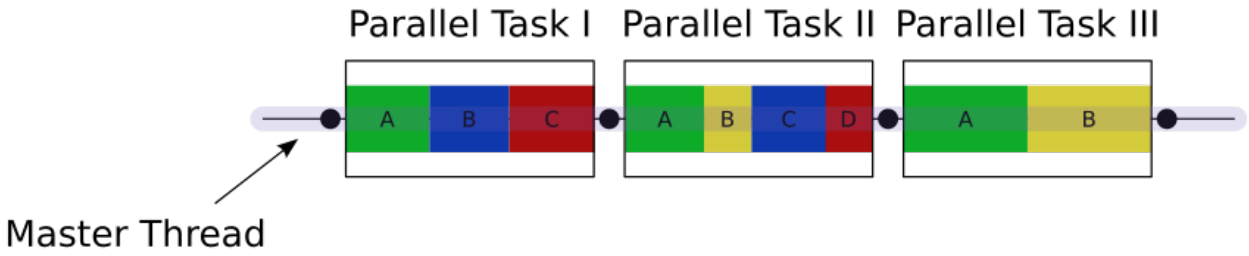
справочники, примеры применения:

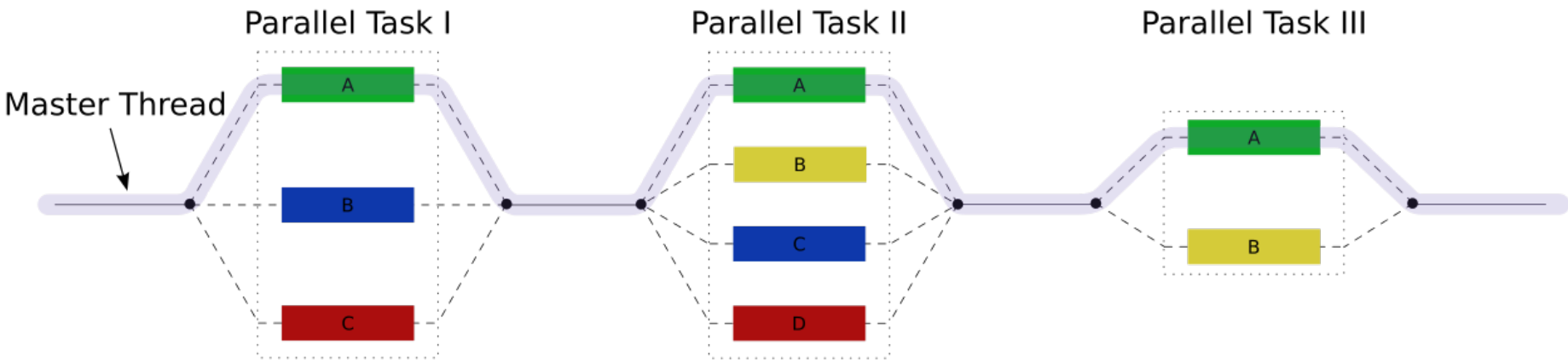
- The OpenMP API specification for parallel programming. https://www.openmp.org
- А.С.Антонов, Параллельное программирование с использованием технологии OpenMP, МГУ, Москва, 2009 http://dodo.inm.ras.ru/konshin/HPC/bib/OpenMP-Antonov.pdf

OpenMP: расширение языка



OpenMP: нити (потоки)





OpenMP:

Мы заново изобретем его с точки зрения пользователя и разработчика

OpenMP: по возможности легкое и незаметное распараллеливание

- макросы
- директивы препроцессора
- *функции*

OpenMP: поддерживается ли?

(Q?) Поддерживается ли OpenMP данным компилятором?
Можно ли использовать средства OpenMP?

(A!) Макрос **_OPENMP** определён в формате ***yyyymm***, где ***yyyy*** и ***mm*** — цифры года и месяца, когда был принят поддерживаемый стандарт OpenMP.

Условная компиляция:

```
#include <stdio.h>
int main() {
#ifdef _OPENMP
   printf("OpenMP is supported!\n");
#endif
}
```

- Задание: *Напечатать значение года и месяца для своего компилятора*

OpenMP: поддерживается ли?

(Q?) Поддерживается ли OpenMP данным компилятором?
Можно ли использовать средства OpenMP?

(A!) Макрос **_OPENMP** определён в формате ***yyyymm***, где ***yyyy*** и ***mm*** — цифры года и месяца, когда был принят поддерживаемый стандарт OpenMP.

Условная компиляция:

```
#include <stdio.h>
int main() {
#ifdef _OPENMP
   printf("OpenMP is supported!\n");
#endif
}
```

- Задание: *Напечатать значение года и месяца для своего компилятора*

OpenMP: поддерживается ли?

(Q?) Поддерживается ли OpenMP данным компилятором?
Можно ли использовать средства OpenMP?

(A!) Макрос **_OPENMP** определён в формате ***yyyymm***, где ***yyyy*** и ***mm*** — цифры года и месяца, когда был принят поддерживаемый стандарт OpenMP.

Условная компиляция:

```
#include <stdio.h>
int main() {
#ifdef _OPENMP
   printf("OpenMP is supported!\n");
#endif
}
```

- **Задание:** *Напечатать значение года и месяца для своего компилятора*

OpenMP: директивы

(Q?) Можем ли мы задействовать препроцессор для OpenMP?

(A!) Директивы OpenMP в языке Си задаются указаниями препроцессору:

#pragma omp directive-name [опция[, опция]...]

Объектом действия директивы является один оператор или блок, перед которым стоит директива в коде программы.

OpenMP: директивы

(Q?) Можем ли мы задействовать препроцессор для OpenMP?

(A!) Директивы OpenMP в языке Си задаются указаниями препроцессору:

#pragma omp directive-name [опция[, опция]...]

Объектом действия директивы является один оператор или блок, перед которым стоит директива в коде программы.

OpenMP: основная и любимая директива

(Q?) Можем ли мы распараллелить циклы?

(A!) Прямо перед циклом в Си можно написать:

#pragma omp parallel for [опция[, опция]...]

В большинстве случаев опции даже не потребуются:

```
#pragma omp parallel for
for {int i=0; i<N; i++} {
    ...
}
```

(A!) **export OMP_NUM_THREADS=n**
— задание количества нитей

¡ Пока еще не требуется никаких библиотек и **#include <...>** *!*

OpenMP: основная и любимая директива

(Q?) Можем ли мы распараллелить циклы?

(A!) Прямо перед циклом в Си можно написать:

#pragma omp parallel for [опция[, опция]...]

В большинстве случаев опции даже не потребуются:

```
#pragma omp parallel for
for {int i=0; i<N; i++} {
    ...
}
```

(A!) export OMP_NUM_THREADS=n
— задание количества нитей

¡ Пока еще не требуется никаких библиотек и **#include <...>** *!*

OpenMP: основная и любимая директива

(Q?) Можем ли мы распараллелить циклы?

(A!) Прямо перед циклом в Си можно написать:

#pragma omp parallel for [опция[, опция]...]

В большинстве случаев опции даже не потребуются:

```
#pragma omp parallel for
for {int i=0; i<N; i++} {
    ...
}
```

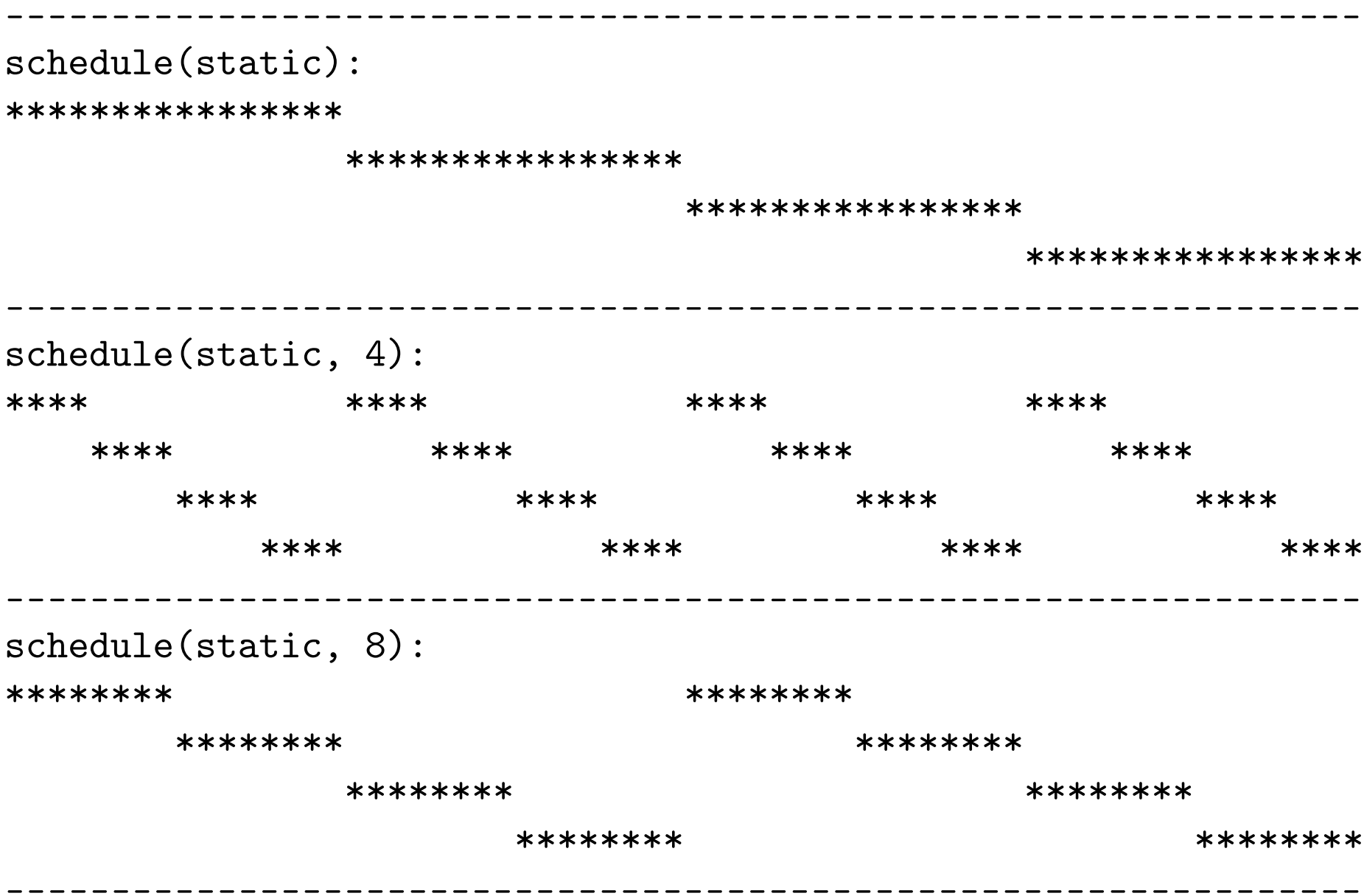
(A!) **export OMP_NUM_THREADS=n**
— задание количества нитей

¡ Пока еще не требуется никаких библиотек и **#include <...>** *!*

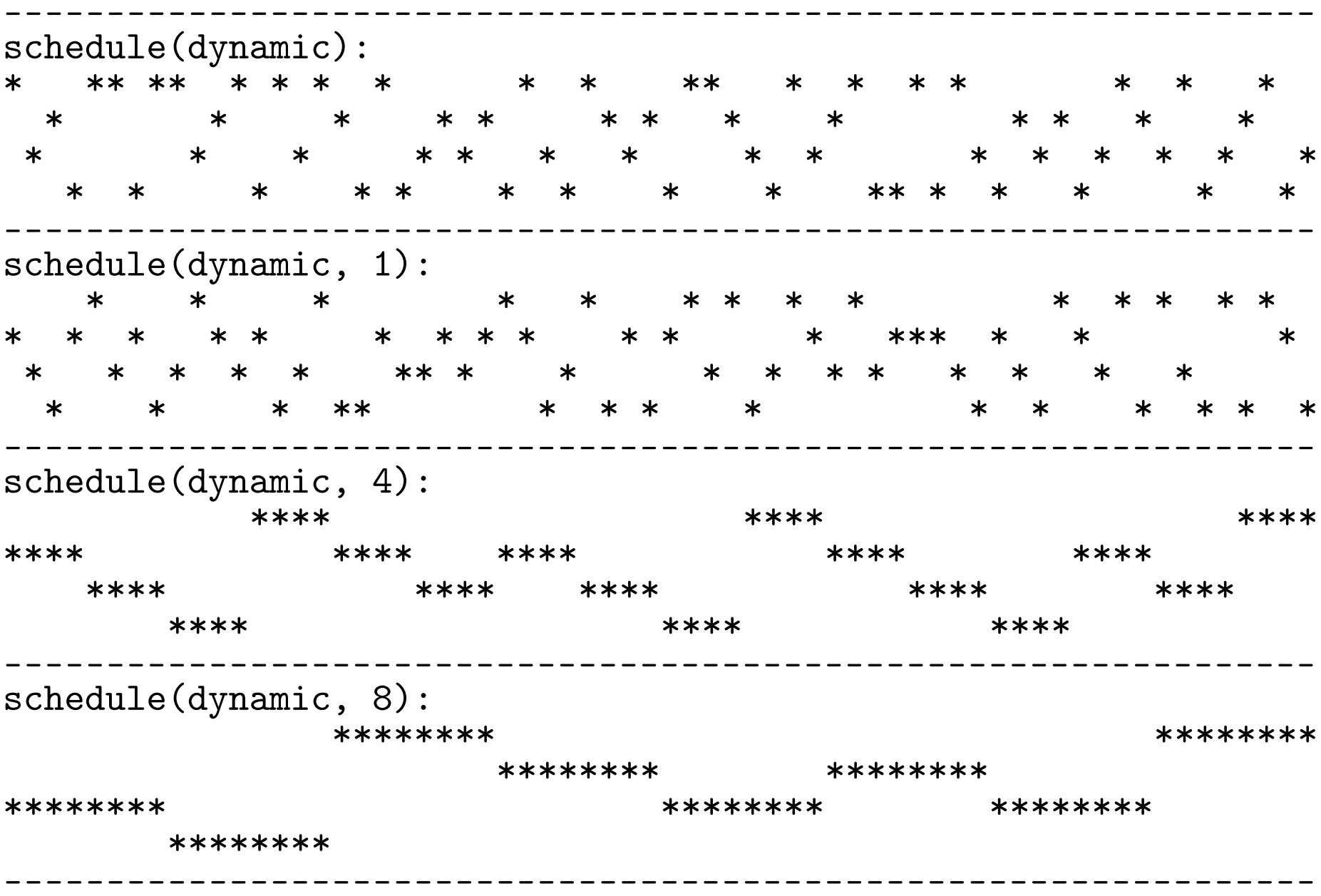
OpenMP: опции директивы **parallel for**

- **if(*условие*)** — выполнение параллельной области по условию
- **num_threads (*целочисленное выражение*)** — явное задание количества нитей
- **private(*список*)** — список локальных переменных для каждой нити
- **shared(*список*)** — список общих переменных для всех нитей
- **reduction(*оператор:список*)** — задаёт оператор (+,*,&&,||,min,max,...) и список общих переменных
- **schedule(*type[, chunk]*)** — задаёт, каким образом итерации цикла распределяются между нитями
- **collapse(*n*)** — директива действует на ***n*** последовательных тесновложенных циклов

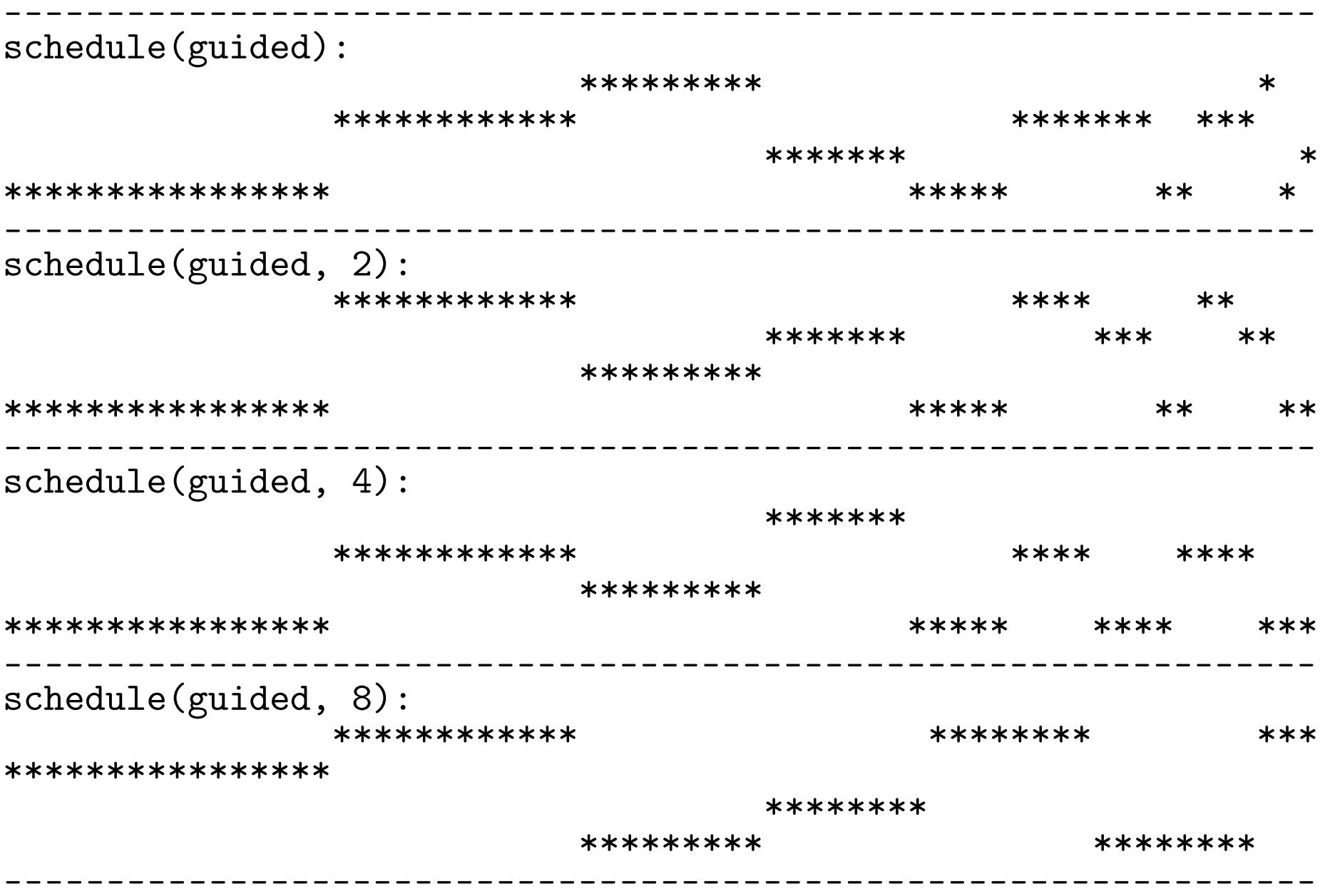
OpenMP: schedule(**static**,chunk)



OpenMP: schedule(**dynamic**,chunk)



OpenMP: schedule(**guided**,chunk)



OpenMP: schedule(**auto** or **runtime**) or **default**

```
-----------------------------------------------------------
schedule(auto):
*****************
                  *****************
                                    *****************
                                                      *****************
-----------------------------------------------------------
runtime:
export OMP_SCHEDULE=...

-----------------------------------------------------------
default:
#pragma omp parallel for
for (...) { ... }
*****************
                  *****************
                                    *****************
                                                      *****************
-----------------------------------------------------------
```

OpenMP: тестирование расписаний на h1.inm.ras.ru

```
OpenMP version _OPENMP = 201611 is supported!
daxpy: N=1000000 M=100 th=32
#pragma omp parallel for schedule(sch,chunk)

sch=static    chunk=1     time=0.469  Mflops=213.4
sch=static    chunk=2     time=0.242  Mflops=413.7
sch=static    chunk=4     time=0.175  Mflops=571.3
sch=static    chunk=10    time=0.128  Mflops=783.5
sch=static    chunk=100   time=0.071  Mflops=1412.1
sch=static    chunk=1000  time=0.010  Mflops=9721.2 **

sch=dynamic   chunk=1     time=6.221  Mflops=16.1
sch=dynamic   chunk=2     time=2.925  Mflops=34.2
sch=dynamic   chunk=4     time=1.416  Mflops=70.6
sch=dynamic   chunk=10    time=0.589  Mflops=169.7
sch=dynamic   chunk=100   time=0.049  Mflops=2042.6
sch=dynamic   chunk=1000  time=0.011  Mflops=9372.1 *

sch=guided    chunk=1     time=0.017  Mflops=5811.6
sch=guided    chunk=2     time=0.016  Mflops=6168.3
sch=guided    chunk=4     time=0.015  Mflops=6570.2
sch=guided    chunk=10    time=0.014  Mflops=6988.6
sch=guided    chunk=100   time=0.012  Mflops=8152.7
sch=guided    chunk=1000  time=0.011  Mflops=8844.8 *

sch=auto                  time=0.013  Mflops=7968.1 *
```

OpenMP: #pragma omp parallel for **collapse**

```
OpenMP version _OPENMP = 201611 is supported!
daxpy: for(10) for(10) for(1000000) { ... } th=8
#pragma omp parallel for collapse

collapse(1) static(1): time=0.092 Mflops=1085.5
collapse(2) static(1): time=0.051 Mflops=1953.8
collapse(3) static(1): time=0.183 Mflops=543.7

collapse(1)            time=0.089 Mflops=1118.9
collapse(2)            time=0.046 Mflops=2143.9
collapse(3)            time=0.064 Mflops=1550.7
```

OpenMP: вместо #pragma omp parallel for

```
#ifdef _OPENMP                          // Old usage
#pragma omp parallel for                // of OpenMP
#endif                                  // directive
    for (i=0; i<n; i++) {...}           // with verification :(

    FOR (i=0; i<n; i++) {...}           // New one :)

#ifdef _OPENMP
    #include <omp.h>
    #ifdef _WIN32 // MS Windows
        #define OMP_PARALLEL_FOR __pragma(omp parallel for)
    #else // Linux
        #define OMP_PARALLEL_FOR _Pragma("omp parallel for")
    #endif
#else // OpenMP is not supported
    #define OMP_PARALLEL_FOR
#endif

#define FOR(iii) OMP_PARALLEL_FOR for(iii)
```

OpenMP: другие директивы

- **#pragma omp barrier**
- **#pragma omp single**
- **#pragma omp master**
- **#pragma omp atomic**

OpenMP: функции

- *Без них всё-таки не обойтись...*

- **#include <omp.h>** — описания функций
- **double omp_get_wtime(void);** — таймер в сек.
- **double omp_get_wtick(void);** — разрешение таймера в сек.
- **int omp_get_max_threads(void);** — возвращает максимально допустимое число нитей для использования в следующей параллельной области
- **int omp_get_thread_num(void);** — уникальный номер нити в текущей параллельной области
- **void omp_set_num_threads(int num);** — установить максимально допустимое число нитей
- **void omp_..._lock(...);** — логика замков для доступа к общим данным

OpenMP: функции

- *Без них всё-таки не обойтись...*

- **#include <omp.h>** — описания функций
- **double omp_get_wtime(void);** — таймер в сек.
- **double omp_get_wtick(void);** — разрешение таймера в сек.
- **int omp_get_max_threads(void);** — возвращает максимально допустимое число нитей для использования в следующей параллельной области
- **int omp_get_thread_num(void);** — уникальный номер нити в текущей параллельной области
- **void omp_set_num_threads(int num);** — установить максимально допустимое число нитей
- **void omp_..._lock(...);** — логика замков для доступа к общим данным

OpenMP: функции

- *Без них всё-таки не обойтись...*

- **#include <omp.h>** — описания функций
- **double omp_get_wtime(void);** — таймер в сек.
- **double omp_get_wtick(void);** — разрешение таймера в сек.
- **int omp_get_max_threads(void);** — возвращает максимально допустимое число нитей для использования в следующей параллельной области
- **int omp_get_thread_num(void);** — уникальный номер нити в текущей параллельной области
- **void omp_set_num_threads(int num);** — установить максимально допустимое число нитей
- **void omp_..._lock(...);** — логика замков для доступа к общим данным

OpenMP: функции

- *Без них всё-таки не обойтись...*

- **#include <omp.h>** — описания функций
- **double omp_get_wtime(void);** — таймер в сек.
- **double omp_get_wtick(void);** — разрешение таймера в сек.
- **int omp_get_max_threads(void);** — возвращает максимально допустимое число нитей для использования в следующей параллельной области
- **int omp_get_thread_num(void);** — уникальный номер нити в текущей параллельной области
- **void omp_set_num_threads(int num);** — установить максимально допустимое число нитей
- **void omp_..._lock(...);** — логика замков для доступа к общим данным

OpenMP: функции

- *Без них всё-таки не обойтись...*

- **#include <omp.h>** — описания функций
- **double omp_get_wtime(void);** — таймер в сек.
- **double omp_get_wtick(void);** — разрешение таймера в сек.
- **int omp_get_max_threads(void);** — возвращает максимально допустимое число нитей для использования в следующей параллельной области
- **int omp_get_thread_num(void);** — уникальный номер нити в текущей параллельной области
- **void omp_set_num_threads(int num);** — установить максимально допустимое число нитей
- **void omp_..._lock(...);** — логика замков для доступа к общим данным

OpenMP: функции

- *Без них всё-таки не обойтись...*

- **#include <omp.h>** — описания функций
- **double omp_get_wtime(void);** — таймер в сек.
- **double omp_get_wtick(void);** — разрешение таймера в сек.
- **int omp_get_max_threads(void);** — возвращает максимально допустимое число нитей для использования в следующей параллельной области
- **int omp_get_thread_num(void);** — уникальный номер нити в текущей параллельной области
- **void omp_set_num_threads(int num);** — установить максимально допустимое число нитей
- **void omp_..._lock(...);** — логика замков для доступа к общим данным

OpenMP: функции

- *Без них всё-таки не обойтись...*

- **#include <omp.h>** — описания функций
- **double omp_get_wtime(void);** — таймер в сек.
- **double omp_get_wtick(void);** — разрешение таймера в сек.
- **int omp_get_max_threads(void);** — возвращает максимально допустимое число нитей для использования в следующей параллельной области
- **int omp_get_thread_num(void);** — уникальный номер нити в текущей параллельной области
- **void omp_set_num_threads(int num);** — установить максимально допустимое число нитей
- **void omp_..._lock(...);** — логика замков для доступа к общим данным

OpenMP: функции

- *Без них всё-таки не обойтись...*

- **#include <omp.h>** — описания функций
- **double omp_get_wtime(void);** — таймер в сек.
- **double omp_get_wtick(void);** — разрешение таймера в сек.
- **int omp_get_max_threads(void);** — возвращает максимально допустимое число нитей для использования в следующей параллельной области
- **int omp_get_thread_num(void);** — уникальный номер нити в текущей параллельной области
- **void omp_set_num_threads(int num);** — установить максимально допустимое число нитей
- **void omp_..._lock(...);** — логика замков для доступа к общим данным

OpenMP: serial → parallel → serial

1. Сначала была последовательная программа...
2. Потом она стала параллельной (были вставлены директивы и функции OpenMP)
3. Что делать, если она снова попала на компьютер, где нет поддержки OpenMP?

- Приличные компиляторы поставляют специальные библиотеки-заглушки (*stub*), в которых эмуляторы функций OpenMP возвращают дефолты для одной нити
- В принципе, такую заглушку за полчаса можно было бы написать и самому...

OpenMP: serial $\rightarrow$ parallel $\rightarrow$ serial

1. Сначала была последовательная программа...
2. Потом она стала параллельной (были вставлены директивы и функции OpenMP)
3. Что делать, если она снова попала на компьютер, где нет поддержки OpenMP?

- Приличные компиляторы поставляют специальные библиотеки-заглушки (*stub*), в которых эмуляторы функций OpenMP возвращают дефолты для одной нити
- В принципе, такую заглушку за полчаса можно было бы написать и самому...

OpenMP: практические задания

(a.c) Распечатать значение макроса `_OPENMP` и узнать

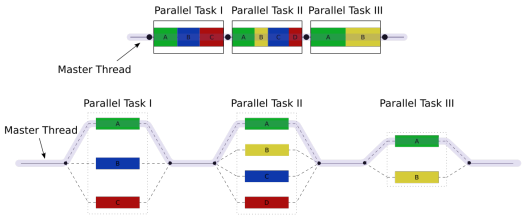
- номер поддерживаемой версии в формате `yyyymm`.

(b.c) Распечатать возвращаемое значение функции `omp_get_num_threads()`:

- в последовательной
- и параллельной области.

(c.c) В одной программе тремя способами задать количество нитей:

- `export OMP_NUM_THREADS=3`
- `#pragma omp parallel for num_threads(4)`
- `omp_set_num_threads(2)`



(d.c) Реализовать матричное умножение `dgemm` (как в BLAS-3) и проверить производительность программы в зависимости от:

- порядка циклов (6 различных вариантов);
- размерности задачи, например, 1000 и 4000;
- `schedule`;
- `chunk`.

OpenMP: практические задания v.2

1. Распечатать значение макроса `_OPENMP` и узнать
   - номер поддерживаемой версии в формате `yyyymm`.
2. Чему равно время:
   - выполнения функции `omp_get_wtime()`;
   - минимального кванта `omp_get_wtick()`.
3. В одной программе тремя способами задать количество нитей:
   - `export OMP_NUM_THREADS=3`
   - `#pragma omp parallel for num_threads(4)`
   - `omp_set_num_threads(2)`
4. Реализовать любую операцию в цикле для достижения максимальной производительности в MFLOPS, при этом подобрать:
   - количество задействованных потоков OpenMP;
   - размеры циклов;
   - `schedule`;
   - `chunk`.

Примеры

- daxpy: $\alpha \cdot \vec{x} + \vec{y}$
- norm: $\|\vec{x}\|$
- mvm: $y = A \cdot x$

Коды программ и результаты // запусков

Пока общая (*а потом и распределенная*) память

Кластер ИВМ РАН http://cluster2.inm.ras.ru/

Вычислительные сегменты кластера:

- x6core
- x8core
- x10core
- x12core

Сегмент **x6core:**

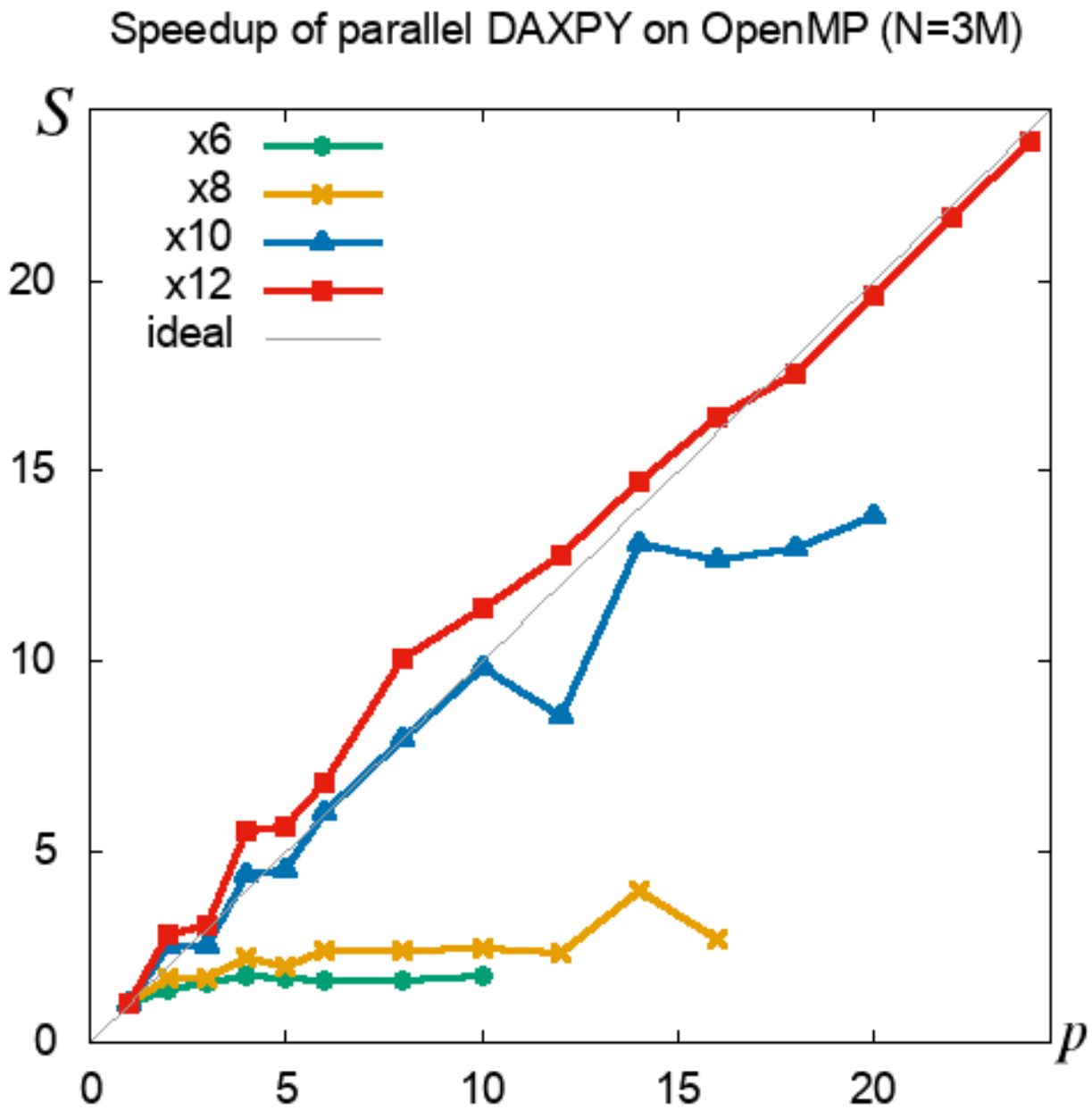
- Compute Node Asus RS704D-E6;
- 12 ядер (два 6-ядерных процессора Intel Xeon X5650@2.67 ГГц);
- Оперативная память: 24 Гб.;
- Операционная система: SUSE Linux Enterprise Server 11 SP1 (x86_64);
- Коммутационная сеть: Mellanox Infiniband QDR 4x.

Для сборки кода использовался компилятор Intel C версии 4.0.1 с библиотекой Intel MPI версии 5.0.3.

OMP daxpy: $\alpha \cdot \vec{x} + \vec{y}$

```
#include <omp.h>

int main()
{
    t = omp_get_wtime();
#pragma omp parallel for
    for (i=0; i<N; i++) y[i] += a * x[i];
    t = omp_get_wtime() - t;

    printf("OMP daxpy: N=%d max_th=%d time=%lf\n",
           N, omp_get_max_threads(), t);
}

$ gcc -fopenmp omp_daxpy.c -O2
$ export OMP_NUM_THREADS=4 ; ./a.out
OMP daxpy: N=1000000 max_th=4 time=0.01
```

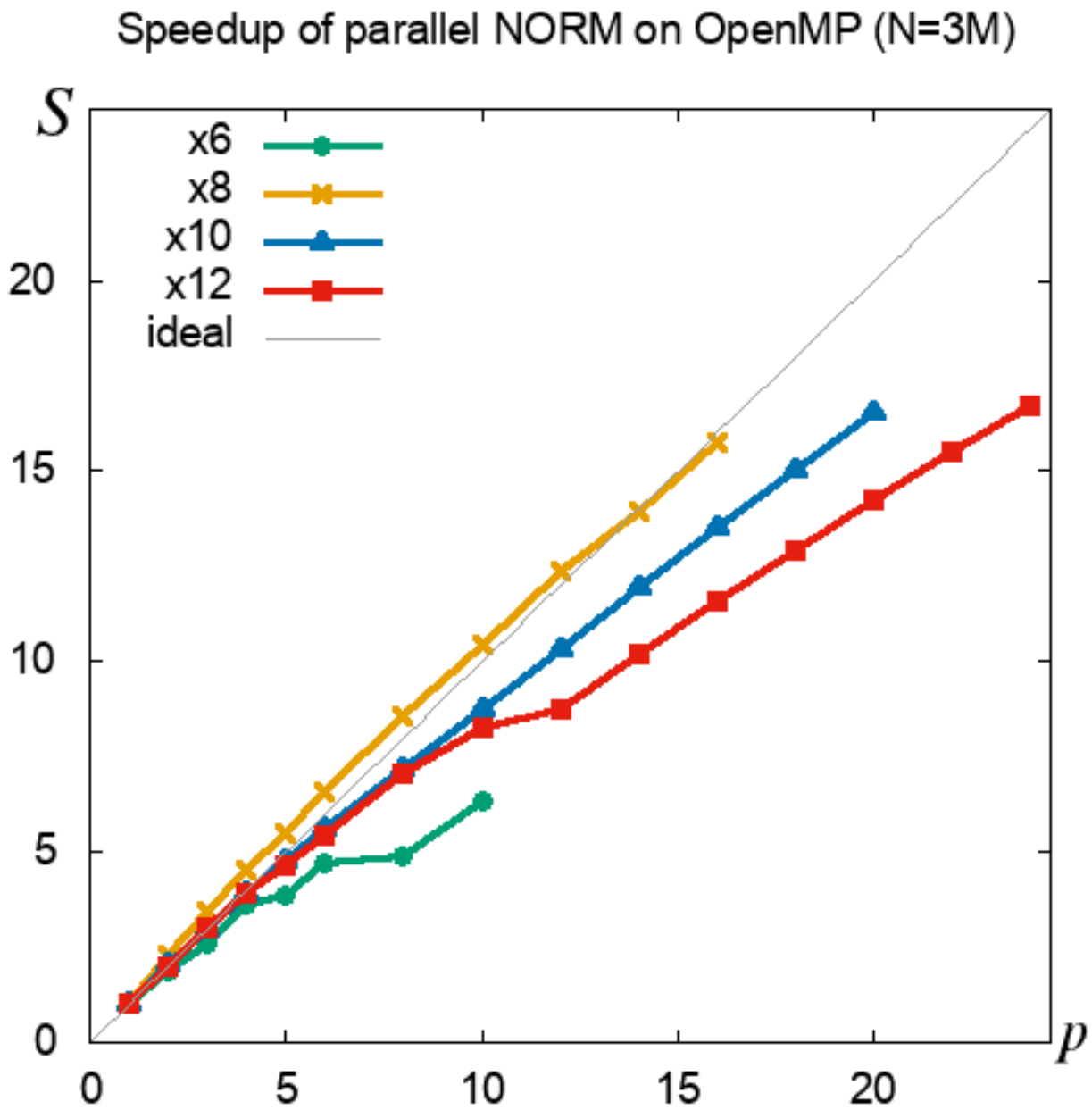
OMP daxpy: $\alpha \cdot \vec{x} + \vec{y}$



OMP norm: $\|\vec{x}\|$

```
#include <omp.h>

#define N 1000000

int main()
{
    t = omp_get_wtime();
    sum = 0.0;
#pragma omp parallel for reduction (+:sum)
    for (i=0; i<N; i++)
        sum += x[i] * x[i];
    sum = sqrt(sum);
    t = omp_get_wtime() - t;

    printf("OMP norm: N=%d max_th=%d time=%lf\n",
           N, omp_get_max_threads(), t);
}

$ gcc -fopenmp omp_norm.c -O2 -lm
$ export OMP_NUM_THREADS=4 ; ./a.out
OMP norm: N=1000000 max_th=4 time=0.01
```

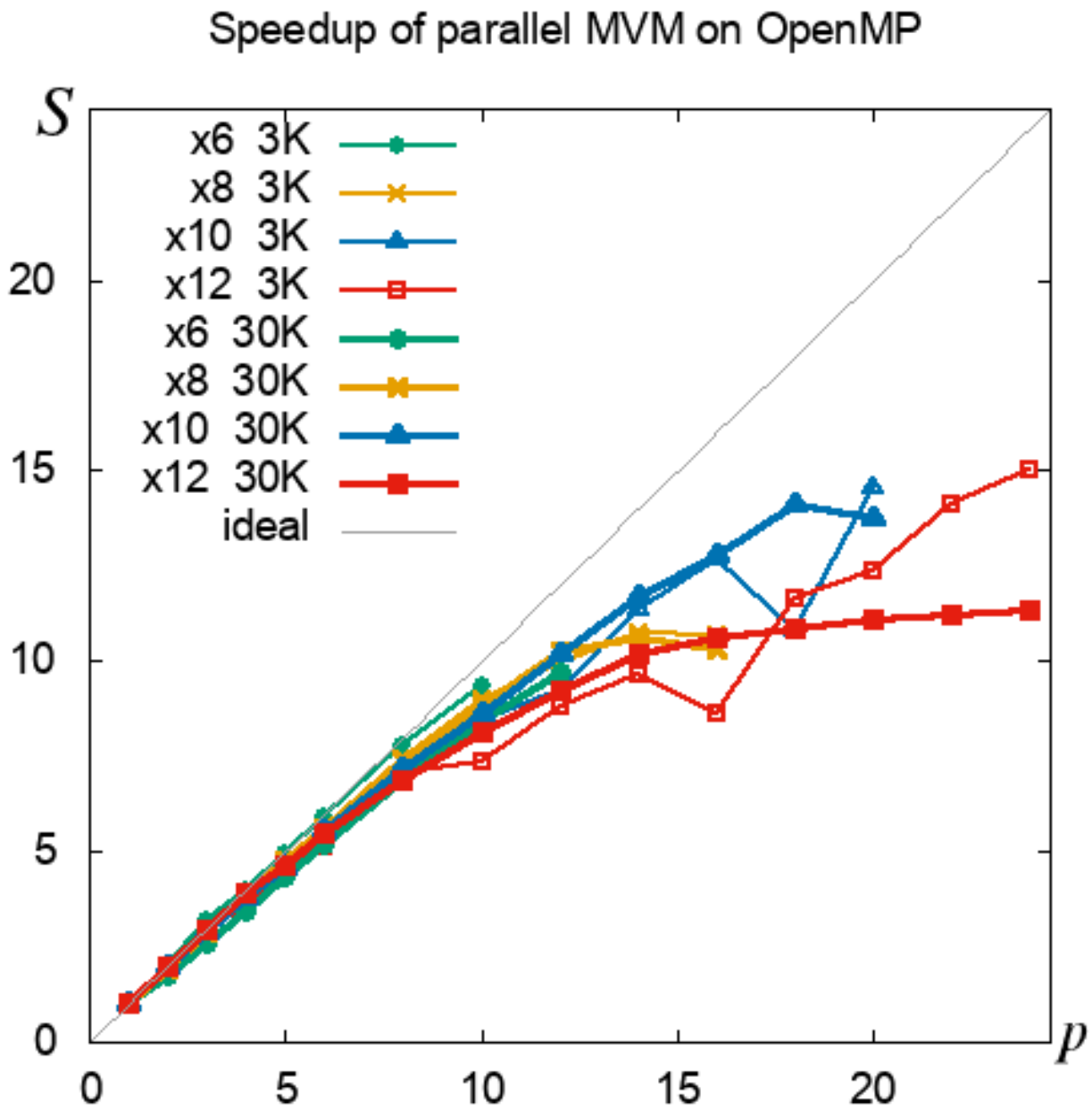
OMP norm: $\|\vec{x}\|$



OMP mvm: $y = A \cdot x$

```
#include <omp.h>

#define N 1000

int main()
{
    double a[N][N], x[N], y[N], t;

    t = omp_get_wtime();

... A code fragment from a practical work ...

    t = omp_get_wtime() - t;

    printf("OMP mvm: N=%d max_th=%d time=%lf\n",
           N, omp_get_max_threads(), t);
}

$ gcc -fopenmp omp_mvm.c -O2
$ export OMP_NUM_THREADS=4 ; ./a.out
OMP mvm: N=4000 max_th=4 time=0.1
```

OMP mvm: $y = A \cdot x$



OpenMP: парадигма

Предположим, что несколько строителей (вычислительные нити) работают (потоки заданий) в некотором не слишком большом помещении. Из-за толкотни (конфликты доступа к памяти) на ограниченном пространстве (кэш память) или ограниченности скорости поставки строительных ресурсов (скорость подкачки памяти) могут происходить задержки в работе, вплоть до замедления выполнения задания.

OpenMP: заключение

OpenMP — распараллеливание *арифметики*

Главные черты:

- общая память
- директивы препроцессора
- достаточно легкое распараллеливание

Литература

- **G.M. Amdahl**, *Validity of the single-processor approach to achieve large scale computing capabilities*. AFIPS Joint Spring Conference Proceedings 30 (Atlantic City, NJ, Apr. 18–20), AFIPS Press, Reston VA, **1967**, pp. 483–485. http://www-inst.eecs.berkeley.edu/~n252/paper/Amdahl.pdf
- The **OpenMP** API specification for parallel programming. https://www.openmp.org