Параллельные вычисления

Учебный год – 2014, весенний семестр

Группы 5110, 5116

Лекция 2

Преподаватели:

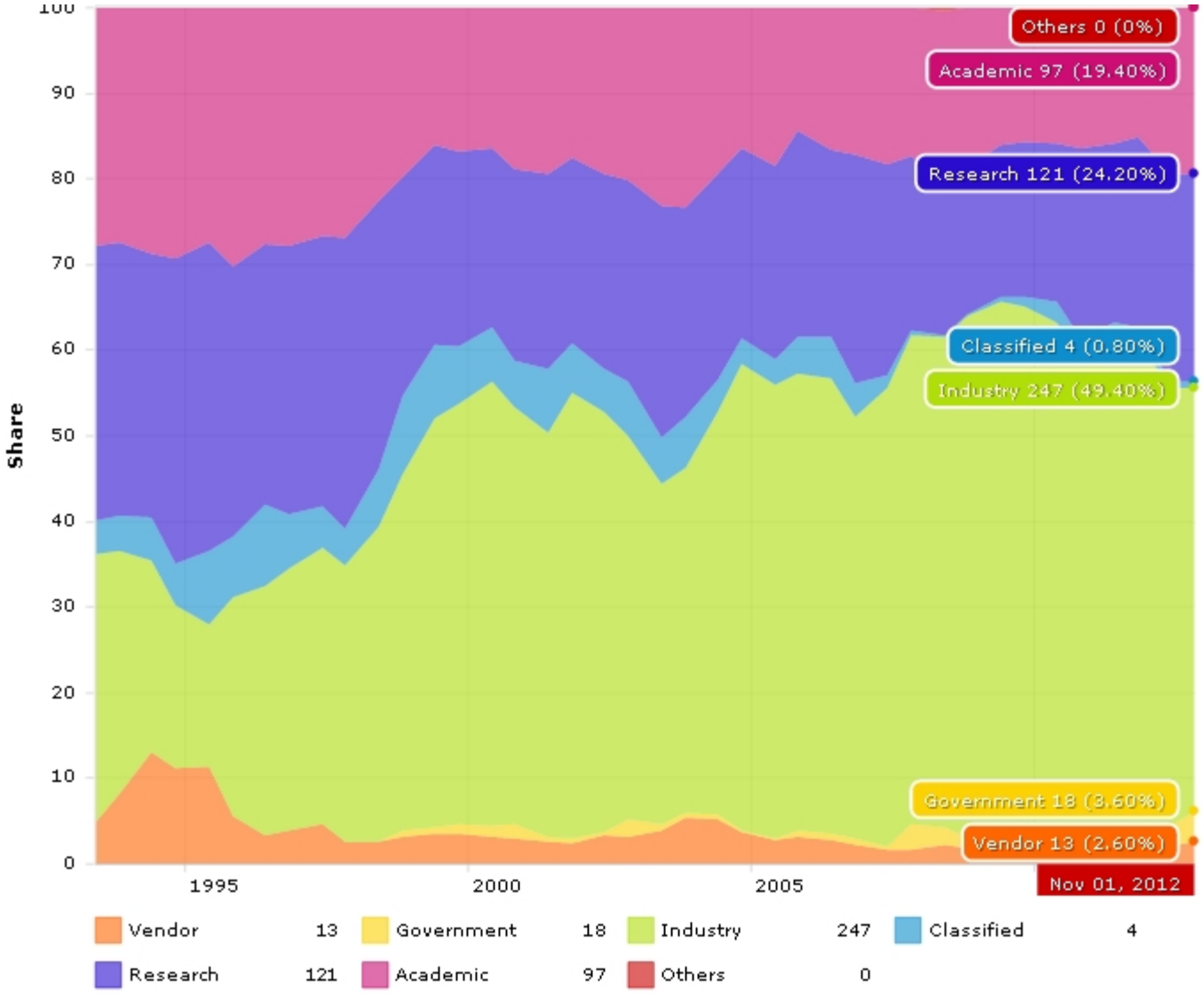
Балакшин Павел Валерьевич

(pvbalakshin@gmail.com),

Соснин Владимир Валерьевич

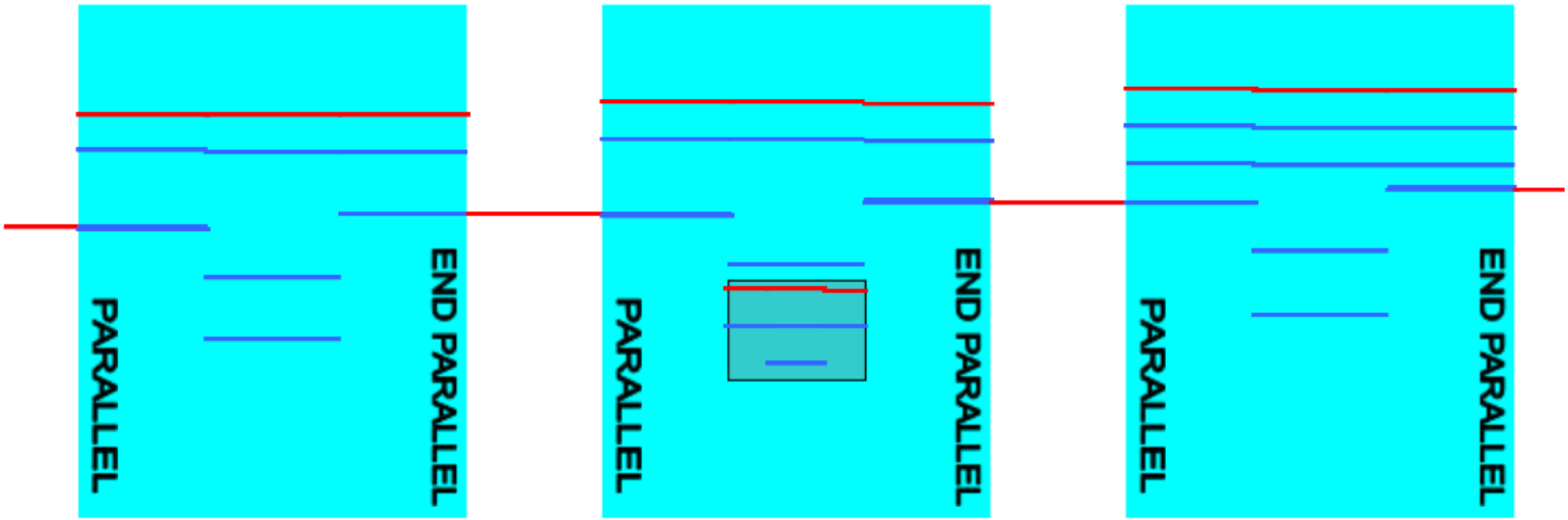
(vsosnin@gmail.com)

Области применения параллельных вычислений



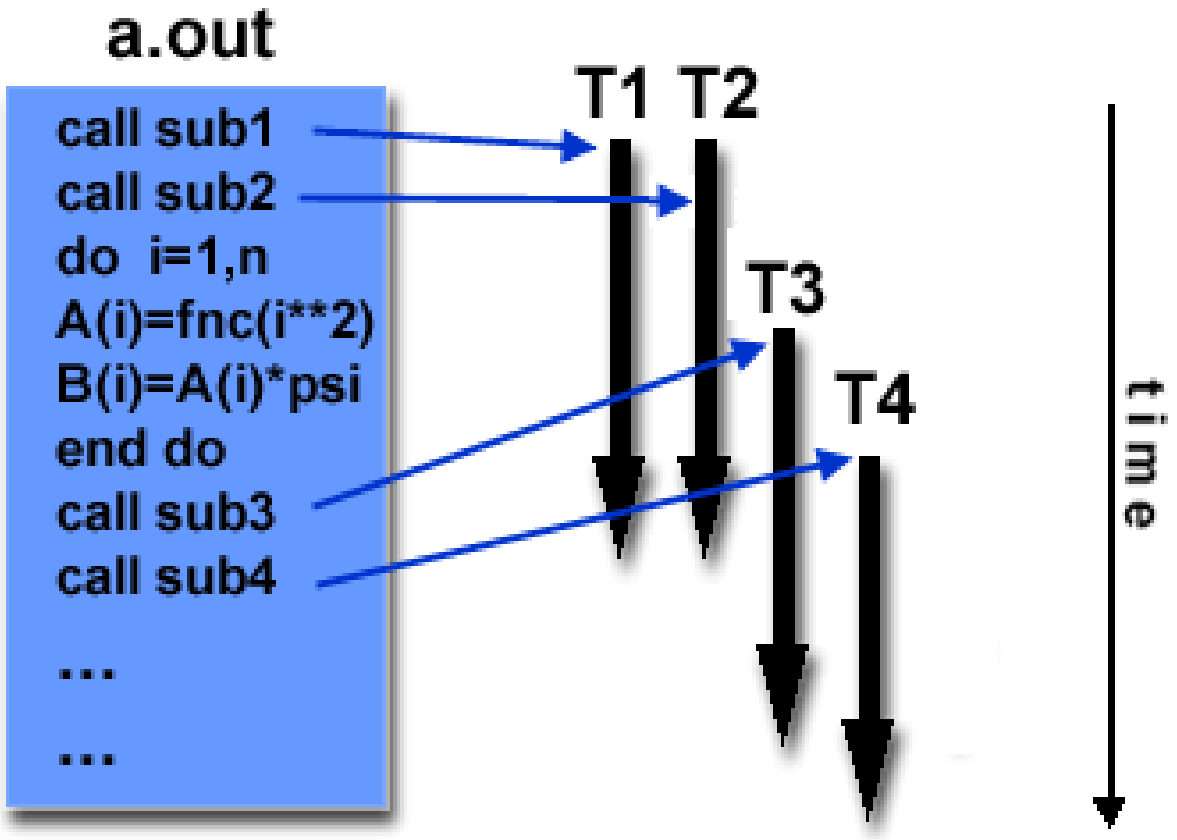
Виды декомпозиции последовательных программ

- Распараллеливание данных.
- Распараллеливание инструкций.

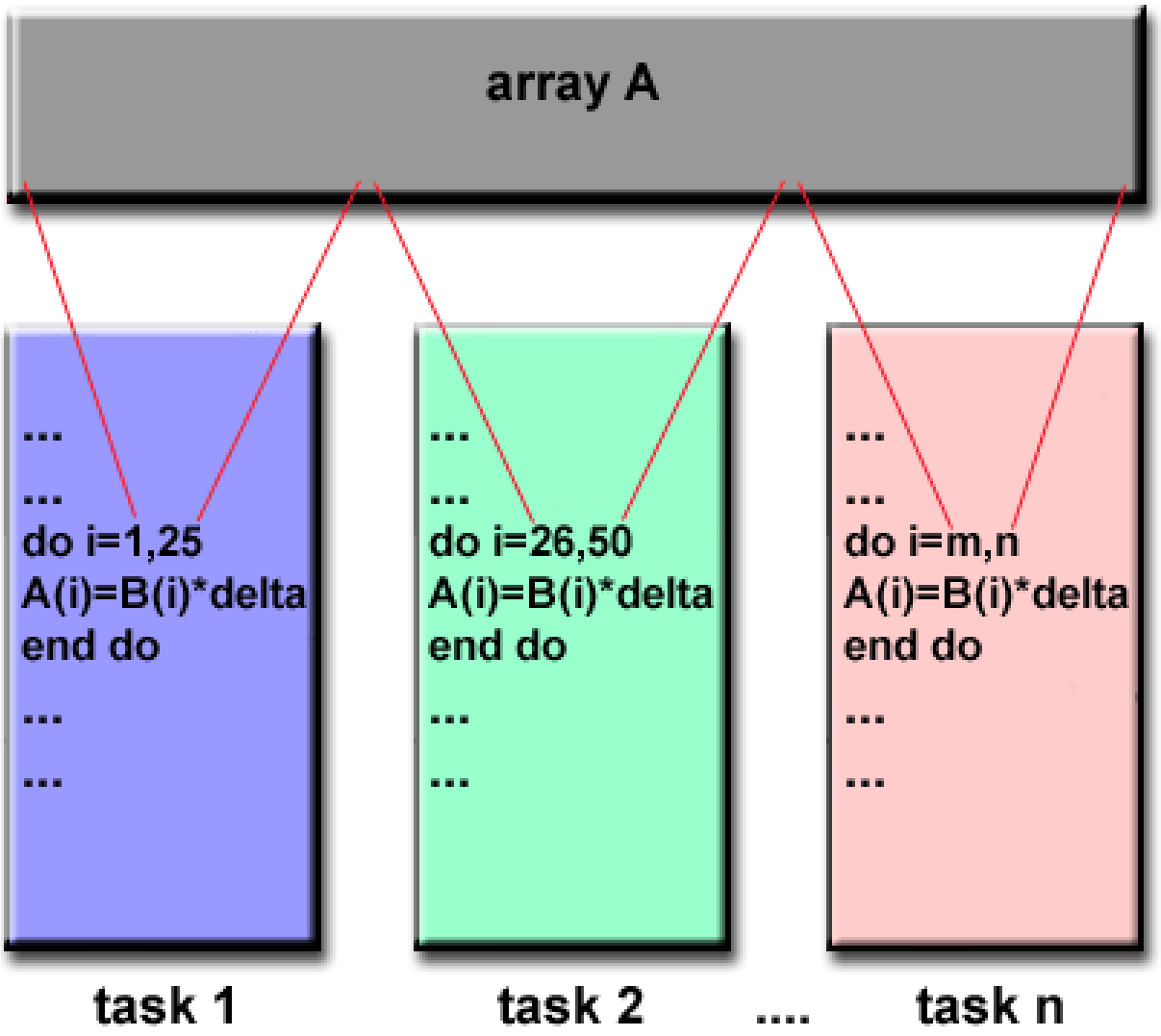


По материалам доц. Бахтина В.А.

Декомпозиция инструкций



Декомпозиция данных



Структура курса

- Shared memory vs Distributed memory. Всегда спорили, но сейчас каждый персональный компьютер SM.
- SM: static threads: ручное управление тредами. Их дорого создавать, поэтому они живут всё время работы программы.
- Очень сложно раздавать поровну работу тредам.

Виды автоматического распараллеливания

- Полностью автоматический.
- Полуавтоматический
  (указание на блокировки и измерение вычислительной сложности).
- Управляемый
  (флаги компилятора, директивы линковщику)

Слабые стороны автоматического распараллеливания

- Возможно ошибочное изменение логики программы.
- Возможно понижение скорости вместо повышения.
- Отсутствие гибкости ручного распараллеливания.
- Эффективно распараллеливаются только циклы.
- Невозможность распараллелить программы со сложным алгоритмом работы.

OpenMP

Парадигма программирования	**SMP** (shared memory parallelism)
Год стандартизации	**1997** (последняя версия 4.0 описана в 2013 г.)
Языки программирования	Фортран, C/C++
Поддержка популярными компиляторами	**gcc** – OpenMP 4.0 **icc** – OpenMP 3.1 **VS2013** – OpenMP 2.0
Ключевые преимущества	1. Простота освоения. 2. «Мало букафф». 3. Прямая (forward) совместимость.

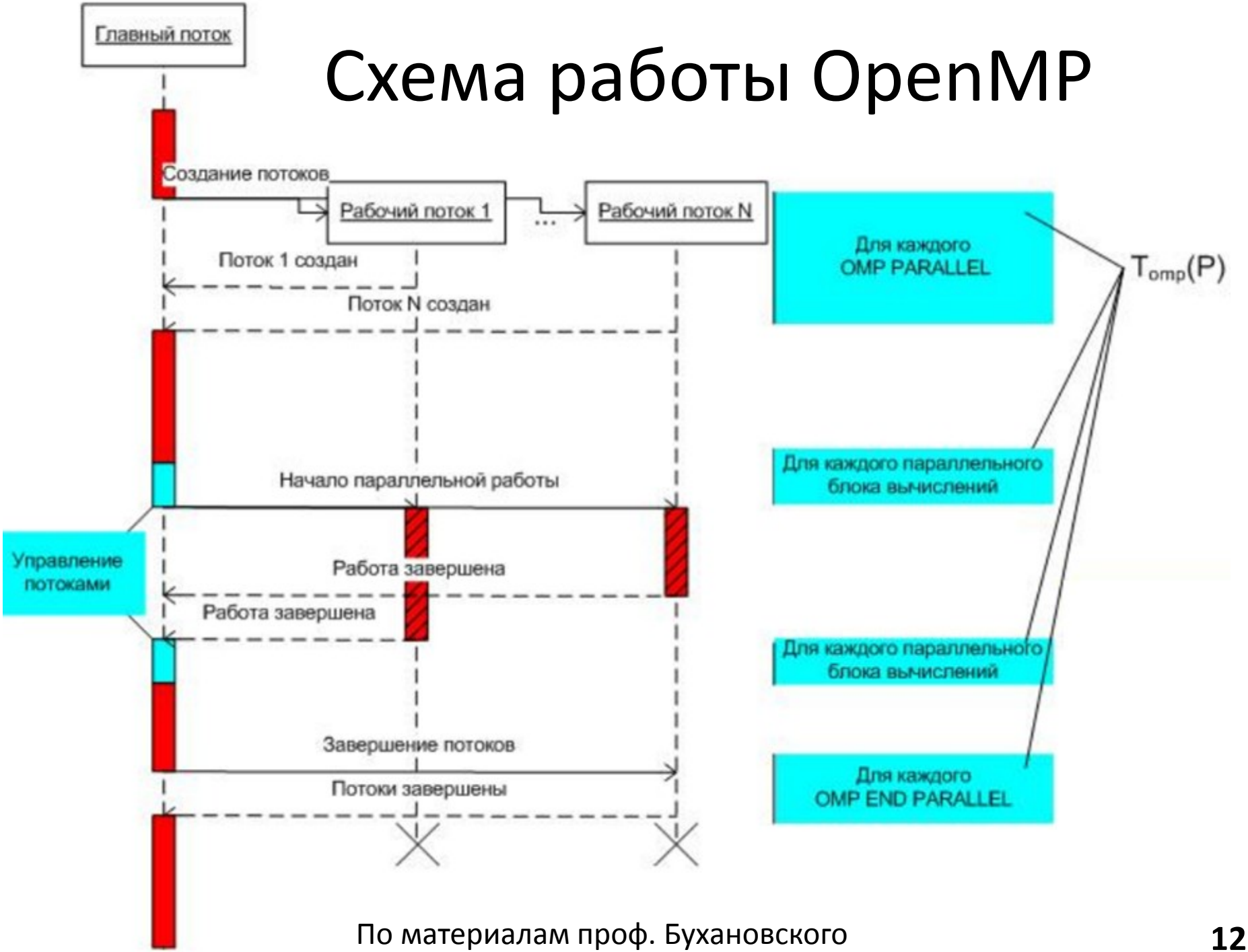
Сильные стороны OpenMP

- Кросс-платформенность.
- Отсутствие сложных механизмов передачи сообщений.
- Простота декомпозиции данных.
- Возможность инкрементного распараллеливания.
- Прямая (forward) совместимость со старыми компиляторами.
- Распараллеливание требует минимальных изменений существующего кода.
- Серьёзная поддержка ведущими производителями, хорошие перспективы развития.

Слабые стороны OpenMP

- Сложно исправлять ошибки синхронизации и гонки.
- Эффективен только в SMP-системах.
- Требует явной поддержки компилятором.
- Масштабируемость ограничена архитектурой памяти.
- Обработка штатных ошибок не развита.
- Нет механизмов привязки потоков к процессорам.
- Ограниченная поддержка вычислений на GPU.
- Дополнительные накладные расходы при запуске программы.

Схема работы OpenMP



По материалам проф. Бухановского

Показатели эффективности

$$T_1(N) = t_c\left(N + M\right)$$

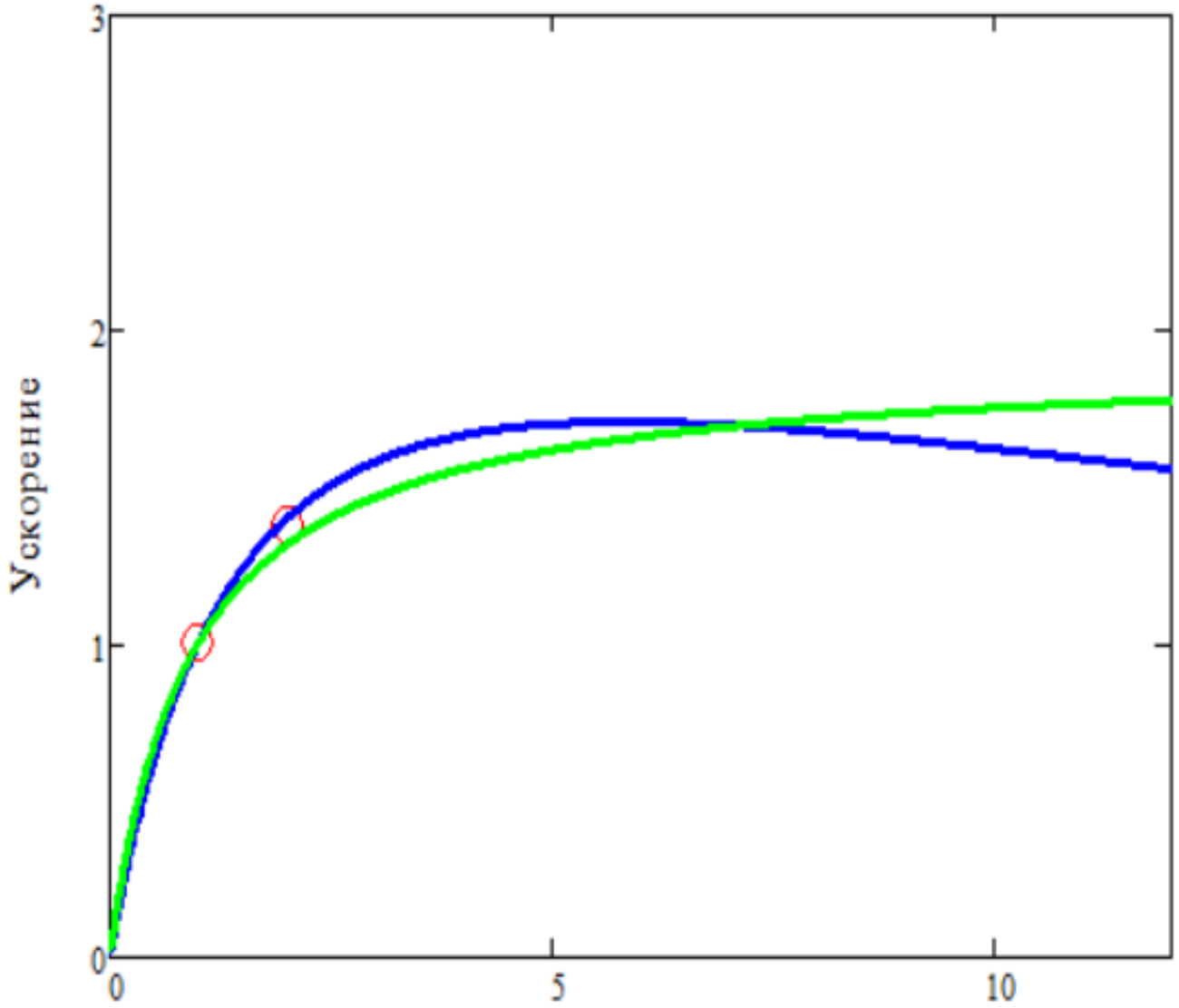
$$T_P(p,\ N) = T_{OMP} + \frac{t_c N}{p} + M \cdot t_c$$

$$T_{OMP} = \alpha(p-1)t_c N$$

$$S(p,\ N) = \frac{T_1}{T_P} = \frac{N + M}{\alpha(p-1)N + \frac{N}{p} + M}$$

По материалам проф. Бухановского

Сравнение с законом Амдала



По материалам проф. Бухановского

Введение в OpenMP

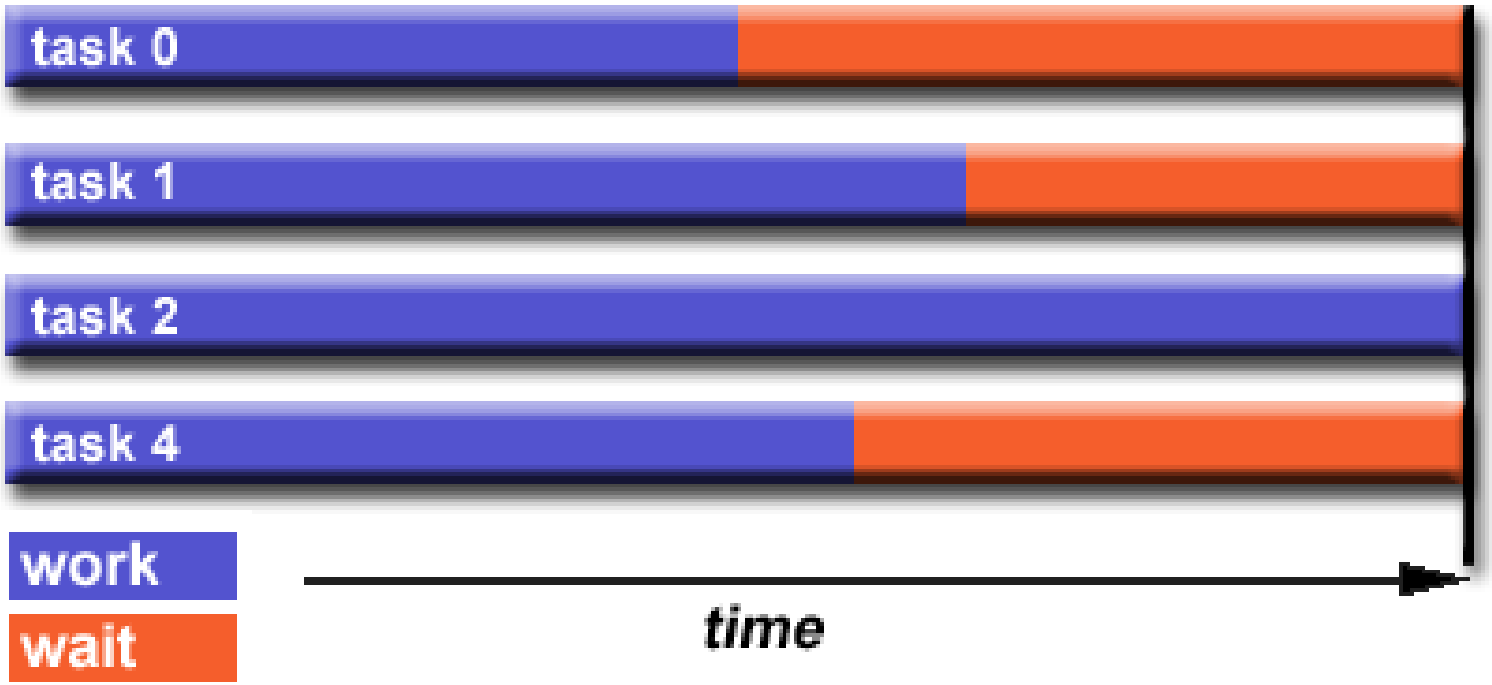
- Подключить заголовок #include <omp.h>
- При компиляции использовать опцию *–fopenmp.*

```
#pragma omp parallel
printf(“Hello, world\n”);
```

```
#pragma omp parallel
printf(“Работает поток номер %d\n", omp_get_thread_num());
```

```
#pragma omp parallel sections
{
        #pragma omp section
        do_fourier_transform(A);
        #pragma omp section
        multiply_matrices(B,C);
}
```

Проблема балансировки нагрузки



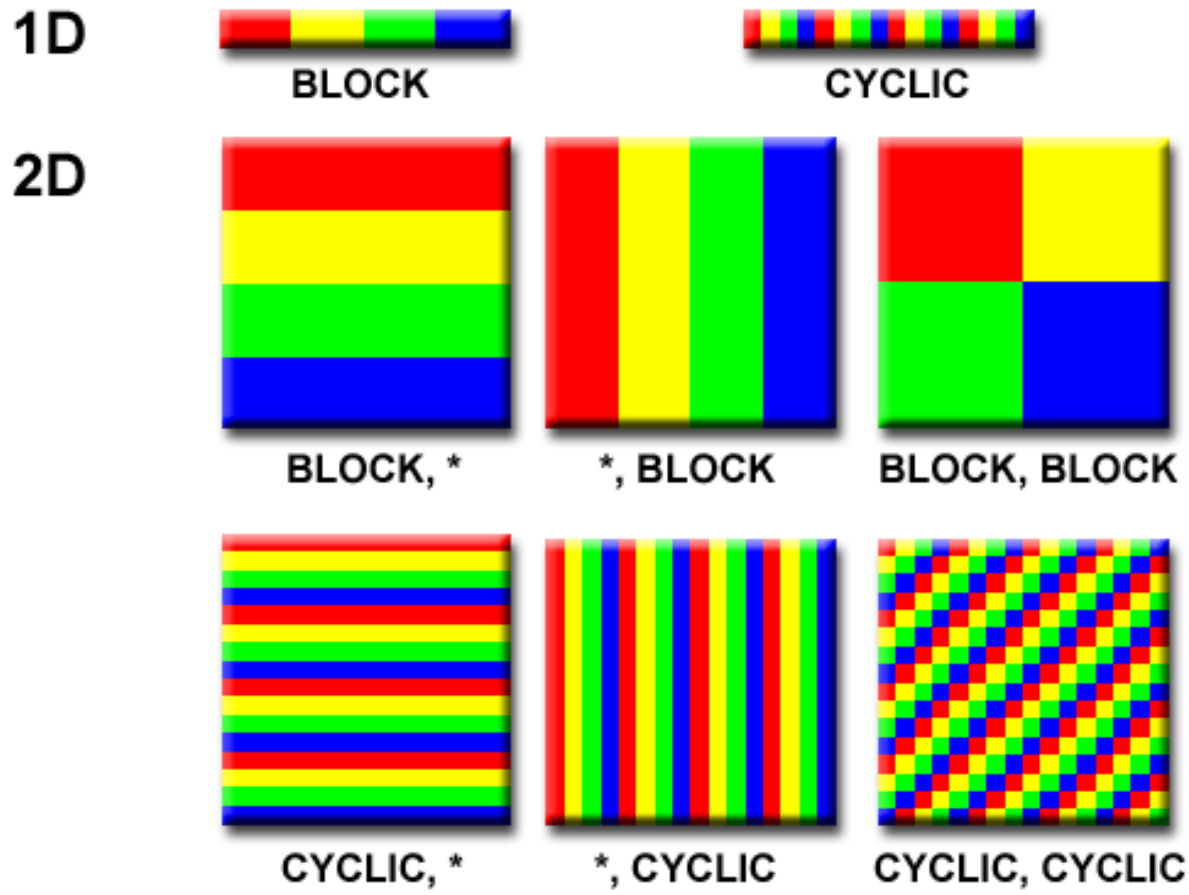
#pragma omp for/ private/reduction

```
#pragma omp parallel for
for(i = 0; i < N; i++)  {
    a[i] = sqrt(sin(x)+cos(x)/ln(x));
}
```

```
#pragma omp parallel for private(j,k)
for(i = 2; i <= N-1; i++)
    for(j = 2; j <= i; j++)
        for(k = 1; k <= M; k++)
            b[i][j] += a[i-1][j]/k + a[i+1][j]/k;
}
```

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

#pragma omp for schedule



#pragma omp for schedule

- #pragma omp for schedule(**static**, chunk_size)
- #pragma omp for schedule(**dynamic**,chunk_size)
- #pragma omp for schedule(**guided**, chunk_size)

```
#pragma omp parallel num_threads(8)
{
   #pragma omp for schedule(dynamic,1)
   for (i = 0; i < 8; i++)
         printf("[1] iter %d, tid %d\n", i, omp_get_thread_num());
  #pragma omp for schedule(dynamic,1)
  for (i = 0; i < 8; i++)
         printf("[2] iter %d, tid %d\n", i, omp_get_thread_num());
}
```