



Нижегородский государственный университет им. Н.И. Лобачевского

Параллельные численные методы

Введение в технологию Intel® Cilk Plus

При поддержке компании Intel

Кустикова В.Д., кафедра математического
обеспечения ЭВМ, факультет ВМК

Содержание

- ❑ Технология Intel® Cilk Plus
- ❑ Терминология
- ❑ Основные компоненты Intel® Cilk Plus и модели исполнения:
 - Ключевые слова (keywords)
 - Гипер-объекты или преобразователи (reducers)
 - Специальное представление массивов (C/C++ Extensions for Array Notation (CEAN))
 - Элементарные функции (elementary functions)



Технология Intel® Cilk Plus

- ❑ Intel® Cilk Plus – расширение языков С и С++, которое упрощает разработку приложений, параллельных по задачам и данным в системах с общей памятью.
- ❑ Разрабатывался с 1994г. в лаборатории Информатики MIT (<http://supertech.csail.mit.edu/cilk/>). В 2009г. выкуплен компанией Intel.
- ❑ Разработчики Cilk Plus предоставляют средства для организации **циклического** и **рекурсивного** («разделяй и властвуй») параллелизма.



Терминология

- ❑ **Задача** – отдельная функция или итерация цикла.
- ❑ **Обработчик (*worker*)** – поток операционной системы, который используется планировщиком Cilk Plus для исполнения задачи.
- ❑ **Нить (*strand*)** – набор инструкций, выполняемых последовательно.

Основные компоненты Intel® Cilk Plus

- ❑ **Ключевые слова** (`cilk_for`, `cilk_spawn`, `cilk_sync`).
Инструмент для организации параллелизма по задачам.
- ❑ **Гипер-объекты** или **преобразователи** (reducers).
Устраняют конкуренцию за переменные, разделяемые между задачами, создавая представление этих переменных для каждой задачи и собирая их обратно в разделяемое значение по завершении всех задач.
- ❑ **Специальное представление массивов** (C/C++ Extensions for Array Notation (CEAN)). Данный компонент обеспечивает параллелизм по данным (векторизацию).
- ❑ **Элементарные функции**. Включают параллелизм по данным для всех функций и операций со специальными массивами или их частями.



Подключение возможности использования средств Intel® Cilk Plus (1)

- ❑ Intel® Cilk Plus поддерживается компилятором **Intel® Windows C/C++ Compiler**.
- ❑ Чтобы применить указанный компилятор, в окне **Solution Explorer** выберите проект.
- ❑ Выполните команду контекстного меню **Intel® Parallel Composer**→**Use Intel® C++....**
- ❑ В диалоговом окне **Confirmation** нажмите **OK**.
- ❑ Кликните правой кнопкой мыши по проекту, в контекстном меню выберите пункт **Properties**.
- ❑ Выполните команду **Configuration Properties**→**C/C++**→**Language**.



Подключение возможности использования средств Intel® Cilk Plus (2)

- ❑ Для свойств **Disable Intel® Cilk Plus Keywords for Serial Semantics** и **Disable All Intel® Language Extensions** установите значение **No**.
- ❑ В исходных кодах проекта необходимо подключить заголовочные файлы. Перечень подключаемых заголовочных файлов определяется компонентами, которые будут задействованы в разработке параллельной реализации.



Основные компоненты Intel® Cilk Plus

КЛЮЧЕВЫЕ СЛОВА



Ключевое слово `cilk_spawn` (1)

□ `cilk_spawn`

- Конструкция, которая может быть использована непосредственно перед вызовом функции, чтобы указать системе, что данная функция *может* выполняться параллельно с вызывающей.
- Вызвавшая функция называется ***родительской***, а вызванная – ***дочерней***.

Ключевое слово `cilk_spawn` (2)

□ Использование `cilk_spawn`:

```
// func() возвращает значение типа type и  
// принимает на вход список аргументов args
```

```
type var = cilk_spawn func(args);
```

```
var = cilk_spawn func(args);
```

```
// func() может возвращать void
```

```
cilk_spawn func(args);
```



Ключевое слово `cilk_spawn` (3)

- Механизм исполнения `cilk_spawn`:
 - Рекомендация планировщику Cilk Plus выполнять нити вызванной и вызывающей функций параллельно.
 - Решение о создании нового обработчика для выполнения вызываемой функции принимается динамически при условии, что в системе имеются свободные ядра.



Ключевое слово `cilk_sync`

□ `cilk_sync`

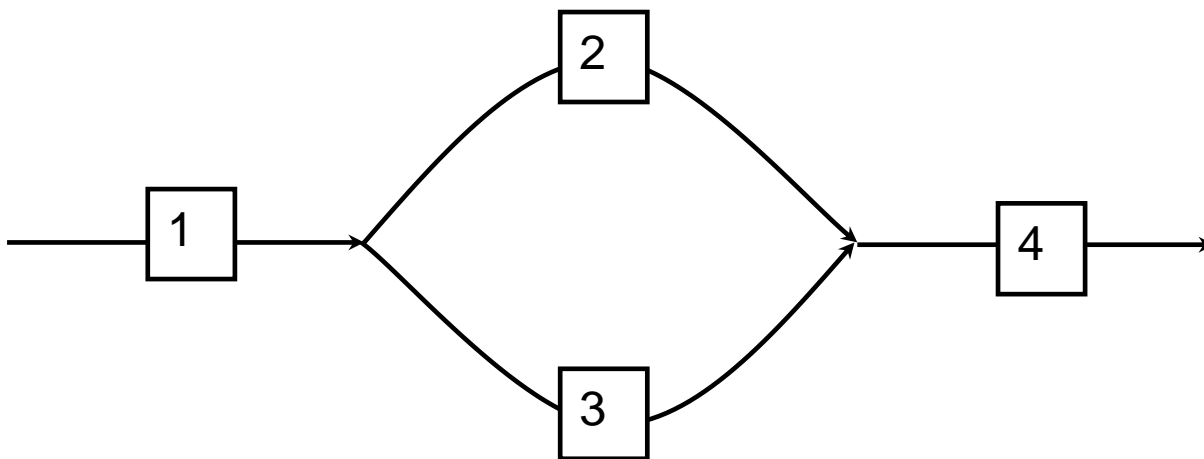
- Точка синхронизации обработчиков.
- Используется, когда дальнейшие вычисления в родительской функции невозможны без результатов дочерней.

□ Использование `cilk_sync`:

`cilk_sync;`

Пример на `cilk_spawn` и `cilk_sync` (1)

```
do_init_stuff(); // выполнение нити 1  
cilk_spawn func3(); // создание нити 3  
do_more_stuff(); // выполнение нити 2 (прод. 1)  
cilk_sync;  
do_final_stuff(); // выполнение нити 4
```



Пример на `cilk_spawn` и `cilk_sync` (2)

- ❑ Планировщик принимает решение о создании потока динамически при наличии доступных ядер.
- ❑ Существует два возможных способа исполнения этих нитей:
 - Последовательное выполнение нитей одним обработчиком в порядке 1-3-2-4.
 - Параллельное выполнение нитей 2 и 3 в разных обработчиках. Сначала выполняются инструкции нити 1 в исходном потоке, затем при вызове **`cilk_spawn`** создается новый обработчик. Нить 2 продолжает выполняться в исходном потоке, а нить 3 – в новом. По завершении нитей 2 и 3 начинает работать нить 4 в исходном потоке.



Ключевое слово `cilk_for` (1)

□ `cilk_for`

- Конструкция, предназначенная для распараллеливания циклов с известным количеством повторений.
- Требования к циклу:
 - Независимость итераций по данным.
 - Тело цикла не должно содержать операторов принудительного перехода (`break`, `continue`, `return`).
 - Наличие только одной переменной-счетчика.



Ключевое слово `cilk_for` (2)

□ Использование `cilk_for`:

```
cilk_for (int i = 0; i < 1000; ++i)
{
    //...
}
```



Ключевое слово `cilk_for` (3)

□ Механизм исполнения `cilk_for`:

- В процессе компиляции тело цикла конвертируется в функцию, которая вызывается рекурсивно в соответствии со стратегией «разделяй и властвуй».

```
void run_loop(first, last) {  
    if (last - first) < grainsize) {  
        for (int i = first; i < last; ++i) LOOP_BODY;  
    }  
    else {  
        int mid = (last-first)/2;  
        cilk_spawn run_loop(first, mid);  
        run_loop(mid, last);  
    }  
}
```

- Планировщик автоматически распределяет поддеревья рекурсии между обработчиками.



Пример использования cilk_for

- Умножение плотной матрицы на вектор:

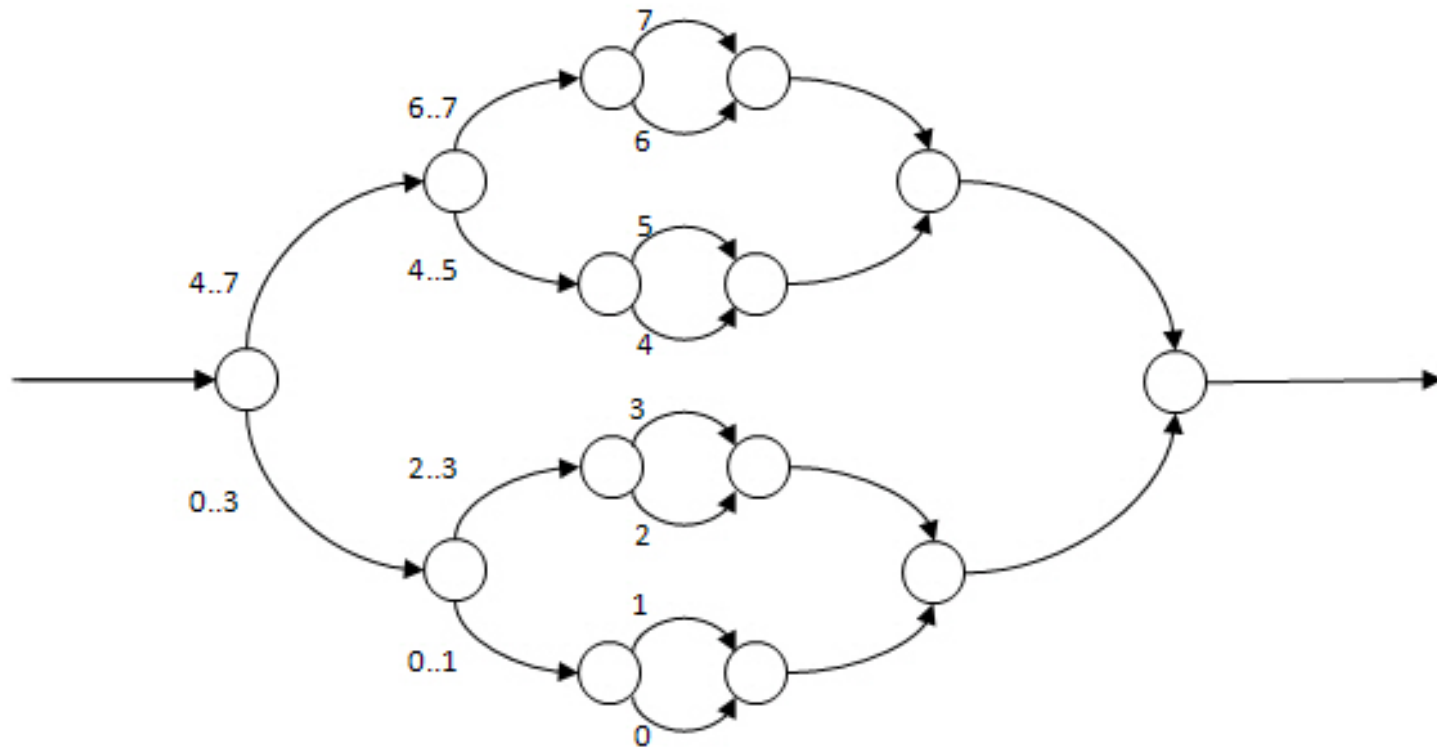
```
#include <cilk/cilk.h>

double dotProd(double *a, double *b, int n);
void Multiplication(double *A, double *x,
                   int n, double *b)
{
    cilk_for (int i = 0; i < n; i++)
    {
        b[i] = dotProd(&(A[i * n]), x, n);
    }
}
```



Разница между использованием `cilk_for` и вызовом `cilk_spawn` в цикле (1)

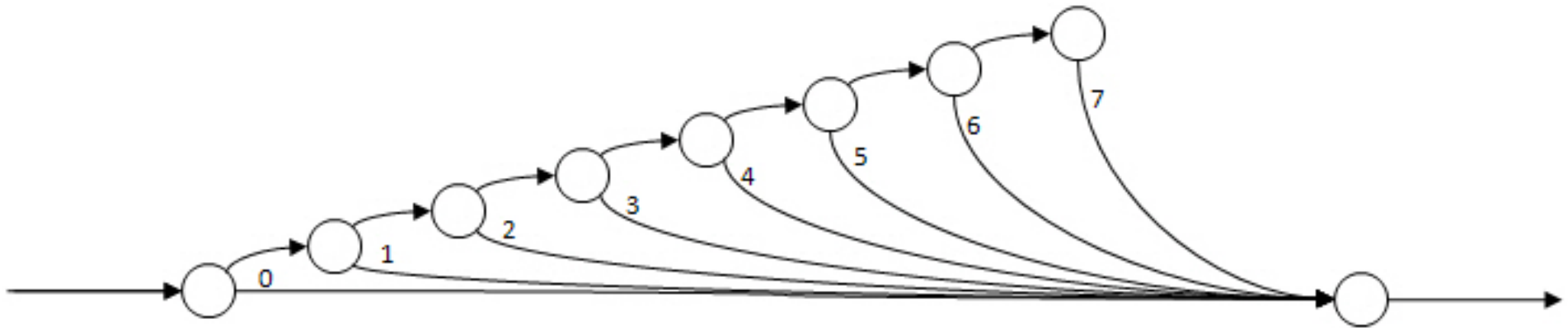
- Диаграмма нитей при использовании конструкции **`cilk_for`**:



- http://software.intel.com/sites/products/documentation/studio/composer/en-us/2011/compiler_c/index.htm#cref_cls/common/cilk_for.htm

Разница между использованием `cilk_for` и вызовом `cilk_spawn` в цикле (2)

- Диаграмма нитей при многократном вызове `cilk_spawn`:



- http://software.intel.com/sites/products/documentation/studio/composer/en-us/2011/compiler_c/index.htm#cref_cls/common/cilk_for.htm

Разница между использованием `cilk_for` и вызовом `cilk_spawn` в цикле (3)

- ❑ Недостатки многократного вызова `cilk_spawn`:
 - Несбалансированность нагрузки между обработчиками, т.к. каждый дочерний обработчик выполняет только одну итерацию.
 - Накладные расходы на синхронизацию.

- ❑ Замечание: для циклов с небольшим количеством итераций и значительными вычислениями в теле цикла разница по времени работы практически не заметна.

Основные компоненты Intel® Cilk Plus

ПРЕОБРАЗОВАТЕЛИ (REDUCERS)



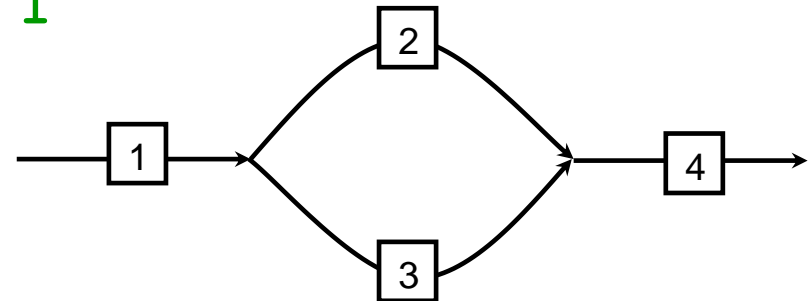
Гипер-объекты или преобразователи (reducers)

- ❑ **Преобразователь (*reducer*)** – это переменная, которая может быть безопасно использована несколькими нитями, запущенными параллельно в разных обработчиках.

- ❑ Свойства преобразователей:
 - Обеспечивают надежный доступ к нелокальным переменным без «гонок» данных.
 - Не требуют явного применения примитивов взаимной блокировки разделяемых переменных.
 - Сохраняется последовательная семантика кода.


Пример


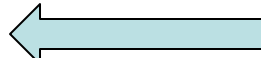



```
#include <cilk/cilk.h>
#include <cilk/reducer_opadd.h>
cilk::reducer_opadd<unsigned int> n(1);
void compute() {
    n++;
}
int main(int argc, char* argv[]) {
    int k = 0; // нить 1
    cilk_spawn compute(); // нить 2
    n++; // нить 3 - продолжение 1
    cilk_sync;
    return 0; // нить 4
}
```



Пример. Нити захватываются одним обработчиком

```
cilk::reducer_opadd<unsigned int> n(1);  n = 1
```

```
void compute() {  
    n++;  n = 2  
}
```

```
int main(int argc, char* argv[]) {  
    int k = 0; // нить 1   
    cilk_spawn compute(); // нить 2   
    n++; // нить 3 - продолжение 1  n = 3  
    cilk_sync;   
    return 0; // нить 4   
}
```




Пример. Параллельные нити выполняются разными обработчиками

```
cilk::reducer_opadd<unsigned int> n(1);
```




$n = 1$

```
void compute() {  
    n++;  
}
```



$n = 1$



$n = 2$

```
int main(int argc, char* argv[]) {  
    int k = 0; // нить 1  
    cilk_spawn compute(); // нить 2  
    n++; // нить 3 - продолжение 1  
    cilk_sync;  
    return 0; // нить 4  
}
```



нить 1



нить 2



нить 3 - продолжение 1

$n = 1$

Совмещение, $n = 3$

Захват, инициализация $n = 0$



нить 4

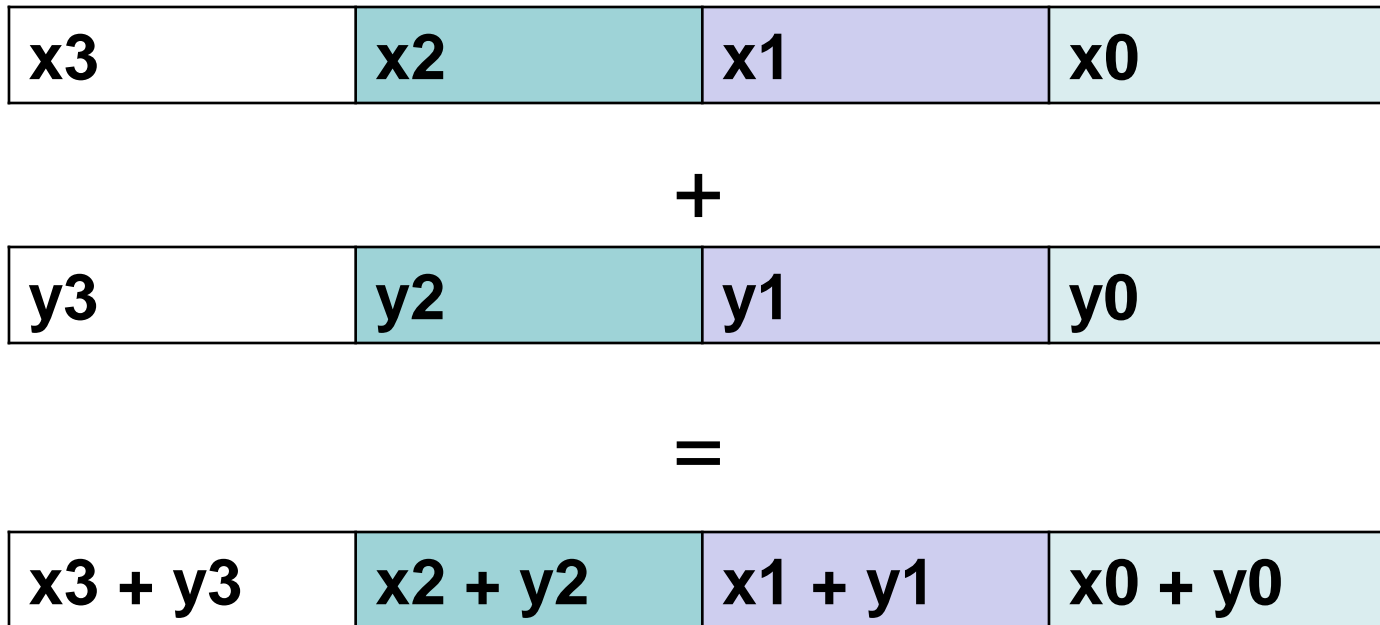
Основные компоненты Intel® Cilk Plus

СПЕЦИАЛЬНОЕ ПРЕДСТАВЛЕНИЕ МАССИВОВ (SEAN)



Специальное представление массивов (C/C++ Extensions for Array Notation (CEAN))

- Специальное представление массивов обеспечивает параллелизм по данным за счет векторизации (SIMD).



Объявление массивов

- ❑ Массив фиксированной длины:

```
int a[28]; // auto
int b[28][35]; // auto
int (*p2d)[28]; // heap
```

```
void example(int a[28]);
```

- ❑ Массив переменной длины:

```
int a[n]; // auto
int b[n][m]; // auto
int (*p2d)[n]; // heap
```

```
void example(int n, int m, int a[n][m]);
```



Обращение к частям массива

□ Обозначение массивов

`<array base> [<lower_bound> : <length> : <stride>]`

- ‘<stride>’ опционально (по умолчанию stride=1)
- отсутствие ‘<length>:<stride>’ означает length=1.
- ‘:’ выбирает все элементы в данном измерении.

□ Примеры:

`A[:]` // Все элементы вектора A

`B[3:45]` // Элементы вектора B с 3 по 45

`C[:,5]` // Столбец 5 матрицы C

`D[0:3:2]` // Элементы 0,2 вектора D

`E[0:3][0:4]` // 12 элементов с E[0][0] по E[2][3] (подматрица)



Основные компоненты Intel® Cilk Plus

ЭЛЕМЕНТАРНЫЕ ФУНКЦИИ



Элементарные функции (1)

□ Редукция:

- суммирование (`__sec_reduce_add(a[:])`);
- умножение (`__sec_reduce_mul(a[:])`);
- поиск минимального / максимального элемента (`__sec_reduce_min(a[:])` / `__sec_reduce_max(a[:])`)
- др.

□ Применение скалярных функций ко всем элементам массива:

```
a[:] = sin(b[:]);
```

```
a[:] = pow(b[:], c); // b[:] в степени c
```

```
a[:] = pow(c, b[:]); // c в степени b[:]
```

```
a[:] = foo(b[:], d[:]); // foo - пользовательская функция
```



Элементарные функции (2)

- ❑ Элементарные функции могут создаваться пользователем:

// объявление функции

```
__declspec (vector) double foo(double x, double y)
{
    return x + y;
}
```

// вызов функции

```
a[:] = foo(b[:], d[:]);
```



Условные выражения

□ Пример поэлементных операций:

```
for (int i = 0; i < n; i++) {  
    if (a[i] > b[i]) {  
        c[i] = a[i] - b[i];  
    }  
    else {  
        d[i] = b[i] - a[i];  
    }  
}
```

□ Преобразованный цикл:

```
c[0:n] = (a[0:n] > b[0:n]) ? a[0:n] - b[0:n] : c[0:n];  
d[0:n] = (a[0:n] <= b[0:n]) ? b[0:n] - a[0:n] : d[0:n];
```



Заключение

- ❑ Intel® Cilk Plus – средства написания параллельных приложений в последовательной семантике.
- ❑ В состав Intel® Cilk Plus входят средства по распараллеливанию рекурсивных («разделяй и властвуй») алгоритмов и циклических конструкций.
- ❑ Intel® Cilk Plus обеспечивает автоматическое распределение нагрузки.
- ❑ Intel® Cilk Plus предоставляет средства организации параллелизма по данным (CEAN).

Вопросы

□ ???



Авторский коллектив

- ❑ Кустикова Валентина Дмитриевна,
ассистент кафедры
Математического обеспечения ЭВМ факультета ВМК ННГУ.
valentina.kustikova@gmail.com

