Шаблоны || программирования

Калишенко Е.Л.
ПОМИ 2014

Этапы разработки

- Определить место || *(Finding concurrency)*
- Выбор алгоритма/шаблона *(Algorithm structure)*
- Отбор подходящих || структур *(Supporting structures)*
- Выбор механизма реализации *(Implementation mechanisms)*

Базовые идеи design space

Выбор декомпозиции:

- По данным
- По управлению (задачам)

Анализ зависимостей:

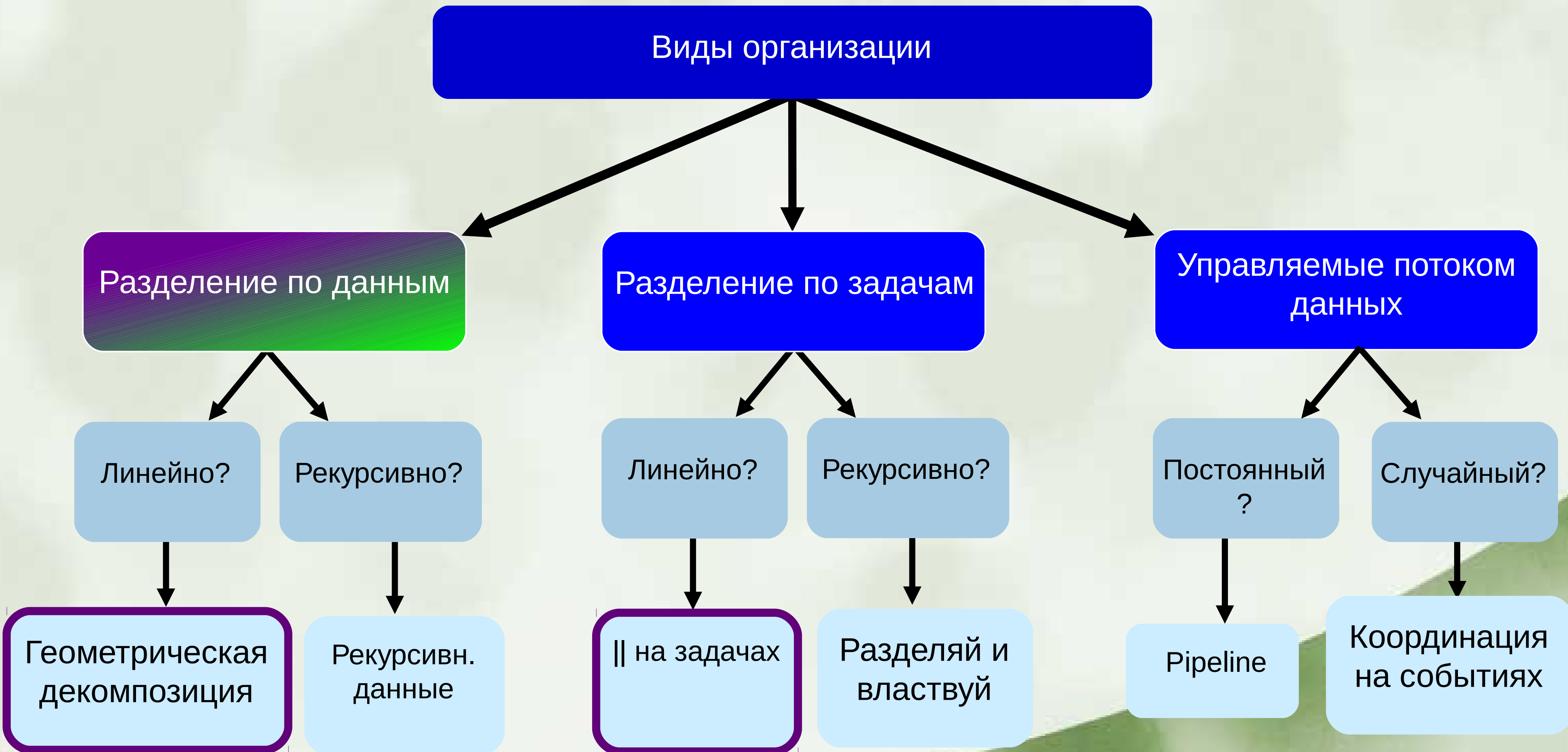
- Группировка задач
- Порядок выполнения задач
- Определение разделяемых данных

Нужно помнить

Принципы проектирования:

- *Эффективность*
- *Простота*
- *Переносимость*
- *Масштабируемость*

Организация вычислений



Геометрическая декомпозиция

Примеры

- *Моделирование климата*
- *Фракталы*

Особенности

- *Разделение структур данных:*
  - *массивы — берём последовательные части*
  - *списки — псевдоэлементы или подсписки*
  - *графы - подграфы*

Подходит «Parallel loops»

OpenMP

```
#pragma omp parallel for schedule(static, 500)
for(int i=0; i<n; i++)
  invariant_amount_of_work(i);
```

Intel TBB

```
parallel_for(blocked_range<int>(0, nElements, 100),
             ArraySummer( p_A, p_B, p_SUM_TBB ) );
```

А ещё есть: `blocked_range2d` и `blocked_range3d`

Ещё подходит SPMD

Single Program Multiple Data: один код над разными данными, совершенно очевидная вещь — одна программа, управляющая вычислениями (процессами, потоками...)

Шаги:

- Инициализироваться
- Раздать идентификаторы нитям/процессам
- Разделить данные
- Завершиться

Parallel Boost Graph Library

- MPI: распределённые графы
- Генераторы графов
- Алгоритмы:
  - Поиск
  - Кратчайший путь
  - Минимальное остовное дерево
  - Раскраска графа
  - ...

Recursive data

Задача

- *Как сделать || операции над рекурсивными структурами (списки, деревья, графы)?*

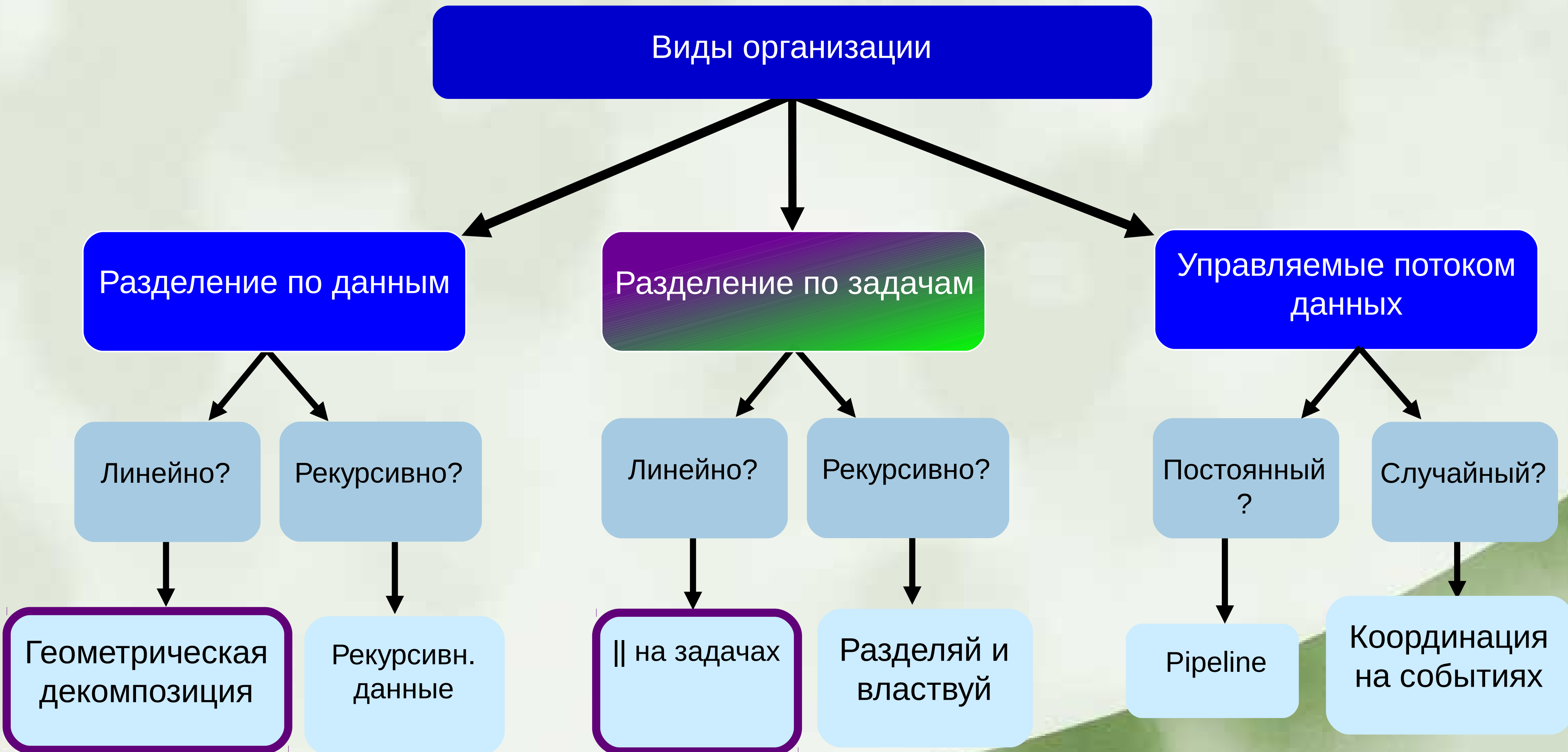
Пример

- *Найти в лесу корни деревьев*

Решение

- *Последовательное O(N) операций / времени*
- *|| O (N log N) операций / O (log N) времени*

Организация вычислений



|| на задачах

Примеры

- *Ray tracing*
- *Задачи молекулярной динамики*

Особенности

- *Динамическое создание задач (методы ветвей и границ)*
- *Ожидание задач или его отсутствие (сортировка или поиск)*
- *Размер задач*

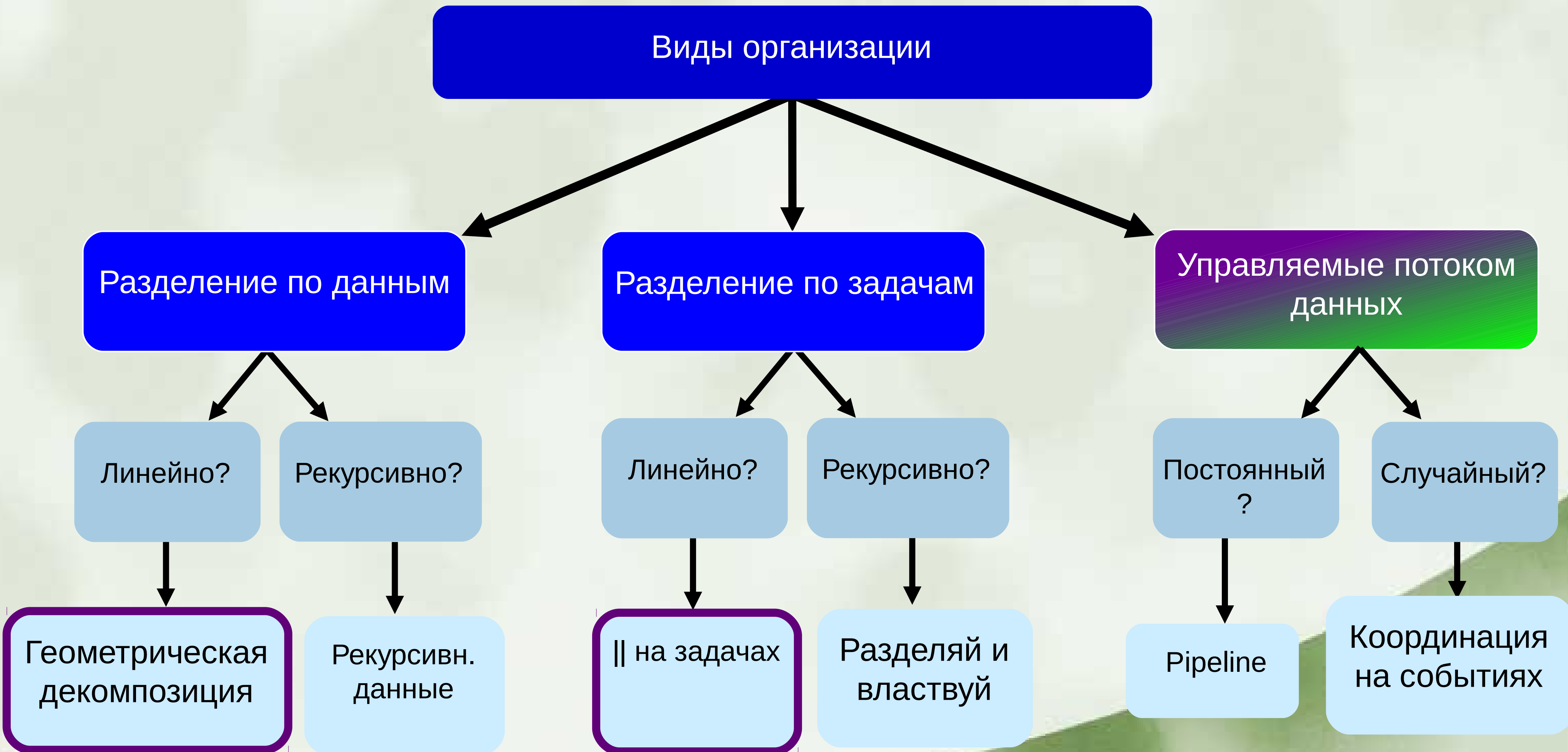
Принципы разделения на задачи

- *Задач >= числу потоков (ядер)*
- *Объём вычислений не превышает издержки*
- *Борьба с зависимостями (shared data — последний рубеж):*
  - *Приветствуется дублирование данных (и, возможно, вычислений) и редукция*
  - *Уборка индуцированных переменных в циклах*
  - *Разделить зависимости по результату через редукцию по данным*

Назначение задач на потоки

- *Формирование очереди задач: notify — получаем классические:*
  - *Producer/consumer — поток кладёт задачи*
  - *Master/Worker — поток готовит задачи*
  - *Приходят пулы потоков*
- *Выбор способа планирования: статическое/динамическое*
  - *балансировка нагрузки: + к динамике*
  - *минусы динамического назначения?*

Организация вычислений



Pipeline

Пример

- *Обработка сигналов*
- *Видео*
- Shell! *«*`cat sampleFile | grep "word" | wc`*»*

Пример

```
parallel_pipeline(threadCount,
    tbb::make_filter<void, std::string*>(
        tbb::filter::serial,
        [&in](tbb::flow_control& fc) -> std::string* {
            auto line = new std::string();
            getline(in, *line);
            if (!in.eof() && line->length() == 0)
            {
                fc.stop();
                delete line;
                line = 0;
            }
            return line;
        }
    )
    &
    tbb::make_filter<std::string*, std::string*>(
        tbb::filter::parallel,
        [](std::string* line) {
            tbb::parallel_sort(line->begin(), line->end());
            return line;
        }
    )
    &
    tbb::make_filter<std::string*, void>(
        tbb::filter::serial,
        [&out](std::string* line) {
            out << *line << std::endl;
            delete line;
        }
    )
);
```

Local Serializer

```
class Serializer {
    tbb::concurrent_queue<WorkItem*> queue;
    tbb::atomic<int> count; // Count of queued items and in-flight item

    void moveOneItemToReadyPile() { // Transfer item from queue to ReadyPile
        WorkItem* item;
        queue.try_pop(item);
        ReadyPile.add(item);
    }

public:
    void add( WorkItem* item ) {
        queue.push(item);
        if( ++count==1 )
            moveOneItemToReadyPile();
    }

    void noteCompletion() {        // Called when WorkItem completes.
        if( --count!=0 )
            moveOneItemToReadyPile();
    }
};
```

Double check (lock)

```
template<typename T, typename Mutex=tbb::mutex>
class lazy {
   tbb::atomic<T*> value;
   Mutex mut;

public:
   lazy() : value() {}              // Initializes value to NULL

   ~lazy() {delete value;}

   T& get() {
     if( !value ) {              // Read of value has acquire semantics.
       Mutex::scoped_lock lock(mut);
       if( !value ) value = new T(); // Write of value has release semantics
     }
     return *value;
   }
};
```

Double check (lock-free)

```
template<typename T>
class lazy {
   tbb::atomic<T*> value;

public:
   lazy() : value() {}                          // Initializes value to NULL

   ~lazy() {delete value;}

   T& get() {
     if( !value ) {
       T* tmp = new T();

       if( value.compare_and_swap(tmp,NULL)!=NULL )
         // Another thread installed the value, so throw away mine.
         delete tmp;
     }
     return *value;
   }
};
```

Модели программ

- SPMD
  - Каждый поток имеет id, по которому просит данные, выбирает итерации...
- Loop ||
  - Объединение вложенных циклов
  - Уборка зависимостей по итерациям
  - Возможна простая статическая балансировка
- Boss/Worker
  - Нужна для динамической балансировки

-D_GLIBCXX_PARALLEL

- Несколько десятков реализаций алгоритмов из <algorithm>: find_if, max_element...
- Использует OpenMP