Высокоуровневое параллельное программирование

Востокин Сергей Владимирович

План

- Понятие алгоритмического скелетона
- Классификация фреймворков алгоритмических скелетонов (ASkF)
- Пример 1: MapReduce
- Пример 2: Bag-of-tasks
- Пример 3: Язык разметки Templet

Понятие алгоритмического скелетона

Определение скелетона

- Высокоуровневая модель параллельного, многопоточного или распределенного программирования
  - Модель программирования: общий способ организации параллельных программ, учитывающих компиляцию и исполнение
- Цель: уменьшить сложность разработки
- Программа состоит из последовательных частей, взаимодействие между которыми неявно определяется используемым скелетоном

Основные литературные ссылки

- ***Murray Cole***. "Algorithmic Skeletons: structured management of parallel computation" MIT Press, Cambdridge, MA, USA, 1989
- Horacio González-Vélez and Mario Leyton (November–December 2010). "A survey of algorithmic skeleton frameworks: high-level structured parallel programming enablers". Software: Practice and Experience 40 (12): 1135–1160. doi:10.1002/spe.1026.

Встречающиеся синонимы

- Англоязычная литература
  - algorithmic skeleton frameworks
  - parallelism patterns
  - skeleton programming
- Литература на русском
  - типовое решение
  - параллельный фреймворк
  - вычислительная парадигма
  - параллельный паттерн

Что представляет собой программа

- Комбинация высокоуровневой программной структуры и ***кода «заглушек»*** (***dummy***)
- Программа похожа на ***псевдокод***, но предполагает компиляцию, запуск и тестирование кода
- Программа разрабатывается методом ***нисходящего проектирования***, предполагающим декомпозицию кода на блоки с последовательным кодом
- С точки зрения паттернов объектно-ориентированного программирования, концепция алгоритмического скелетона основана на паттерне ***Шаблонный метод***

Классификация фреймворков алгоритмических скелетонов (ASkF)

Функциональная классификация

- Data-parallel (параллелизм данных)
  - Описывают применение одинаковых операций к структурам данных большого объёма
- Task-parallel (параллелизм задач)
  - Определяют взаимодействие между задачами
- Resolution (типовые решения)
  - Определяют общий метод распараллеливания для семейства схожих вычислительных проблем

Data-parallel ASkF

- Map
  - Некоторая операция применяется к набору данных, затем результат объединяется (вариант SIMD)
- Fork
  - Набор различных операций применяется к набору данных (вариант MIMD)
- Reduce
  - Выполнение бинарной ассоциативной операции (например, суммирование) над списком данных

Task-parallel ASkF

- Farm
  - Универсальный планировщик независимых задач (master-worker, master-slave, bag of tasks)
- Pipe
  - Этапы вычислений одновременно выполняются на ступенях конвейера
- Sequential, If, For, While
  - Не содержат параллельных операций, используются при определении вложенных скелетонов

Resolution ASkF

- Divide & Conquer
  - Рекурсивное обобщение скелетона map. Исходная задача делится на подзадачи с контролем глубины рекурсии. Затем происходит объединение результатов подзадач. Пример: сортировка слиянием, сортировка Хоара, метод адаптивной квадратуры
- Branch & Bound
  - Метод ветвей и границ - общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации

Классификация AskF по парадигмам программирования

- Язык координации
  - Используется специальный высокоуровневый язык, который транслируется в целевой язык
- Функциональный стиль
  - Структурированный параллелизм реализуется в виде синтаксического расширения параллельного функционального языка или в виде функторов
- Объектно-ориентированный стиль
  - Элементы скелета параллельной программы представляются в виде классов
- Императивный стиль
  - Используются API, реализованные на процедурных языках

Основные характеристики реализаций ASkF

- Язык программирования
  - Язык при помощи которого определяется скелетон. Используются: функциональные языки, языки координации, языки разметки, императивные языки, объектно-ориентированные языки, графические языки
- Язык исполнения
  - Целевой язык в который компилируется код скелетона. Обычно отличается от языка программирования при использовании функционального языка. Функциональные программы, например, компилируются в C
- Среда исполнения
  - API при помощи которого достигается параллелизм, его абстракции скрыты от программиста. Обычно MPI
- Типобезопасность
  - Способ обнаружения несоответствия типов. В языке программирования или в языке исполнения
- Поддержка вложенности
  - Возможна ли иерархическая композиция скелетонов
- Список поддерживаемых шаблонов

Примеры реализации ASkF (1/2)

	Programming language	Execution language	Distribution library	Type safe	Skeleton nesting	Skeleton set
Alt	Java	Java	Java RMI	Yes	No	map, zip, apply, scan, sort, reduce, replicate
ASSIST	Custom control lang.	C++	TCP/IP + ssh/scp	Yes	No	seq, parmod
Calcium	Java	Java	ProActive	Yes	Yes	seq, pipe, farm, for, while, map, d&c, fork
Eden	Haskell (extension)	C	PVM / MPI	Yes	Yes	map, d&c, pipe, iterUntil, torus, ring
eSkel	C	C	MPI	No	Yes	pipe, farm, deal, Butterfly, hallowSwap
HDC	Haskell (subset)	C	MPI	Yes	Yes	map, red, scan, filter, dca, dcB, dcD, dcE, dcF
HOC	Java	Java	Globus	No	No	farm, pipe, wavefront
JaSkel	Java	Java	RMI	No	Yes	farm, pipe, heartbeat
Lithium	Java	Java	RMI	No	Yes	pipe, map, farm, reduce

Примеры реализации ASkF (2/2)

	Programming language	Execution language	Distribution library	Type safe	Skeleton nesting	Skeleton set
Mallba	C++	C++	NetStream MPI	Yes	No	exact, heuristic, hybrid
Muesli	C++	C++	MPI OpenMP	Yes	Limited	array, matrix, farm, pipe, parallel comp.
Muskel	Java	Java	RMI	No	Yes	farm, pipe, seq, custom
P^3L	Custom control lang.	C	MPI	Yes	Limited	map, reduce, seq, comp, pipe, farm, scan, loop
QUAFF	C++	C	MPI	Yes	Yes	seq, pipe, farm scm, pardo
SAC	Custom	C-like	Threads	No	No	genarray, modarray, fold
SCL	Custom control lang.	Fortran	*Ad hoc* Tools	Yes	Limited	map, scan, farm, fold, SPMD, iterateUntil
Skandium	Java	Java	Threads	Yes	Yes	seq, pipe, farm, for, while,map, d&c, fork
SKELib	C	C	MPI	No	No	farm, pipe
SkeTo	C++	C++	MPI	Yes	No	list, matrix, tree
SkIE	GUI / Custom control lang.	C++	MPI	Yes	Limited	farm, pipe, map reduce, loop
Skil	C (subset)	C	—	Yes	No	pardata, map, fold
SkiPPER	CAML	C	SynDex	Yes	Limited	scm, df, tf, intermem

Пример 1: MapReduce

Основные сведения

- Назначение: распределённая обработка больших объёмов данных
- Реализации: Google MapReduce (C++, Python), Apache Hadoop (Java), Apache CouchDB (Erlang), Disco (Python), Infinispan (Java), MongoDB (C++, JavaScript, C), Riak (Erlang) …
- Можно рассматривать как Resolution skeleton

Компоненты модели программирования MapReduce

- Функции обработки реализуют
  - Распределители (mapper)
  - Редукторы (reducer)
- Структуры данных, через которые взаимодействуют функции
  - Списки
  - Пары <ключ, значение>

Интерфейсы распределителей и редукторов

- **map** (in_key, in_value) -> list(out_key, intermediate_value)
  - Обрабатывает входные пары <ключ, значение>
  - Производит множество промежуточных пар <ключ, значение>
- **reduce** (out_key, list(intermediate_value)) -> list(out_value)
  - Комбинирует все промежуточные значения для конкретного ключа
  - Производит множество объединенных значений (множество обычно состоит из одного значения)

Схема последовательного выполнения

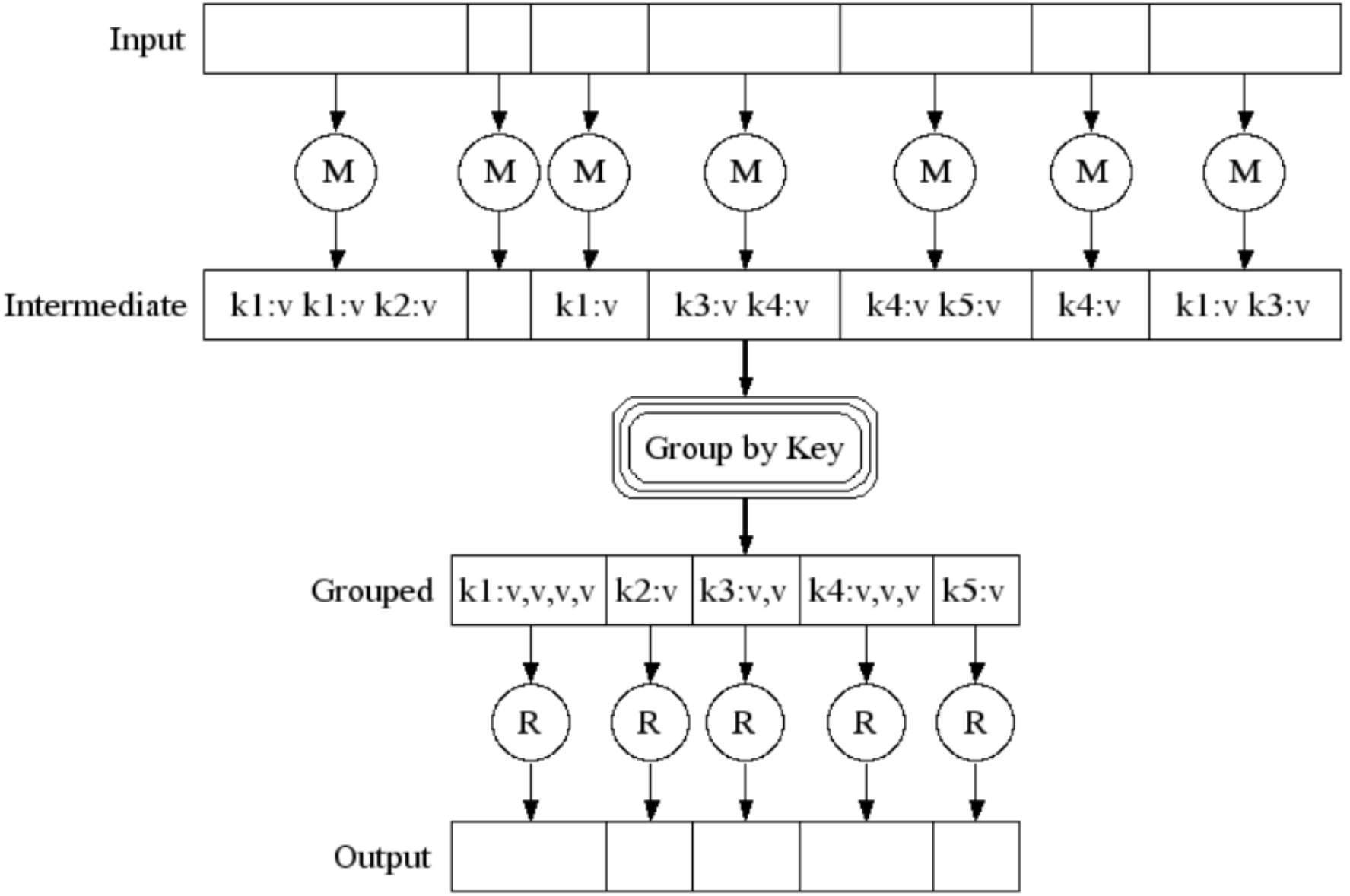
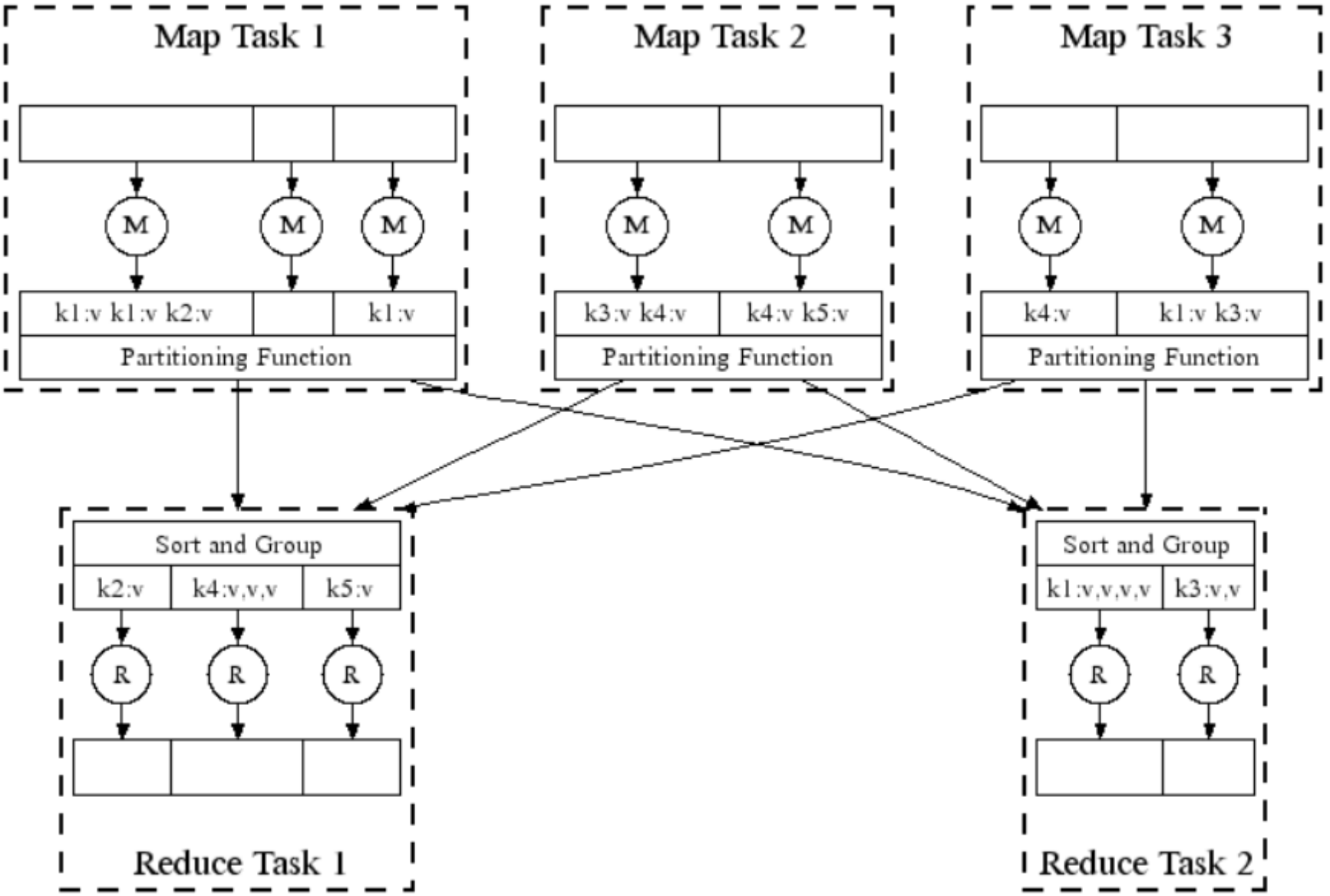


Схема параллельного выполнения



Псевдокод для подсчёта частоты использования слов в документе

```
map(String filename, String document) {
    List<String> T = tokenize(document);
    for each token in T {
        emit ((String)token, (Integer) 1);
    }
}
reduce(String token, List<Integer> values) {
    Integer sum = 0;
    for each value in values {
        sum = sum + value;
    }
    emit ((String)token, (Integer) sum);
}
```

Подробнее о Hadoop

- Чак Лэм. Hadoop в действии. - М.: ДМК Пресс, 2012. - 424 с.: ил.
- Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113. (http://research.google.com/archive/mapreduce.html)
- Проект Apache Hadoop http://hadoop.apache.org/

Пример 2: Bag-of-tasks

Основные сведения

- Вычисление независимых задач в рабочих процессах, управляемых через один головной процесс
- Порядок сбора результатов задач не влияет на конечных результат
- Количество задач может быть заранее неизвестным и определяться в процессе счёта
- Пригоден для распределённых вычислений
- Обеспечивает автоматическую балансировку

Элементы программной модели (1/3)

Структуры данных:

- **task**
  - описывает исходные данные задачи
- **result**
  - описывает результат вычисления задачи
- **bag**
  - хранит информацию о не выданных задачах и промежуточный / конечный результат вычислений

Элементы программной модели (2/3)

Процедуры обработки данных:

- **process** (task) -> result
  - Вычисление независимой задачи
- **get** (bag) -> task
  - Формирование / извлечение очередной задачи
- **put** (result, bag) -> bag
  - Сохранение результата вычисления задачи

Элементы программной модели (3/3)

Функция:

- **if_task** (bag) -> true | false
  - Проверка, имеются ли задачи на обработку

Схема последовательного выполнения

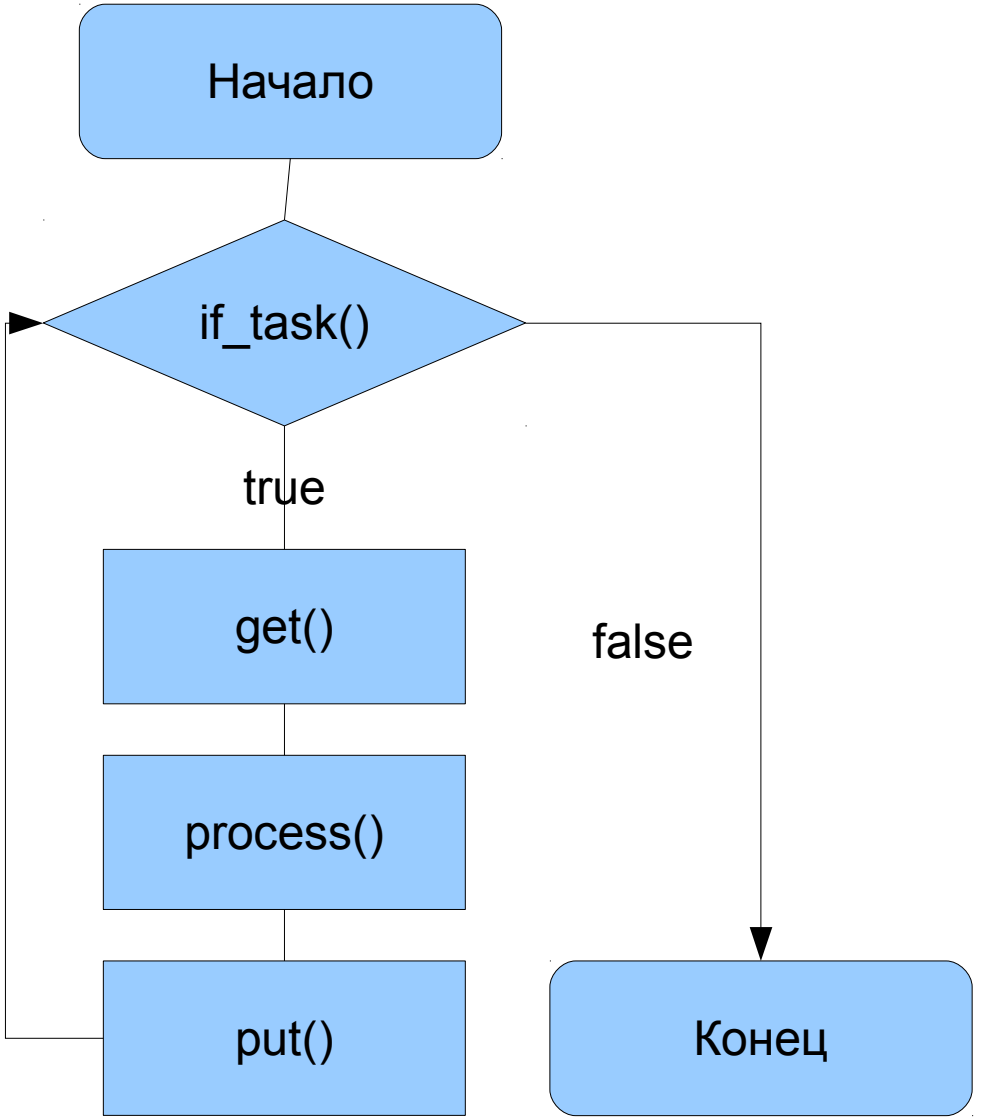
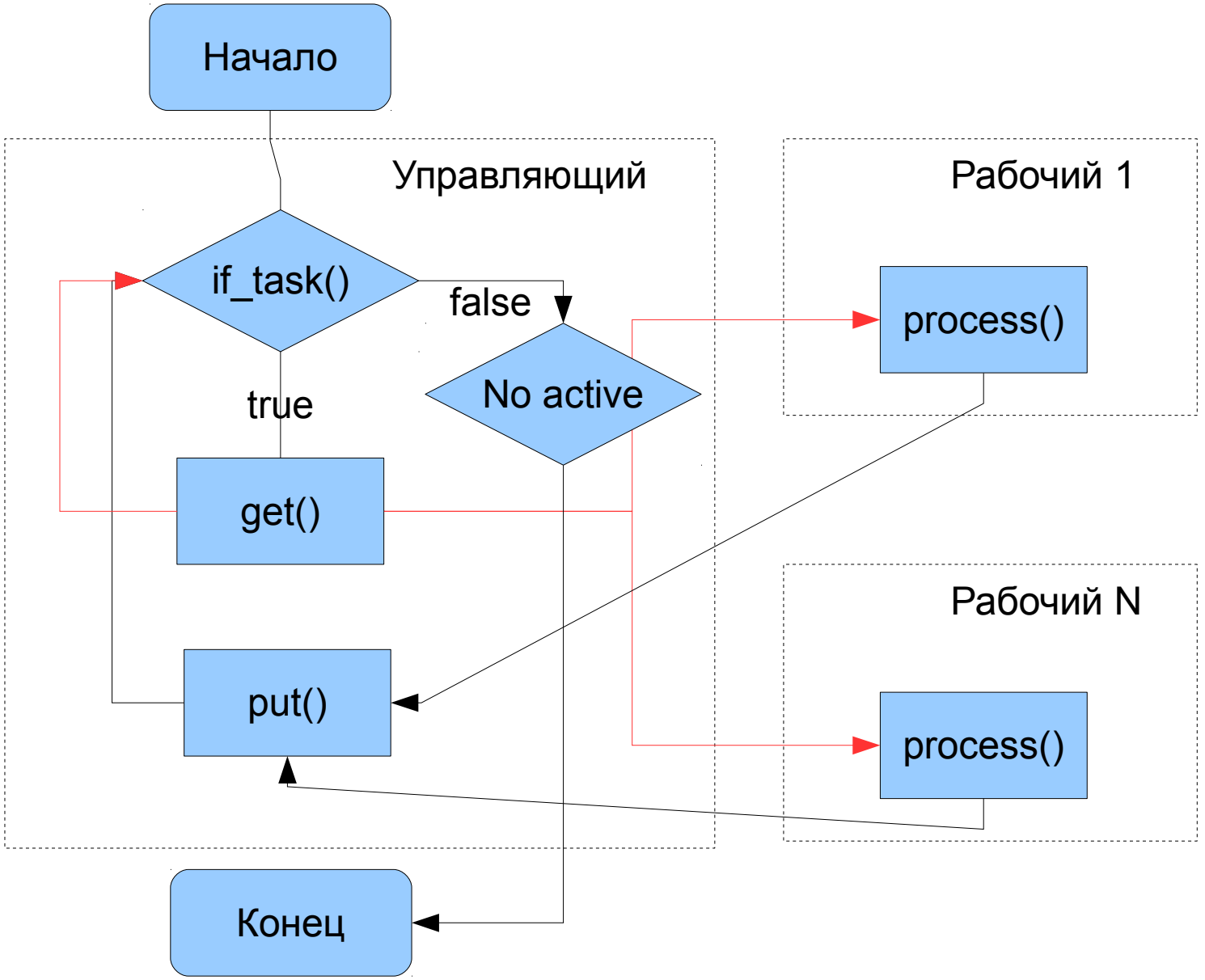


Схема параллельного выполнения



Пример кода построчного умножения матриц (1/3)

```
const int N=10;
double a[N][N],b[N][N],c[N][N];

class TaskBag:public TEMPLET::TBag{
public:
    class TaskBagTask:public TBag::Task{
    public:
        int num;// номер вычисляемой строки
    }
};

class TaskBag:public TEMPLET::TBag{
public:
    int cur;//номер текущей строки в матрице C
};
```

Пример кода построчного умножения матриц (2/3)

```
class TaskBag:public TEMPLET::TBag{
public:
    class TaskBagTask:public TBag::Task{
    public:
        // запись задачи в поток  (только для распределенной реализации)
        void send_task()  {send(&num,sizeof(num));}
        // чтение задачи из потока (только для распределенной реализации)
        void recv_task()  {recv(&num,sizeof(num));}
        // запись результата в поток (только для распределенной реализации)
        void send_result(){send(&num,sizeof(num)); send(c[num],sizeof(double)*N);}
        // чтение результата из потока (только для распределенной реализации)
        void recv_result(){recv(&num,sizeof(num)); recv(c[num],sizeof(double)*N);}
    }
};
```

Пример кода построчного умножения матриц (3/3)

```
class TaskBag:public TEMPLET::TBag{
public:
    // задачи есть, пока не все строки выданы на обработку
    bool if_job(){return cur<N;}
    // в данном примере результат распределённых вычислений
    // сохраняется в recv_result()
    // поэтому метод put() определять не требуется
    void put(Task*t){TaskBagTask* mt=(TaskBagTask*)t; }
    // записываем номер текущей строки в параметр задачи
    // увеличиваем номер строки
    void get(Task*t){TaskBagTask* mt=(TaskBagTask*)t; mt->num=cur++;}
    // вычисляем строку матрицы с номером num
    void proc(Task*t){TaskBagTask* mt=(TaskBagTask*)t;
        int i=mt->num;
        for(int j=0;j<N;j++){
            c[i][j]=0.0;
            for(int k=0;k<N;k++)c[i][j]+=a[i][k]*b[k][j];
        }
    }
};
```

Подробнее о Bag-of-tasks

- Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования. – М.: Вильямс, 2003. – 512 с.
- Шаблон Taskbag в системе управления проектами научных вычислений Templet Web http://templet.ssau.ru/templet/template/show/5

Пример 3: Язык разметки Templet

Основные сведения

- Парадигма программирования: язык координации
- Язык программирования: любой, текущая реализация – C++
- Язык исполнения: любой, текущая реализация – C++
- Среда исполнения: любая, согласованная с языком исполнения
  - текущая реализация – потоки C++11, Windows API, POSIX
  - Планируемая – MPI
- Типобезопасность: поддерживается через язык координации и реализации
- Поддержка вложенности: нет, используется горизонтальная, а не иерархическая композиция
- Список поддерживаемых шаблонов: Farm (Taskbag), Pipe (Pipeline), Ring, пользовательские

Особенности реализации

- Применение языка разметки для контроля структуры кода на исходном языке программирования
- Универсальная для всех шаблонов модель выполнения
- Передача параллельной семантики выполнения на исходном языке программирования

Разметка OpenMP vs Templet (1/2)

```
#pragma omp parallel for shared(a, b, c) private(i)
  for (i = 0; i < N; i++)
    c[i] = a[i] + b[i];
```

Разметка OpenMP vs Templet (2/2)

```
#include <runtime.h>                          <-- base-language
/*templet$$include*/                          <-- user-prefix
  #include <iostream>                         <-- base-language
/*end*/                                       <-- user-postfix
/*templet*                                    <-- scheme-prefix
  *hello<function>.                           <-- module scheme
*end*/                                        <-- scheme-postfix
void hello(){                                 <-- base-language
/*templet$hello$*/                            <-- user-prefix
  std::cout << "hello world!!!";              <-- base-language
/*end*/                                       <-- user-postfix
}                                             <-- base-language
```

Многопоточная модель выполнения (1/2)

```
struct thread{void(*tfunc)(thread*);};


struct mutex{};
void lock(mutex*);
void unlock(mutex*);


struct event{};
void wait(event*,mutex*);
void notify(event*);
```

Модель выполнения Templet (2/2)

```
struct chan{};
struct proc{void(*recv)(chan*,proc*);};
void send(chan*,proc*);
bool access(chan*,proc*);
```

Способ передачи семантики выполнения

```
  size_t rsize;
  // выполняем, пока в очереди есть каналы
  while(rsize=ready.size()){
    //случайным образом выбираем канал, выполняющий передачу
сообщения
    //извлекаем этот канал из очереди сообщений
    //изменяем его состояние на «нет передачи»
    int n=rand()%rsize; auto it=ready.begin()+n;
    BaseChannel*c=*it;ready.erase(it); c->sending=false;
    //из канала извлекаем процесс, в который выполняется
передача
    //запускаем метод обработки сообщения recv в этом канале
    //и передаём сам канал в качестве аргумента метода recv
    c->p->recv(c);
  }
```

Пример использования (1/2)

- Проверим выполнение тождества

  $Sin^2x+Cos^2x=1$

  для заданного x
- Применим шаблон Fork:

  выражения Sin^2x и Cos^2x вычислим параллельно

Пример использования (2/2)

Секция module-scheme:

```
~Link = +BEGIN ? ArgCos -> CALCCOS | ArgSin -> CALCSIN;
          CALCCOS ! Cos2 -> END; CALCSIN ! Sin2 -> END.
*Master =
        p1:Link ! Sin2 -> join; p2:Link ! Cos2 -> join;
       +fork(p1!ArgSin,p2!ArgCos); join(p1?Sin2,p2?Cos2).
*Worker =
        p : Link ? -> DO;
    DO:sin2(p?ArgSin,p!Sin2)|cos2(p?ArgCos,p!Cos2).
```

Подробнее о Templet

- С.В. Востокин, “Препроцессор языка Templet: инструмент программирования в терминах модели «процесс-сообщение»”, Вестн. Сам. гос. техн. ун-та. Сер. Физ.-мат. науки, 3(36) (2014), 169–182

  (http://dx.doi.org/10.14498/vsgtu1334)

- Шаблон Templet

  (http://templet.ssau.ru/templet/template/show/7)