Моделирование распределённых вычислений

Востокин Сергей Владимирович

План

- Обзор моделей распределённых вычислений
- Обзор параллельных и распределённых алгоритмов
- Формальные теории параллельных процессов. Пример: темпоральная логика действий Лампорта (TLA)
- Спецификация параллельных/распределённых алгоритмов. Пример: модель акторов в терминах темпоральной логики Лампорта
- Пример: алгоритм метода Гаусса-Зейделя, сравнение акторного и последовательно-параллельного параллельных алгоритмов
- Порядок выполнения самостоятельных заданий

Обзор моделей распределённых вычислений

Проблемы моделирования параллельных и распределённых вычислений

- Не очевидна связь между реализацией алгоритма и его свойствами – формальные методы служат для установления этой связи
- Методы, пригодные для последовательных вычислений, не подходят – моделирование функций не является адекватным методом, так как не учитывает взаимодействие между процессом и его окружением
- Не разработано единой фундаментальной концепции, аналогичной имеющейся для последовательных вычислений (Тезис Чёрча — Тьюринга)

Классификация формальных методов: подходы на основе синтеза и анализа

Методы синтеза (алгоритм строится на основе спецификации)

- Распараллеливание последовательных алгоритмов (динамические, для ациклических и циклических участков кода, для отдельных операторов программы)
- Методы, использующие непроцедурные спецификации
- Построении кода алгоритма совместно с доказательством корректности
- Эквивалентные алгебраические преобразования

Методы анализа (исследуется готовый алгоритм)

- Точные
  - Верификация (метод утверждений)
  - Спецификация (темпоральные логики)
- Имитационные
  - Стратегии планирования ресурсов
  - Дискретно событийные модели, управляемые событиями
  - Методы построения и анализа трасс алгоритма по его спецификации

Классификация формальных методов: понятие состояния и локальности

- В основе моделирования лежит понятие ***локальности***: действия алгоритмов изменяют лишь часть общего состояния системы
- Смена состояний может описываться последовательностью состояний или действий (порождаемых событиями)
- Вычислительный процесс может представляться либо множеством цепочек, либо деревом действий (событий)

Классификация формальных методов: подходы к описанию состояний

- ***Глобальное состояние*** – подход основан на возможности упорядочивания всех событий в системе из чего делается вывод о наблюдаемости глобального состояния
- ***Локальное состояние*** – подход основан на том, что при наличии отказов процессы не могут согласовать свои действия, процесс может только накапливать локальную информацию о системе в целом (за счёт поступающих сообщений)
  - Понятие локального состояния полезно при доказательстве невозможности построения алгоритмов по некоторым спецификациям при наличии отказов

Методы описания действий и их синхронизации

- Методы, использующие понятие потока управления
  - Последовательно-параллельный метод
  - С взаимодействием параллельных ветвей
    - Атомарные операции; процедурные методы; сообщения
- Асинхронные методы
  - С событийным управлением
  - Потоковые
  - Динамические
  - Сети Петри и их подклассы
- Другие: SPMD, MPMD, CSP (Occam)

Особенности моделей распределённых вычислений

- Обычно основаны на концепциях процессов и обмена сообщениями
  - Другие модели процессов относят к распределённым, если они могут быть описаны в терминах процессов и сообщений
- Содержат два ограничения:
  - Время работы алгоритма существенно зависит от времени передачи сообщений
  - Корректная работа любого процесса не зависит от отказов других процессов с которыми он взаимодействует

Классификация моделей процессов, обменивающихся сообщениями (1/3)

- Сетевая топология
  - Неориентированные / ориентированные коммуникационные графы
  - Связанность (граф полносвязанный или нет)
- Синхронность
  - Полностью асинхронная модель
  - Модель с таймерами
  - Модель с часами
  - Синхронная модель

Классификация моделей процессов, обменивающихся сообщениями (2/3)

- Виды отказов
  - Коммуникационные отказы
    - Потери сообщений
    - Отказ линка
    - Потеря связанности коммуникационного графа
  - Отказы процессов
    - Византийский отказ
    - Отказ-пропуск
    - Отказ-остановка

Классификация моделей процессов, обменивающихся сообщениями (3/3)

- Способы буферизации сообщений
  - Неограниченная буферизация
  - Ограниченная буферизация
    - С ожиданием процесса-отправителя
    - Со сбоем процесса-отправителя
  - FIFO буферизация

Другие модели процессов

- Глобальные разделяемые переменные (не отказоустойчивы)
- Локальные разделяемые переменные (чтение переменной из отказавшего процесса предполагает возврат специального значения)
- Модель Хоара (CSP) (не отказоустойчива)
- Удалённый вызов процедуры (RPC) и рандеву (предусматривают обработку ошибок)

Обзор параллельных и распределённых алгоритмов

Группы алгоритмов

- Алгоритмы, использующие разделяемые переменные
- Алгоритмы достижения соглашения
- Сетевые алгоритмы
- Алгоритмы управления параллельными базами данных

Алгоритмы с разделяемыми переменными (1/3)

- Задача о критической секции
  - Алгоритм Дейкстры (Деккера) – не учитывает эффект отталкивания процессов
  - Алгоритм кондитерской Лампорта – учитывает отталкивание
  - Алгоритмы, основанные на примитиве TEST-AND-SET
  - Стохастические алгоритмы
  - Алгоритмы, основанные на обмене сообщениями
  - С произвольным числом процессов в критической секции

Алгоритмы с разделяемыми переменными (2/3)

- Задача об обедающих философах
  - Алгоритм Дейкстры, использующий семафоры
  - Стохастические и другие алгоритмы
- Проблемы кооперации процессов типа «поставщик-потребитель»
  - Задача об ограниченном буфере
  - Маркированные графы – дают общее решение задачи

Алгоритмы с разделяемыми переменными (3/3)

- Задача о параллельном сборе мусора
- Задачи об отказоустойчивых кооперативных алгоритмах
- Задача писатели-читатели
  - удалось определить «универсальные» классы переменных, для которых можно построить алгоритмы, реализующие атомарную операцию чтения–записи
  - показано, что при некоторых условиях одновременное чтение-запись не позволяет реализовать примитив TEST-AND-SET

Алгоритмы достижения соглашения

- Задача о двух генералах
    - При наличии коммуникационных отказов в системе из двух процессов невозможно достичь соглашения
- Задачи выбора общего значения
    - Задача о византийских генералах
    - Задача об одновременном действии (задача о распределенных стрелках)
    - Задача о синхронизации часов
    - Задача о распределённой транзакции

Алгоритмы достижения соглашения (результаты)

- Найдены предельные значения количества ненадежных процессов и минимальное число циклов обмена сообщениями, при которых имеет решение задача выборе общего значения
- Показано, что стохастические алгоритмы позволяют сократить число циклов обмена сообщениями
- Найден общий способ преобразования алгоритма выбора значения в алгоритм об одновременном действии
- Установлено, что для асинхронных моделей задача выбора общего значения не разрешима даже при наличии одиночного отказа-остановки
- Стохастические алгоритмы решают проблему для полностью асинхронной модели с византийскими отказами, но завершаются с вероятностью, стремящейся к единице
- Для полностью асинхронной модели с византийскими можно получить решение в вещественных числах с заданной точностью

Алгоритмы достижения соглашения (результаты, продолжение)

- Задача о синхронизации часов неразрешима, если треть и более процессов подвержена византийским отказам
- Из невозможности решения задачи о двух генералах следует неразрешимость задачи о транзакции при условии потери сообщений
- Невозможно решение задачи о транзакции в асинхронных системах при наличии отказов процессов даже в отсутствии потери сообщений (окно отказа)
- трехфазный протокол решения задачи о транзакции, не имеющий окна отказа, предполагает надежную доставку сообщений и возможность обнаружения отказа процесса

Сетевые алгоритмы

- Алгоритмы определения оптимальных маршрутов передачи сообщений
  - Алгоритмы широковещательной передачи сообщения
- Алгоритмы выбора лидера
  - Разные подходы различаются: свойствами коммуникационных графов; начальной информацией процессов; синхронностью; применением случайных или детерминированные методов; использованием уникальных идентификаторов процессов
- Алгоритмы вычисления функции на сети процессов (например, вычисление среднего значения)
- Алгоритмы определения завершения вычислений и тупика

Сетевые алгоритмы (продолжение)

- Динамические алгоритмы
  - Алгоритм снимка глобального состояния
  - Алгоритмы-синхронизаторы (методы преобразования синхронных сетевых алгоритмов в асинхронные)
- Протоколы линка
  - Протокол ABP (Alternating Bit Protocol)

Управление параллельными базами данных

- Особенности
  - Цель алгоритмов – обеспечить неделимость и упорядоченность: результат нескольких одновременных транзакций должен быть таким, как если бы они выполнялись автономно в некотором порядке
  - Алгоритмы допускают возможность отмены результата транзакции
- Алгоритмы, использующие блокировки
- Алгоритмы, использующие отметки времени
- Алгоритмы вложенных транзакций

Формальные теории параллельных процессов. Пример: темпоральная логика действий Лампорта (TLA)

Вычислительный процесс как объект моделирования

- в качестве наблюдаемого поведения дискретной системы будем рассматривать множество историй её выполнения бесконечной длины σ вида

$$s_0 \xrightarrow{A_1} s_1 \xrightarrow{A_2} s_2 \xrightarrow{A_3} s_3 \xrightarrow{A_4} \ldots$$

Определение модели

- Модель F – это формула, смысл которой определен в терминах историй бесконечной длины, однозначно определяющая заданное множество историй
- Интерпретация формулы F — это функция $[[\cdot]]$, которая ставит в соответствие формуле F истинное или ложное высказывание об историях вычислительного процесса

$$\langle s_0, s_1, s_2, \ldots \rangle [[F]] \equiv \sigma[[F]] : \mathrm{St}^{\infty} \rightarrow \{И, Л\}$$

Определение функции интерпретации модели

- Функция интерпретации строится на основе понятия **состояния:** функции, отображающей множество переменных Var на множество значений переменных Val

$$St : Var \to Val$$

- Таким образом, смысл переменных в формуле состоит в представлении значений в некотором состоянии вычислений

Функции состояния

- Математические выражения, построенные из переменных, знаков математических операций, знаков отношений, логических связок, кванторов и т.п. называются ***функциями состояния*** и интерпретируются согласно выражению:

$$s[[f]] \stackrel{def}{=\!=} f(\forall' v' : s[[v]] / v)$$

- Частный случай функции состояния – ***предикат состояния***

Действие

- Действие — это выражение, аналогичное функции состояния по своей структуре, которое дополнительно включает переменные с символом апострофа, принимает значения истина или ложь и интерпретируется в виде

$$s[[A]]t \overset{def}{=\!=} A(\forall ' v' : s[[v]]/v, t[[v]]/v')$$

- Формула для действия дает ответ на вопрос: могут ли два состояния **s** и ***t*** быть соседними в некоторой истории

Расширение интерпретации F для историй бесконечной длины

- Для описания историй бесконечной длины с использованием действий вводится специальный модальный оператор необходимости, который читается «всегда»

$$\sigma[[\Box F]] \overset{def}{=\!=} \forall n \in \mathrm{N} : \langle s_n, s_{n+1}, s_{n+2}, \ldots \rangle [[F]]$$

- Где ***F*** – предикат состояния или действие

Расширение интерпретации *F* для историй бесконечной длины (продолжение)

- Смысл функций состояния и действий расширяют на бесконечные истории следующим образом

$$\langle s_0, s_1, s_2, \ldots \rangle [[A]] \stackrel{def}{==} s_0[[A]]s_1,$$

$$\langle s_0, s_1, s_2, \ldots \rangle [[P]] \equiv s_0[[P]]$$

- Тогда

$$\langle s_0, s_1, s_2, \ldots \rangle [[\Box A]] \stackrel{def}{==}$$

$$\forall n \in \mathrm{N} : \langle s_n, s_{n+1}, s_{n+2}, \ldots \rangle [[A]] \equiv \forall n \in \mathrm{N} : s_n[[A]]s_{n+1}$$

$$\langle s_0, s_1, s_2, \ldots \rangle [[\Box P]] \equiv \forall n \in \mathrm{N} : s_n[[P]]$$

Интерпретация произвольной формулы

- Сложные формулы строятся путем комбинирования предикатов состояний и действий с использованием логических связок и кванторов. Для передачи смысла таких формул достаточно определить семантику логической основы формулы для произвольной системы базисных функций, например так:

$$\sigma[[\neg F]] \stackrel{def}{=} \neg \sigma[[F]]$$

$$\sigma[[F \wedge G]] \stackrel{def}{=} \sigma[[F]] \wedge \sigma[[G]]$$

Специализация модели (1/2)

- Пусть процесс X периодически увеличивает переменную x

  [x=0]→[x=1]→[x=2]→[x=3]→[x=4]→[x=5]→[x=6]→[x=7]...

- Пусть процесс Y периодически увеличивает переменную y

  [y=0]→[y=1]→[y=2]→[y=3]→[y=4]→[y=5]→[y=6]→[y=7]...

- Рассмотрим композицию систем X и Y. Так как системы независимы, то допустимы истории, где x изменяется, а y – нет. Например, такая

  [x=0,y=0]→[x=1,y=0]→[x=2,y=1]→[x=2,y=2]→[x=2,y=3]...

Специализация модели (2/2)

- Композицию систем X и Y естественно рассматривать как конъюнкцию формул: $X \wedge Y$
- История системы должна быть также историей для входящих в систему подсистем: $X \wedge Y \rightarrow X$ и $X \wedge Y \rightarrow Y$
- Этого можно добиться, если процесс описывается множеством историй, инвариантных по отношению к преобразованию дублирования одного состояния в соседних отсчетах времени (stuttering)
- Для этого вводятся дополнительные синтаксические ограничения на структуру темпоральных формул

Общий вид формул, инвариантных к stuttering-преобразованию

$$\Phi \stackrel{def}{==} I \wedge \Box[N]_f \wedge F$$

F — формулы, обозначающие справедливое в слабом (*WF*) или сильном смысле (*SF*) выполнение действий из *N* (поясняется далее)

Пояснения оператора stuttering преобразования

$$f \stackrel{def}{=\!=} (v_1, v_2, v_3, \ldots) \qquad f' \equiv (v_1, v_2, v_3, \ldots)' \equiv (v_1', v_2', v_3', \ldots)$$

$$unchanged(f) \stackrel{def}{=\!=} f = f'$$

$$[A]_f \stackrel{def}{=\!=} A \vee (f = f') \equiv A \vee unchanged(f)$$

$$\langle A \rangle_f \stackrel{def}{=\!=} A \wedge (f \neq f') \equiv A \wedge \neg unchanged(f)$$

Выражение понятия «справедливого выполнения» в TLA

Если допустить инвариантность относительно stuttering преобразования, то возможны истории, в которых вычисления произвольно останавливаются

Поэтому в общую формулу добавляются части *WF* и *SF*, оговаривающие отсутствие самопроизвольной остановки

$$WF_f(A) \stackrel{def}{==} \left(\Box\Diamond\langle A\rangle_f\right) \vee \left(\Box\Diamond\neg Enabled\langle A\rangle_f\right)$$

$$SF_f(A) \stackrel{def}{==} \left(\Box\Diamond\langle A\rangle_f\right) \vee \left(\Diamond\Box\neg Enabled\langle A\rangle_f\right)$$

Пояснения операторов в формулах «справедливого выполнения» в TLA

Оператор возможности:

$$\Diamond F \overset{def}{=\!=} \neg \Box \neg F$$

$$\langle s_0, s_1, s_2, \ldots \rangle [[\Diamond F]] \equiv \exists n \in \mathrm{N} : \langle s_n, s_{n+1}, s_{n+2}, \ldots \rangle [[F]]$$

Комбинации операторов необходимости и возможности:

$$\langle s_0, s_1, s_2, \ldots \rangle [[\Box \Diamond A]] \equiv \forall n \in \mathrm{N} : \exists m \in \mathrm{N} : s_{n+m}[[A]]s_{n+m+1}$$

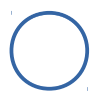
$$\langle s_0, s_1, s_2, \ldots \rangle [[\Diamond \Box A]] \equiv \exists n \in \mathrm{N} : \forall m \in \mathrm{N} : s_{n+m}[[A]]s_{n+m+1}$$

Оператор Enabled

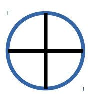
$$s[[Enabled\ A]] \overset{def}{=\!=} \exists t \in \mathrm{St} : s[[A]]t$$

Спецификация параллельных и распределённых алгоритмов. Пример: модель акторов в терминах темпоральной логики Лампорта

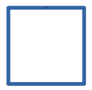
Графические условные обозначения акторной модели Templet



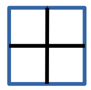
Актор



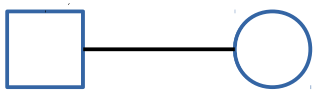
Актор обрабатывает сообщение



Сообщение

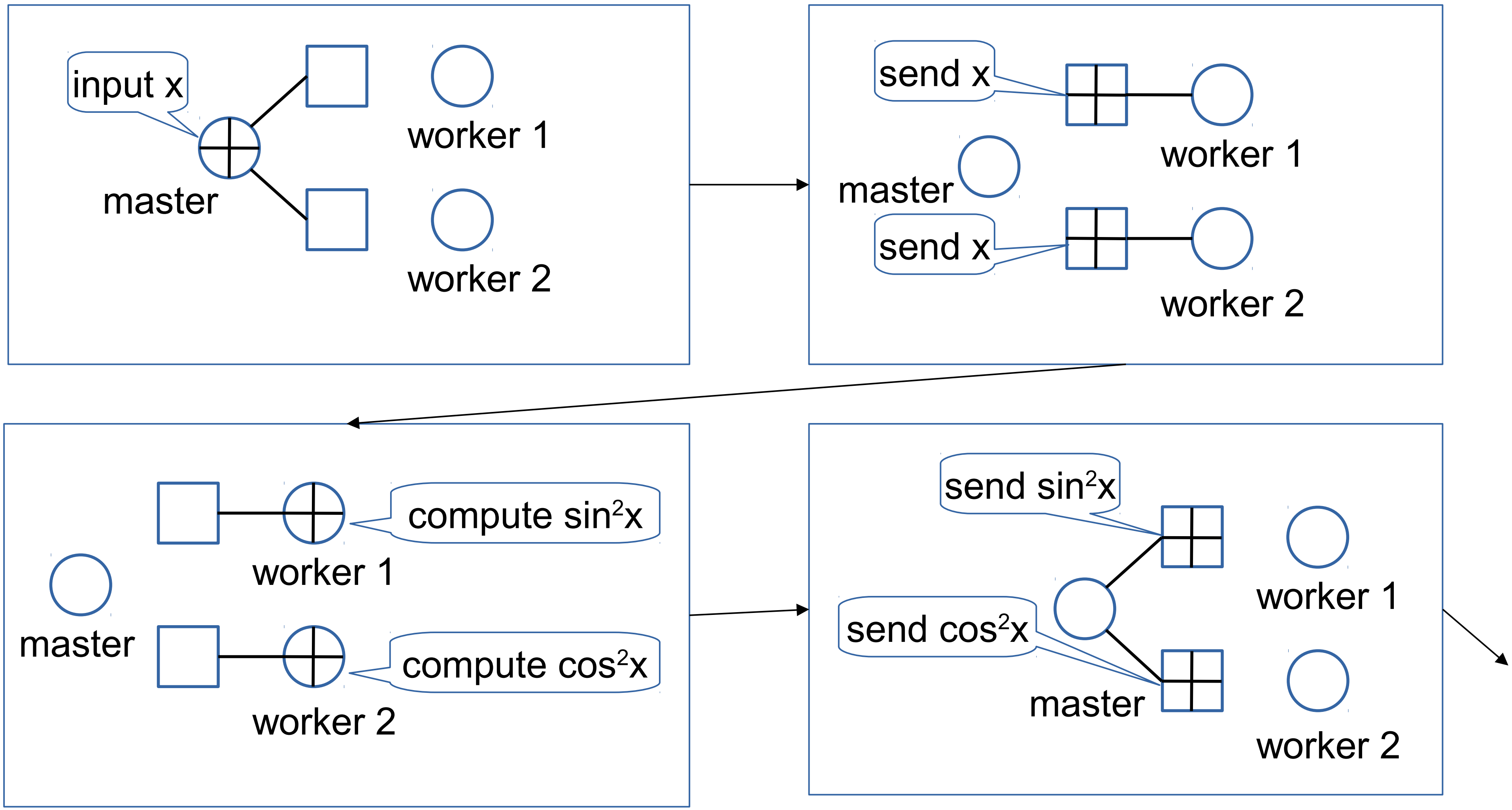


Сообщение доставляется актору

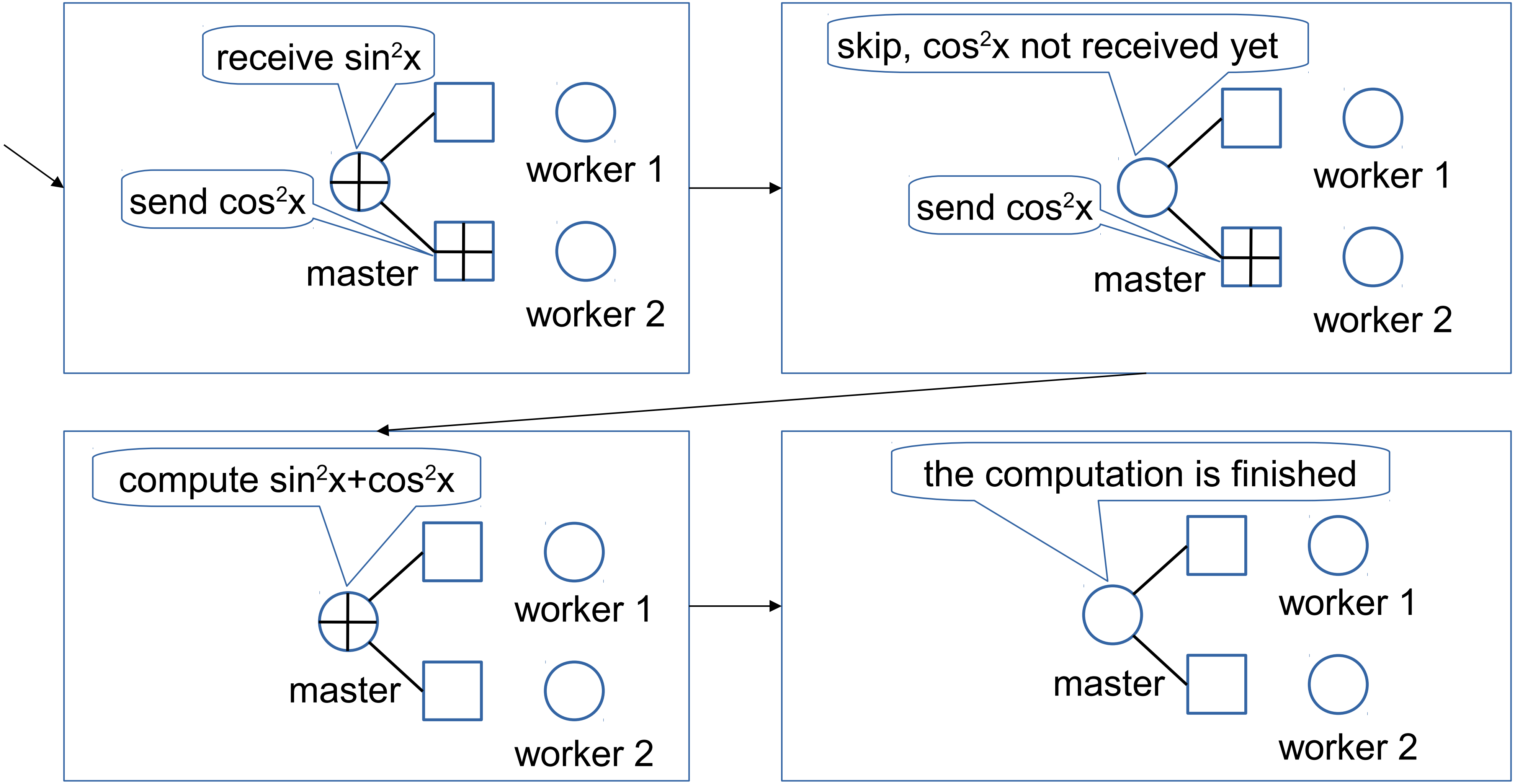


Сообщение всегда ассоциировано с актором,
в который оно доставляется либо доставлено

Пример использования условных обозначений: проверка тождества $\sin^2x+\cos^2x=1$



Пример использования условных обозначений: проверка тождества $\sin^2 x + \cos^2 x = 1$ (продолжение)



Текстовые условные обозначения модели Templet

$a \in \mathbb{N}$ – идентификатор актора (темпоральная константа)

$m \in \mathbb{N}$ – идентификатор сообщения (темпоральная константа)

$p[.]:\mathbb{N} \to \{0,1\}$ – массив переменных, определяющих активность актора, указанного в качестве аргумента (массив темпоральных переменных)

$s[.]:\mathbb{N} \to \{0,1\}$ – массив переменных, определяющих активность сообщения, указанного в качестве аргумента (массив темпоральных переменных)

$r[.]:\mathbb{N} \to \mathbb{N}$ – массив переменных, определяющих получателя сообщения, указанного в качестве аргумента (массив темпоральных переменных)

Текстовые условные обозначения модели Templet (продолжение)

$\boxed{A_1 \equiv \exists\, !m : \neg p[a] \wedge r[m] = a \wedge s[m] \wedge p'[a] \wedge r'[k] = a \wedge \neg s'[m]}$ – поступающие в актор сообщения обрабатываются последовательно

$\boxed{A_2 \equiv p[a] \wedge \neg p'[a]}$ – актор заканчивает обработку

$\boxed{A_3 \equiv \exists\, a : p[a] \wedge r[m] = a \wedge \neg s[m] \wedge s'[m]}$ – работающий актор отправляет сообщения

$\boxed{A_4 \equiv s[m] \wedge \neg s'[m]}$ – сообщение доставляется

Текстовые условные обозначения модели Templet (продолжение)

$\boxed{f_1 \equiv p[a]}$ – состояние акторов

$\boxed{f_2 \equiv (s[m], r[m])}$ – состояние сообщений

$\boxed{I \equiv \exists a : p[a] \vee \exists m : s[m]}$ – начальное состояние

$\boxed{\Phi \equiv I \wedge \Box [A_1 \vee A_2]_{f_1} \wedge \Box [A_3 \vee A_4]_{f_2} \wedge WF_{f_1}(A_2) \wedge WF_{f_2}(A_4)}$ – итоговая модель вычислений

Текстовые условные обозначения модели Templet (продолжение)

$$recv_{(call)}(a,m) \equiv \neg p[a] \wedge r[m]=a \wedge s[m] \wedge p'[a] \wedge r'[k]=a \wedge \neg s'[m]$$

$$recv(a)_{(return)} \equiv p[a] \wedge \neg p'[a]$$

$$access(a,m) \equiv r[m]=a \wedge \neg s[m]$$

$$send(a,m) \equiv r'[m]=a \wedge s'[m]$$

Пример: алгоритм метода Гаусса-Зейделя, сравнение акторного и последовательно-параллельного параллельных алгоритмов

Последовательный алгоритм

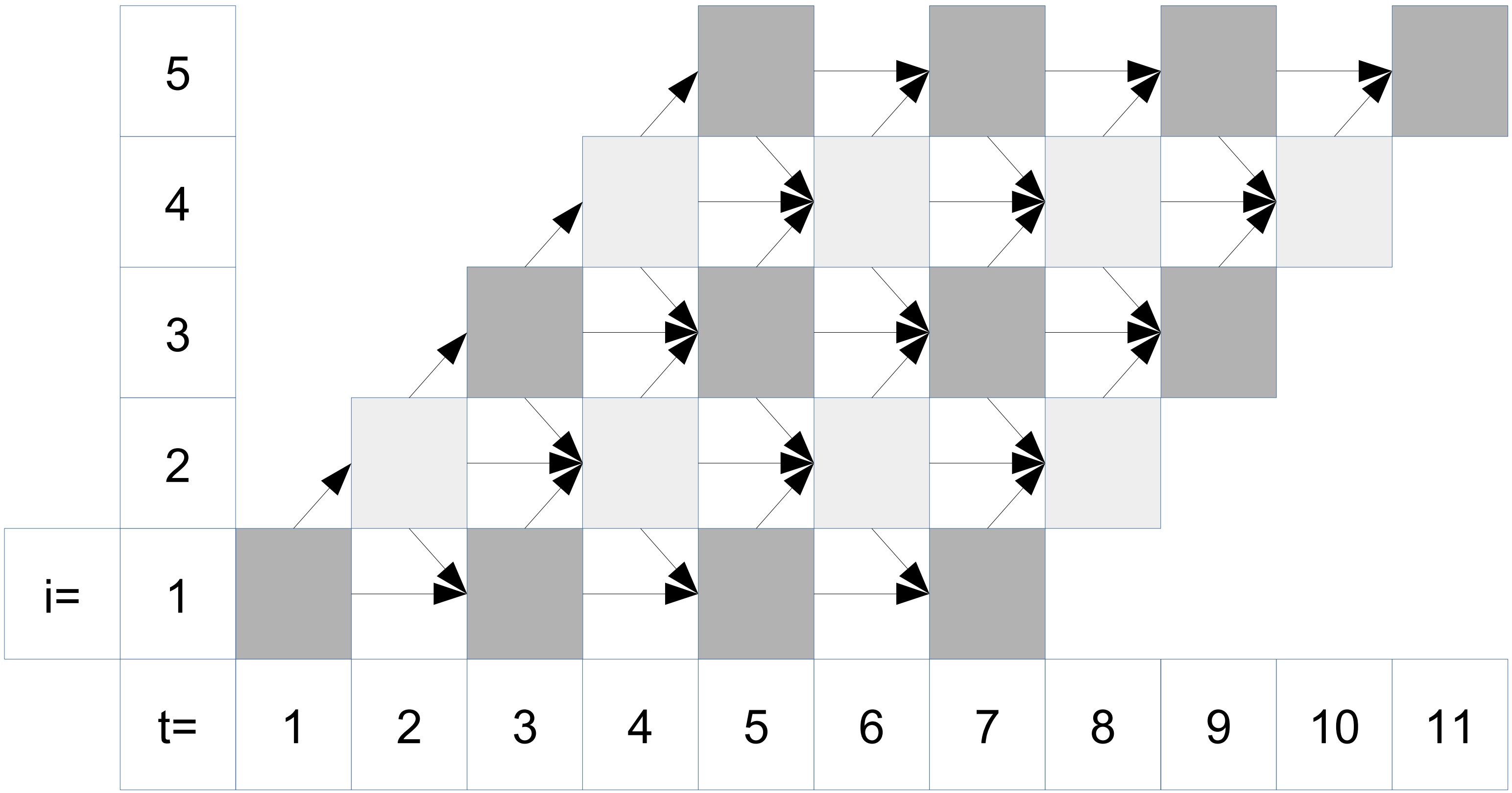
```
void op(int i){
 for (int j=1; j<W-1; j++)
   u[i][j]=(u[i][j-1]+u[i][j+1]+
     u[i-1][j]+u[i+1][j])*0.25;
}


for (int t=1;t<=T;t++)
 for (int i=1;i<H-1;i++) op(i);
```

Параллельно-последовательный алгоритм

```
for (int t=1; t<=(2*T-1)+(H-3); t++){
   if(t%2==1){
      parallel_for(int i=1;i<H-1;i+=2)
         if(i<=t && i>t-2*T)  op(i);
   }
   if(t%2==0){
      parallel_for(int i=2;i<H-1;i+=2)
         if(i<=t && i>t-2*T)  op(i);
   }
}
```

Иллюстрация вычислений по параллельно-последовательному алгоритму



Акторный алгоритм

```
const int N=H-2;
actor a[N]; message m[N-1]; int t[N]={1};

for (int i=0; i<N-1; i++){
   m[i].r=i; m[i].s=(i==0)?1:0;
}

for(int i=0; i<N; i++)a[i].p=0;

void recv(int i){
 if ((i==0 || access(a[i], m[i-1])) &&
    (i==N-1 || access(a[i], m[i])) &&
    (t[i]<=T)){
       op(i+1);t[i]++;
       if (i!=0) send(a[i-1],m[i-1]);
       if (i!=N-1) send(a[i+1],m[i]);
 }
}
```

Иллюстрация вычислений по параллельно-последовательному алгоритму на двух процессорах

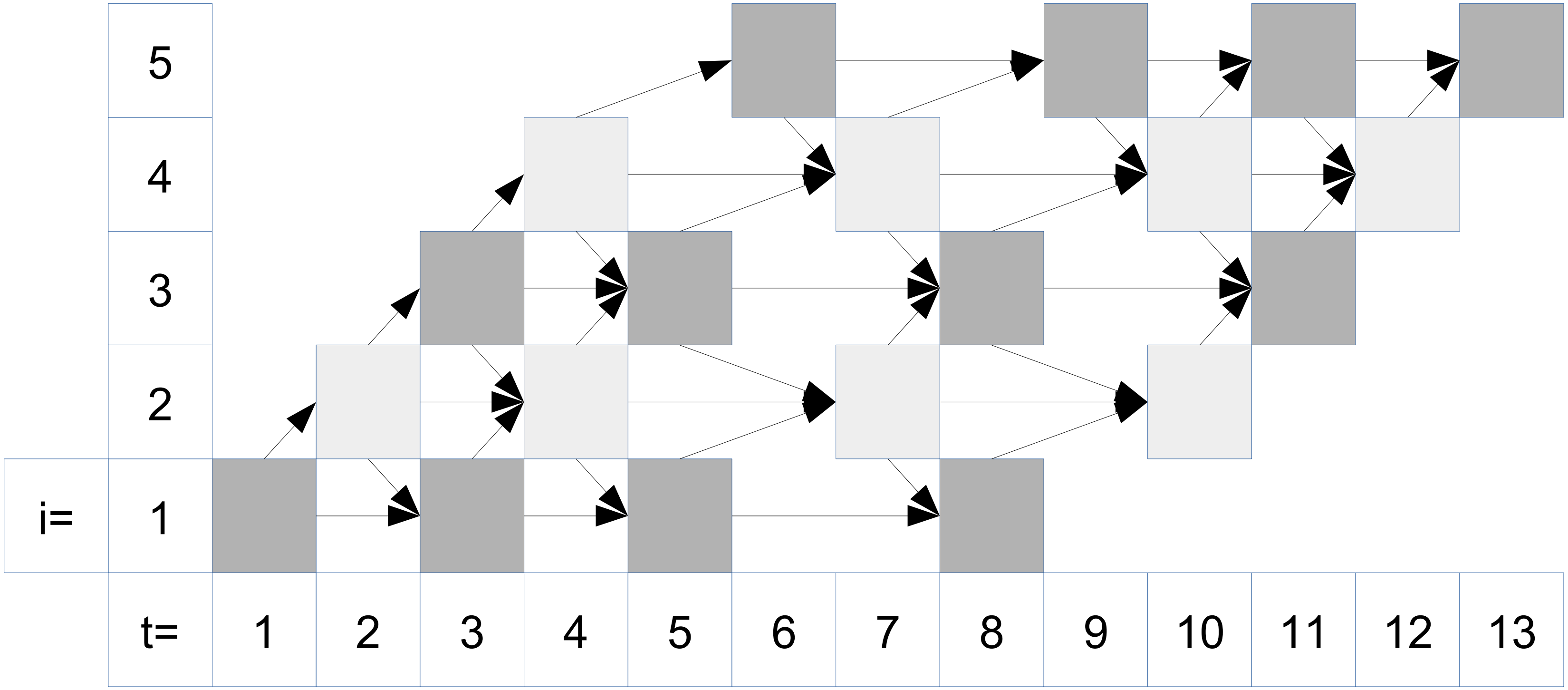
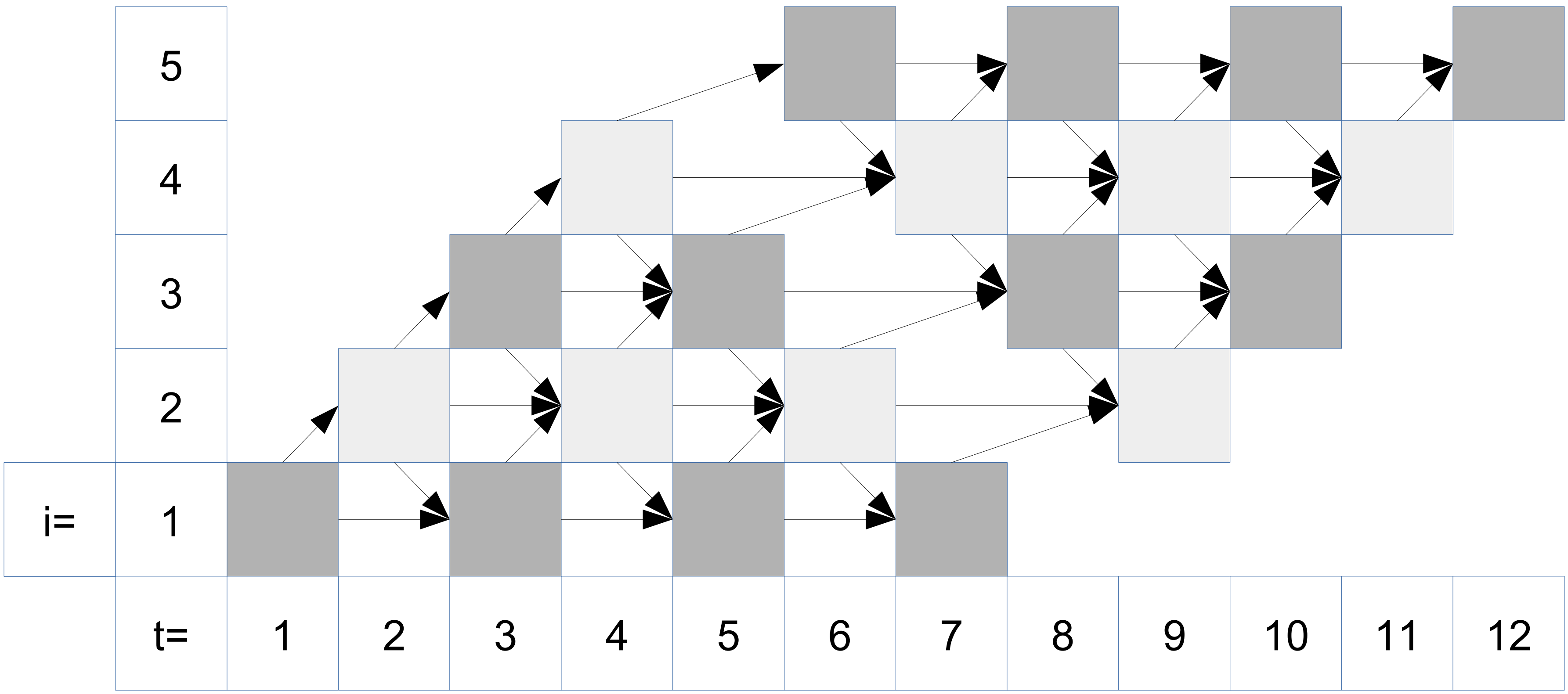


Иллюстрация вычислений по акторному алгоритму на двух процессорах



Порядок выполнения самостоятельных заданий

Порядок выполнения заданий

1. Выполнить словесное описание алгоритма в терминах приёма и отправки сообщений. Для наглядности можно использовать поясняющие диаграммы.

2. Выполнить точное описание алгоритма из п.1 в терминах акторной модели:
- определить необходимые типы акторов, их состояние, алгоритмы обработки поступающих сообщений;
- определить необходимые типы сообщений;
- определить начальное состояние вычислений (имеющиеся вначале работы алгоритма акторы; отправленные, но не принятые начальные сообщения);
- определить критерий остановки (достижения требуемого состояния системой акторов).

3. Построить программную модель алгоритма п.2 с использованием каркаса Templet

Каркас Templet

```
struct engine; struct proc; struct chan;

struct engine{std::vector<chan*> ready;};
struct proc{ virtual void recv(chan*);};
struct chan{ proc*p; bool sending;};

inline void send(engine*e, chan*c, proc*p)
{
    if (c->sending) return; c->sending = true; c->p = p;   e->ready.push_back(c);
}

inline bool access(chan*c, proc*p)
{
    return c->p == p && !c->sending;
}

inline void run(engine*e, int n = 1)
{
    size_t rsize;
    while (rsize = e->ready.size()){
        int n = rand() % rsize;  auto it = e->ready.begin() + n;
        chan*c = *it;    e->ready.erase(it); c->sending = false;
        c->p->recv(c);
    }
}
```

Порядок выполнения заданий (продолжение)

Определите акторы реализации алгоритма п.2 как расширения `struct proc`, а сообщения – как расширения `struct chan`.

4. Подготовить тестирующий код. Выполнить тестирование программной реализации алгоритма п.3.

5. Оформить письменный отчет.

Примеры заданий

1. Алгоритм (по выбору) использующий распределённый портфель задач.
2. Распределённое умножение матриц с использованием циркуляции столбцов.
3. Нахождение простых чисел методом решета Эратосфена, реализованного в виде конвейера из фильтрующих процессов.
4. Вычисление простых чисел в архитектуре управляющий-рабочие (см. Г. Эндрюс с.266).
5. Сортировка массива из n чисел при помощи конвейера из n процессов. Каждый из процессов оперирует 2 числами: следующим введенным и текущим минимумом.
6. Реализовать алгоритм сортировки слиянием, для этого реализовать актор merge, объединяющий два отсортированных потока чисел в один отсортированный поток.
7. Кольцевой алгоритм голосования. Считать, что отказавший процесс пропускает через себя сообщение ГОЛОСОВАНИЕ, не добавляя себя к списку претендентов; процессы, запускающие голосование выбираются произвольно.
8. Алгоритм забияки. Считать, что отказавший процесс посылает в ответ на сообщение ГОЛОСОВАНИЕ специальное сообщение ОТКАЗ; процессы, запускающие голосование выбираются произвольно.
9. Двухфазный протокол подтверждения транзакции.
10. Трёхфазный протокол подтверждения транзакции.