Модели разработки конкурентных и параллельных программ

Цифровые ресурсы в научном исследовании

Шаповалова Ирина

Группа 3540904/10202

Актуальность

- Использование преимуществ многоядерной архитектуры
- Повышение производительности
- Обеспечение высокой отзывчивости, надежности и эффективности ПО

Конкуренция и параллелизм

- **Конкуренция** (concurrency) – способ одновременного решения множества разных задач
- **Параллелизм** (parallel) – способ одновременного выполнения разных частей одной программы

1. Потоки выполнения и блокировки

Потоки выполнения и блокировки

Средства:

- Взаимоисключения (mutual exclusion, мьютексы)
- Критические секции
- Семафоры
- События
- Условные переменные

Проблемы:

- Состояние гонки
- Взаимоблокировки
- Видимость памяти

Потоки выполнения и блокировки

Сильные стороны:

- Широкая применимость
- Лежит в основе многих других моделей
- Является лишь формализацией аппаратной части компьютера
- Легко интегрируется с большинством языков программирования

Слабые стороны:

- Не имеет прямого отношения к параллелизму
- Поддерживаются в основном только в архитектурах с разделяемой памятью
- Большая сложность в использовании
- Сложное тестирование

2. Функциональное программирование

Функциональное программирование

- Функциональные программы **лишены изменяемого состояния**
- Более декларативны
- **Все функции** являются **ссылочно-прозрачными** – где бы ни вызывалась функция, можно заменить вызов его результатом
- **Вычисления** могут протекать независимо, а значит можно выполнять их в **любом порядке**
- Можно легко параллелить код

Функциональное программирование

Сильные стороны

- Легко поддается распараллеливанию
- Легко поддается организации конкуретного выполнения задач
- Не имеет изменяемого состояния (отсутствует большинство проблем конкуренции)

Слабые стороны

- Некоторые решения могут быть менее эффективными и вести к снижению производительности

3. Разделение идентичности и состояния – язык Clojure

Разделение идентичности и состояния – язык Clojure

- Соединяет **возможности функционального программирования** с **изменяемым состоянием**.
- Вводится понятие **сохраненных структур**, которые *разделяют идентичность и состояние*. Если получить текущее состояние, связанное с идентичностью, то само по себе состояние будет неизменяемым, независимо от того, что произойдет с идентичностью

Разделение идентичности и состояния – язык Clojure

Сильные стороны

- Поддержка приемов ФП
- Сохраненные структуры отделяют идентичность от состояния

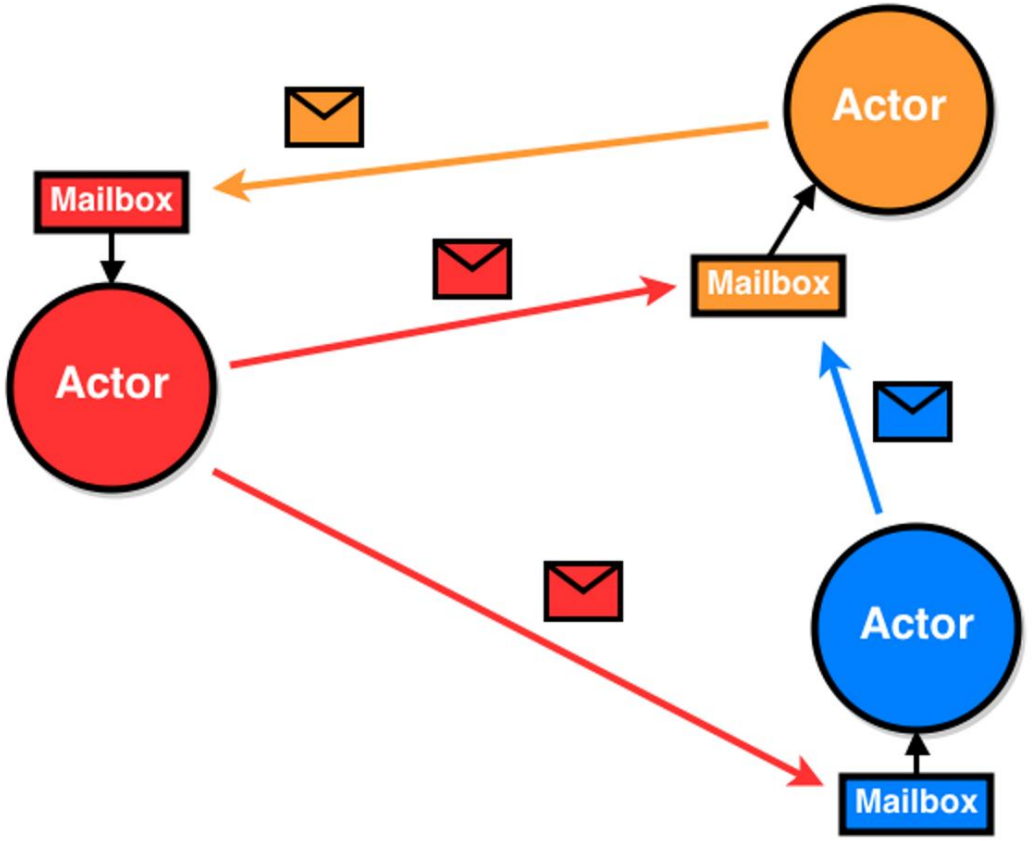
Слабые стороны

- Отсутствие поддержки распределенного программирования

4. Акторы

Акторы

- Программирование с акторами **сохраняет изменяемое состояние**, но *исключает его совместное использование*.
- Акторы **инкапсулируют в себе состояние и методы взаимодействия** с другими акторами посредством обмена сообщениями
- Поддержка **распределенных** приложений



Акторы

- Используется асинхронная передача сообщений. Прежде чем попасть в актор, сообщения сначала попадают в почтовый ящик.
- Акторы выполняются конкуретно по отношению к другим акторам, но обработка сообщений выполняется последовательно, в порядке поступления их в почтовый ящик

Акторы

Сильные стороны

- Обмен сообщениями и инкапсуляция
  - Не имеют совместно используемого состояния, выполняются конкуретно с другими акторами, сами акторы действуют последовательно
  - Акторы легко тестировать по отдельности
- Отказоустойчивость
- Распределенные приложения (поддержка разделяемой и распределенной памяти)

Слабые стороны

- Все еще восприимчивы к проблемам взаимоблокировок
- Проблема переполнения почтовых ящиков
- Не имеет прямой поддержки параллелизма

5. Взаимодействие последовательных процессов (communicating sequential processes, CSP)

Взаимодействие последовательных процессов (CSP)

- Модель *похожа на акторы* – независимые сущности, выполняющиеся конкуретно и взаимодействующие посредством обмена сообщений
- Отличие – **в центре** модели находятся **каналы**, по которым осуществляется обмен.

Взаимодействие последовательных процессов (CSP)

Сильные стороны:

- Гибкость каналов, являющихся независимыми сущностями
- Использование приема инверсии управления упрощает модель и повышает эффективность

Слабые стороны:

- Нет проработки вопросов отказоустойчивости и распределенного программирования
- Все еще подвержена взаимоблокировкам
- Не имеет прямой поддержки параллелизма

Языки, хорошо поддерживающие конкуретность

- Erlang/Elixir (акторы)
- Haskell
- Go (CSP)
- Scala (акторы)
- Clojure

Специализированные решения

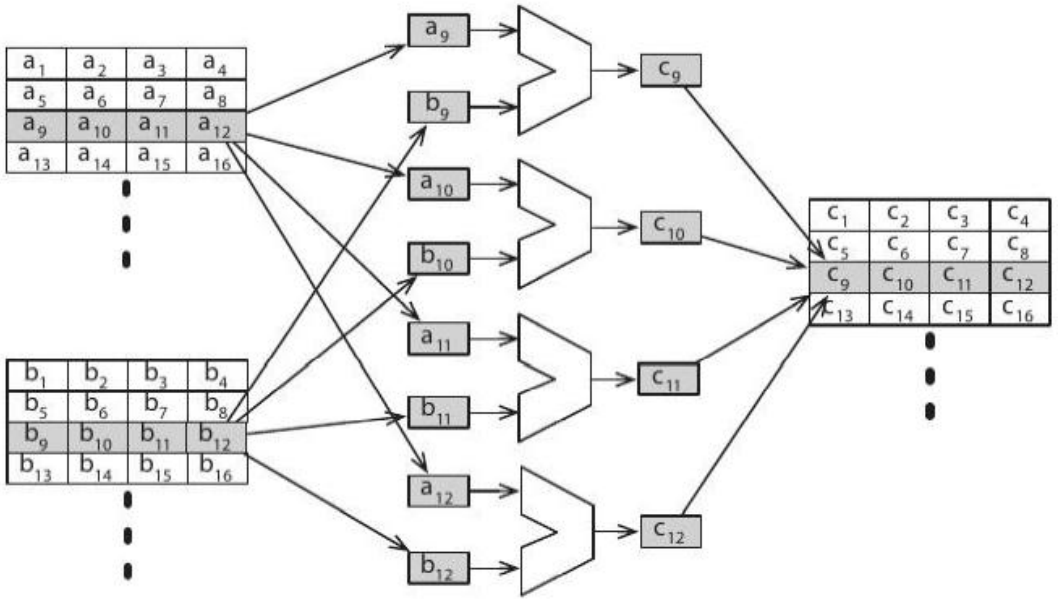
Параллелизм данных

GPU – мощный графический процессор, поддерживающий параллельную обработку данных.

GPGPU – программирование универсальных вычисление на GPU (general purpose computing on GPU).

Параллелизм можно реализовать множеством разных способов. Базовые:

- Конвейерная обработка
- Применение множества ALU



Фреймворки – CUDA, DirectCompute, OpenCL

Параллелизм данных

Сильные стороны

- Идеально походит для ситуаций, когда требуется обрабатывать огромные объемы числовых данных
- Графические процессоры обладают мощными возможностями параллельной обработки данных
- Графические процессоры эффективны с точки зрения потребления электроэнергии

Слабые стороны

- Набор инструментов специализирован именно для числовых вычислений
- Эффективность оптимизации нередко зависит от понимания архитектурных особенностей аппаратуры
- Проблемы с высокой производительностью при кроссплатформенных решениях

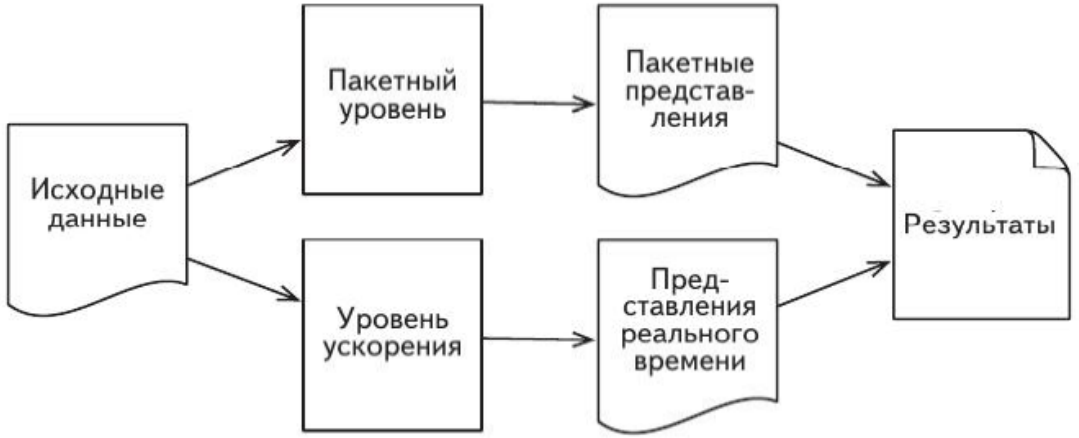
Лямбда-архитектура

Лямбда-архитектура является одним из подходов к **обработке больших объёмов данных** и опирается на

- параллельную обработку данных
- масштабирование

На пакетном уровне используются пакетные технологии (Hadoop, Spark). На уровне ускорения – потоковые технологии (Storm)

Целесообразно применять только при действительно больших объемах данных.



Спасибо за внимание!