Параллельное программирование: Введение

Учебный курс

Кензин Максим Юрьевич
Кафедра Информационных технологий
ИМИТ ИГУ / Институт динамики
систем и теории управления СО РАН

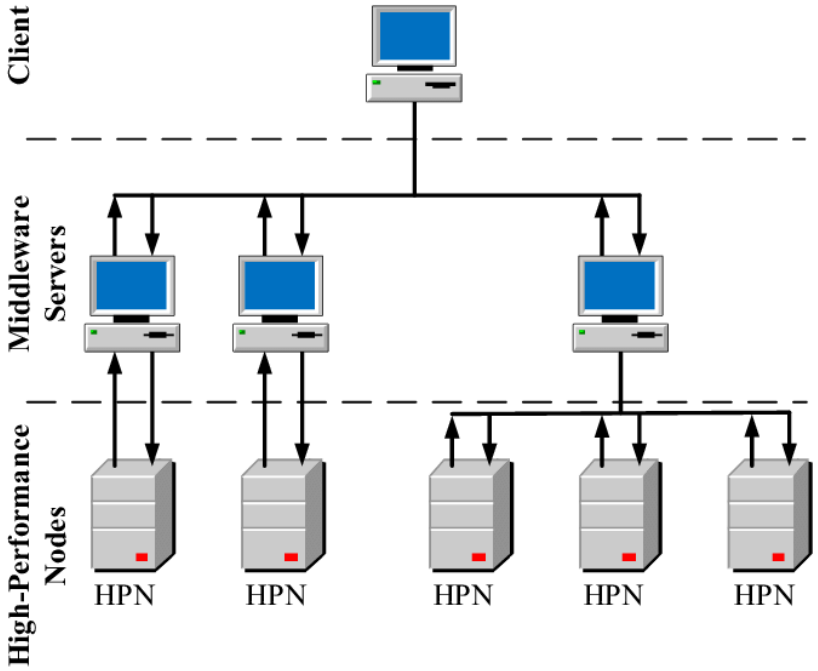
ИМИТ ИГУ, 2022

Параллельное программирование

Когда-то процессоры были одноядерные (до 2001 года), а в системах был только один процессор (до 1966 года). А затем инструкции стали выполняться одновременно в рамках одной системы.

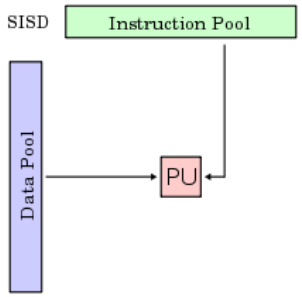
Сейчас:

- Многопроцессорные системы;
- Кластеры;
- GRID’ы;
- Cloud Computing.

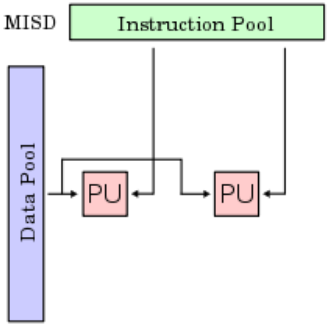


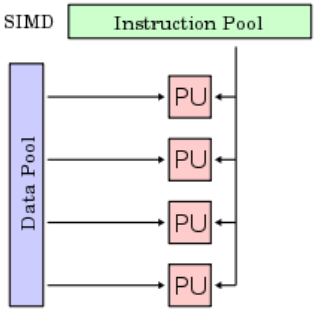
Классификация Флинна

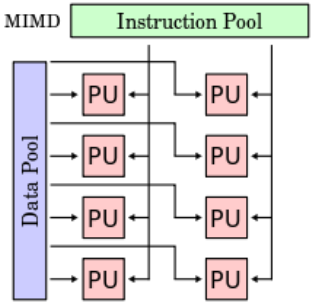
Общая классификация архитектур ЭВМ по признакам наличия параллелизма в потоках команд и данных.



	Одиночный поток команд (single instruction)	Множество потоков команд (multiple instruction)
Одиночный поток данных (single data)	SISD (Фон Нейман)	MISD (Конвейер-ные ЭВМ)
Множество потоков данных (multiple data)*	SIMD (Векторные ЭВМ)	MIMD (Много-процессор.)

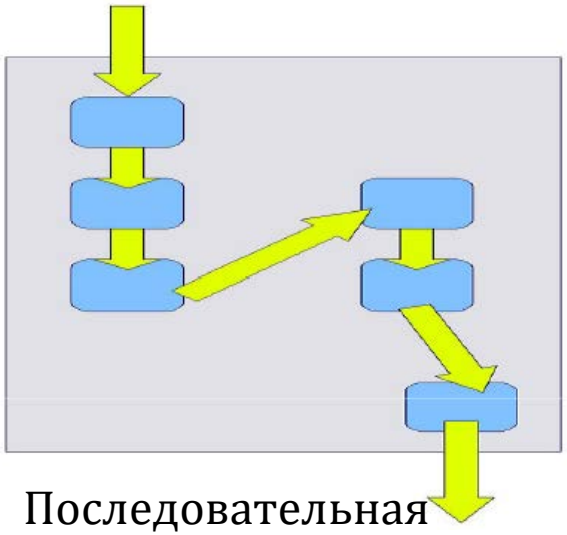




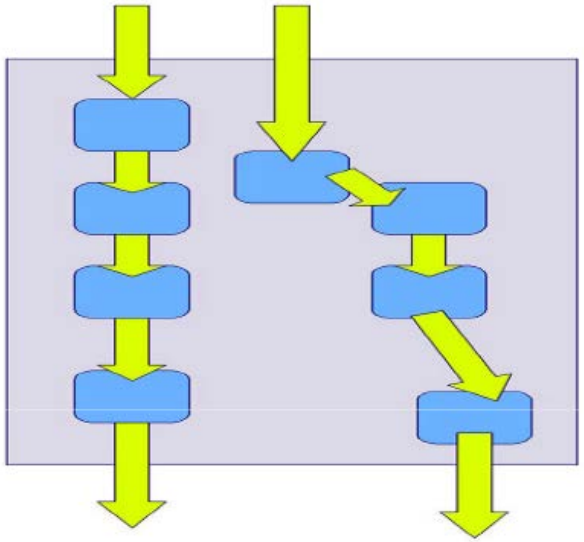


*Общая память vs Распределенная память

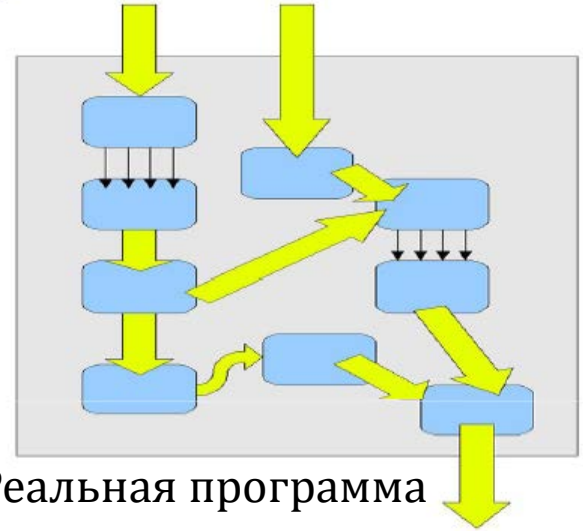
Данные внутри программы



Последовательная программа



Параллельная программа



Реальная программа

Характеристики моделей

1. Последовательная модель:
- Невысокая производительность;
- Стандартные ЯП;
- Хорошая переносимость.

2. Параллельная модель:
- Возможность добиться более высокой производительности;
- Применение специализированных инструментов программирования;
- Повышенная трудоемкость программирования;
- Проблемы с переносимостью программ.

Требования к параллельным программам:
- Хорошая масштабируемость;
- Детерминизм;
- Эффективность.

Два вида параллелизма

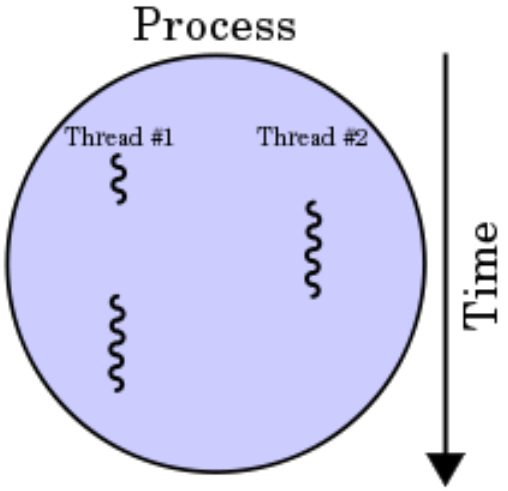
Конкурентность (Concurrency) – объединение независимо выполняемых элементов кода (функций). Несколько задач ⬅ один процессор.

Параллелизм (Parallelism) – одновременно выполняемые (преимущественно очень похожие) элементы кода (функции). Одна большая задача ➡ несколько процессоров ➡ ускорение.

Элементы кода (функции) ⋙ Потоки.

Поток (англ. thread, нить)

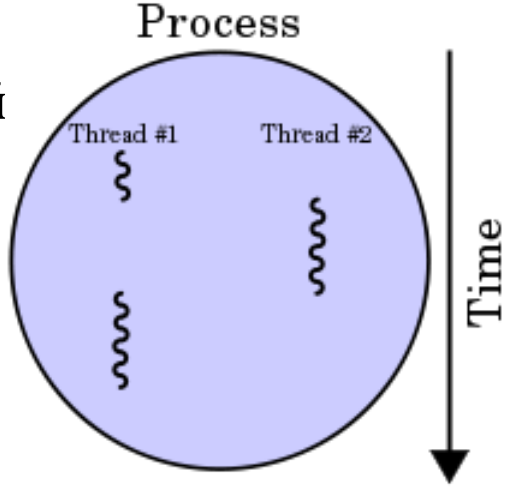
- Потоки — абстракция операционной системы, позволяющая выполнять некие куски кода параллельно.
- Поток — это средство эмуляции параллельного исполнения относительно других потоков.
- Поток — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы.
- Поток — последовательный поток управления (последовательность команд) в рамках одной подпрограммы.



Поток (англ. thread, нить)

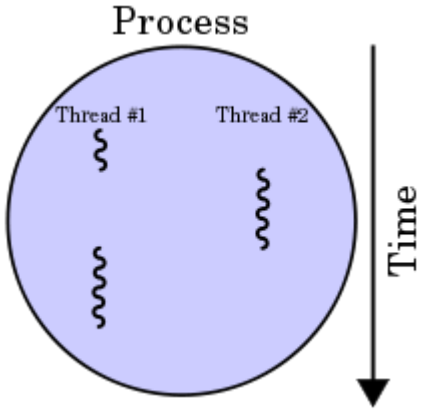
Свойства:

- Поток выполнения находится внутри процесса;
- Потоки могут сосуществовать в рамках одного и того же процесса, используя совместные ресурсы;
- Потоки регулируются операционной системой (планирование и диспетчеризация потоков);
- В процессор все потоки попадают уже как единый последовательный набор команд;
- Каждый поток имеет «контекст», в который входит кусок оперативной памяти под стек и копию регистров процессора;
- Потоки могут иметь значение приоритета, позволяющий работать дольше/чаще;
- Потоки можно погружать в сон для освобождения ресурсов под другие потоки.



Поток выполнения

На одном процессоре многопоточность обычно происходит путём *временного мультиплексирования* (как и в случае многозадачности): процессор переключается между разными потоками выполнения.



Это переключение контекста обычно происходит достаточно часто, чтобы пользователь воспринимал выполнение потоков или задач как одновременное.

Планирование должно быть хорошим, чтобы каждый поток работал примерно одинаковое (достаточно малое время) время. Балансу latency/throughput (вычислительное время и время на переключение контекстов / производительность).

В многопроцессорных и многоядерных системах потоки или задачи могут реально выполняться одновременно, при этом каждый процессор или ядро обрабатывает отдельный поток или задачу.

Потоки и процессы

Отношение:

- Процесс имеет главный поток main(), инициализирующий выполнение команд процесса;
- Любой поток может порождать в рамках одного процесса другие потоки;
- Каждый поток имеет собственный стек;
- Потоки, соответствующие одному процессу, имеют общие сегменты кода и данных.

Различия:

- Процессы, как правило, независимы, а потоки существуют как составные элементы процессов;
- процессы имеют отдельные адресные пространства, а потоки выполнения совместно используют их адресное пространство;
- переключение контекста между потоками в одном процессе, как правило, быстрее, чем переключение контекста между процессами.

Планирование потоков

Для того чтобы понимать, в каком порядке исполнять код различных потоков, необходима организация планирования этих потоков. Операционные системы планируют выполнение потоков одним из двух способов:

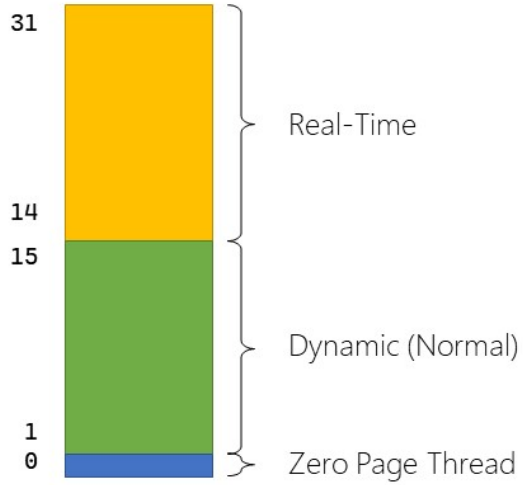
- *Приоритетная (вытесняющая) многопоточность* – операционная система определяет, когда должно происходить переключение контекста. Недостаток приоритетной многопоточности состоит в том, что система может сделать переключение контекста в неподходящее время, что может привести к ряду негативных эффектов, которых можно избежать, применяя кооперативную многопоточность.
- *Кооперативная многопоточность* полагается на сами потоки и отказывается от управления, если потоки выполнения находятся в точках остановки. Это может создать проблемы, если поток выполнения ожидает ресурс, пока он не станет доступным.

Уровни приоритета

Уровни приоритета

Windows имеет 32 уровня приоритета: от 0 до 31, где 31 =наивысший:

- 16 уровней – реального времени;
- 16 уровней – обычные, динамические приоритеты



Работа с памятью

Ядер много, а память одна.

Несколько потоков начинают менять одну и ту же память:

- Гонки за данными (состояние гонки) – ошибка проектирования, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода;
- Блокировки – ожидание потоком условия, которое не может быть выполнено (напр., удержание ресурса другим потоком);
- Потеря данных – одновременная запись потоками переменной в разные ячейки памяти (Long – ново значение ; *, [] – ошибка).

Чтобы этого не происходило, нужно использовать примитивы синхронизации, чтобы получать гарантии того, что некоторые блоки кода будут выполняться в строгом порядке относительно каких-то других блоков кода.

Синхронизация

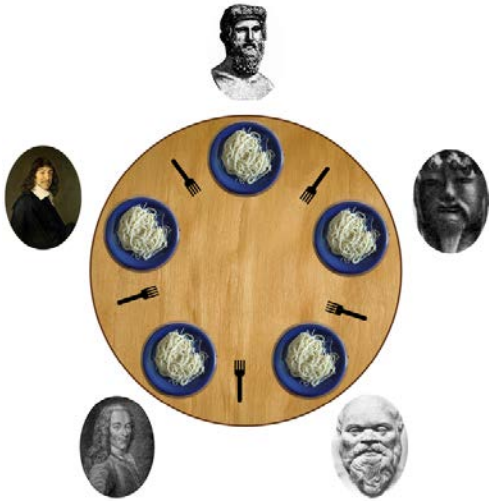
Семафор (англ. semaphore) — примитив синхронизации работы процессов и потоков, в основе которого лежит счётчик, над которым можно производить две атомарные операции: увеличение и уменьшение значения на единицу, при этом операция уменьшения для нулевого значения счётчика является блокирующейся.

Двоичные семафоры – мьютексы. Состояния 1 и 0, изменять состояние может только поток, использующий ресурсы.

Задача об обедающих философах

Иллюстрация проблем синхронизации при разработке параллельных алгоритмов (1965):

1. Пять безмолвных философов сидят вокруг круглого стола, перед каждым философом стоит тарелка спагетти. Вилки лежат на столе между каждой парой ближайших философов.
2. Каждый философ может либо есть, либо размышлять. Запас спагетти бесконечен, но философ может есть только тогда, когда держит две вилки — взятую справа и слева.
3. Каждый философ может взять ближайшую вилку (если она доступна) или положить — если он уже держит её. Взятие каждой вилки и возвращение её на стол являются раздельными действиями, которые должны выполняться одно за другим.



Разработать модель поведения (параллельный алгоритм), чтобы ни один из философов не голодал (вечное чередование пища/размышления).

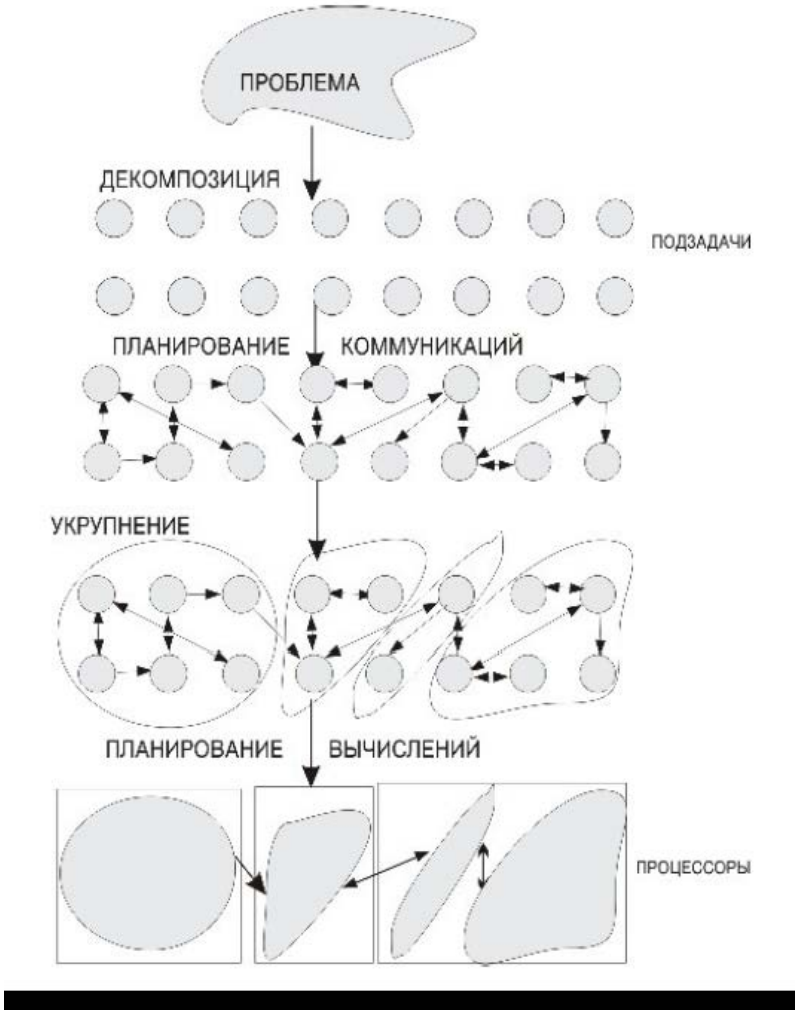
Закон Амдала

Пусть необходимо решить некоторую вычислительную задачу. Предположим, что её алгоритм таков, что доля *a* от общего объёма вычислений может быть получена только последовательными расчётами, а, соответственно, доля 1-*a* может быть распараллелена идеально. Тогда ускорение, которое может быть получено на вычислительной системе из *p* процессоров, по сравнению с однопроцессорным решением не будет превышать величины:

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

a \ p	10	100	1 000
0	10	100	1 000
10 %	5,263	9,174	9,910
25 %	3,077	3,883	3,988
40 %	2,174	2,463	2,496

Разработка параллельной программы



Разработка параллельной программы

Повышенная трудоёмкость параллельного программирования связана с тем, что программист должен заботиться:

- об управлении работой множества процессов;
- об организации межпроцессных пересылок данных;
- о вероятности тупиковых ситуаций (взаимных блокировках);
- о нелокальном и динамическом характере ошибок;
- о возможной утрате детерминизма («гонки за данными»);
- о масштабируемости;
- о сбалансированной загрузке вычислительных узлов.