

1

Роман Елизаров, Никита Коваль

Подробности в группе курса.

Будут домашние задания, которые рекомендуется не откладывать. Они примерно по одному после каждой лекции. Сдача дз через гитхаб.

Сдача дз – допуск к экзамену. Домашних заданий будет много (существенно больше 10)

Литература:

- The art of Miltiprocessor programming
Если прочесть будет большее понимание, но при этом всё, что будет требоваться, будет рассказано.

Зачем нам это?:

- все системы многоядерные. Чем дальше в лес – тем больше ядер.
- The free lunch is over.
- Закон мура (он именно про количество транзисторов на интегральных схемах)
- Частот ядра тоже увеличивается экспоненциальна. Вернее увеличивалась. А после 2005 перестала меняться.
- Производительность ядра при этом не перестала расти. Рост замедлился, но не прекратился.
2005 год – начало эры многопоточности. И уже их число стало расти экспоненциально.

Масштабирование:

- закон Мура (традиционанное) – если дать коду больше ресурсов, то во столько же раз он станет быстрее работать

- сейчас мы хотим так же, но когда мы даём ему больше ядре. Два ядра – в два раза, четыре ядра – в четыре, ...

Но есть проблемы. Ядра нужно синхронизировать (поговорим что и зачем) и это хорошо так съедает производительности.

Закон Амдала:

Ускорение кода $S = \frac{\text{Время на 1 ядре}}{\text{Время на } N \text{ ядрах}}$

$$S = \frac{1}{1 - P + \frac{P}{N}}, \quad P \text{ — доля параллельного кода}$$

график про закон амдала

Из этого можно также вывести, что даже при бесконечном количестве ядер всё упирается в долю параллельного кода.

Вывод: для **масштабируемости** нужно больше **параллелизма**.

scale-up – больше ядер

scale-out – больше машин в сеть

Уровни параллелизма:

- на уровне инструкций

```
a = b + c //(1)
d = e + f //(2)
```

нет зависимости по данным.

Способы: конвейер, суперскалярное исполнение, предсказание переходов, длинное машинное слово, векторизация (single instruction multiple data).
side note: современные процессоры это комбинация суперскалярных и векторизации.

У параллелизма на уровне инструкций есть предел → параллельное программирование.

Большая часть курса будут рассказана на какой-то абстрактной машине, но полезно понимать какими они

бывают и как устроены.

Мультипроцессорность:

- симметричная (SMP). У процессов общая память. Связь с памятью – узкое место.
- одновременная многозадачность (SMT) – ядро физически одно, но оно действует оно как два потока.
Память очень медленная в сравнении с частотой процессора. Ожидание может занимать сотни тактов. Но в SMT один поток может использовать общие модули ядра, пока другой простаивает.
- Ассиметричный доступ к памяти (Non-Uniform Memory Access, NUMA). Модель программирования такая же – общая память, но у ядра есть близкая у нему память и память близкая к другому ядру
картинка ассиметричного доступа

Операционные системы

Типы:

- однозадачные
- система с пакетными заданиями (batch processing)
- многозадачные / с разделением времени (time sharing, много терминалов к одной машине, переключение контекстов). Инструкция переключения/ожидания в коде. А есть вытесняющая многозадачность (preemptive multitasking, в любой момент может произойти переход)

Основные понятия в современных ОС:

- **Процесс** – владеет памятью и ресурсами. ОС создаёт иллюзию, что каждый процесс имеет абстрактную вычислительную систему в своём полном распоряжении
- **Поток** – контекст исполнения внутри процесса. В одном процессе может быть несколько потоков.
- Но в *теории* мы их будем смешивать

Формализм

Нужна формальная модель параллельных вычислений?

Чтобы доказывать:

- корректность алгоритмов
- невозможность построения тех или иных алгоритмов
- минимально-необходимые требования для тех или иных алгоритмов.

Для формализации отношений между прикладным программистом и разработчиком компилятора и системы исполнения кода.

От программиста не ожидается, что он чётко знает как устроен конкретный процессор. Он может писать согласно некоторой модели и рассчитывать, что программа будет работать.

Модели программирования:

"Классическое" однопоточное/однозадачное:

- ресурсы многоядерной системы задействуются только запуском множество разных, независимых задач.

Многозадачное

- одна задача может задействовать ресурсы многих ядер

Модель с общими объектами (общей памятью)

картинка 

Потоки выполняют операции над общими, разделяемыми объектами.

Многозадачное программирование:

- одна задача – ресурсы многих ядер

Варианты:

- модель с общей памятью
- модель с передачей сообщений (распределённое программирование)

Базовый общий объект – общая переменная:

- значение определённого типа

- операции чтения (read) и записи (write)
Является базовым строительным блоком, потому что реализован аппаратно.
Модели с общими переменными это хорошая абстракция современных многопроцессорных систем и многопоточных ОС.
На практике это область памяти *процесса*, коорая одновременно доступна многим потокам.

Свойства многопоточных программ:

- Последовательные программы детерминированы. (Если нет явного использования случайных чисел или другого общения с недетерминированным внешним миром). Их свойства можно установить анализируя **исполнение** при данных входных параметрах.
- Многопоточные программы в общем случае недетерминированны. Даже если код каждого потока детерминирован. Результат зависит от того как пойдёт исполнение.

Говорим **программа А имеет свойство Р**, если программа А имеет свойство Р при любом исполнении.

```
shared int x = 0, y = 0
```

```
1: x = 1  
2: r1 = y  
3: stop
```

```
1: y = 1  
2: r2 = x  
3: stop
```


Метод чередования – перебираем все последовательности исполнения. Считаем, что сначала работает один процессор, а потом другой.

S – общее состояние

Два перехода f, g

Два новых состояния $f(S), g(S)$

Операции над разными общими переменными коммутируют ($x=1, y=1$)

 пример-всё вместе

Выводы:

- модель чередования не отражает реальность

В теоретических трудах это называется регистрами (нет это не те регистры, которые локальная быстрая память процессора)

1

 **Параллельное программирование с 2022-09-05**