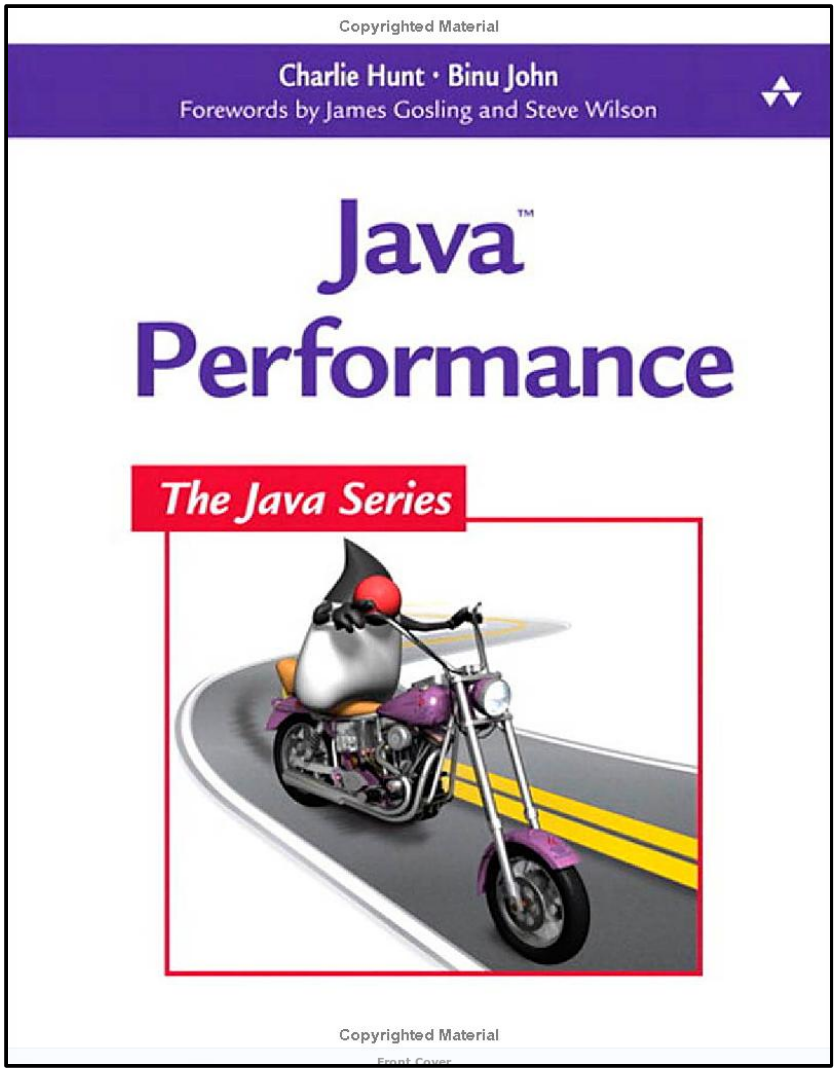
MOVING JAVA FORWARD

ORACLE

Performance Engineering How-To

Сергей Куксенко
Java SE Performance
sergey.kuksenko@oracle.com
@kuksenk0

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.



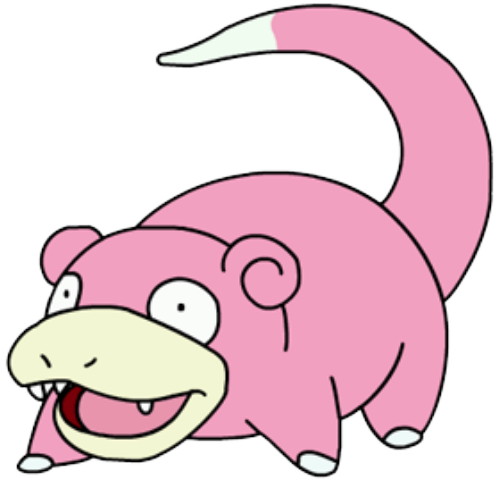
ORACLE

Performance Engineering

Об абстракциях

- Computer Science → Software Engineering
  - Строим приложения по функциональным требованиям
  - В большой степени абстрактно, в “идеальном мире”
    - Теоретически неограниченная свобода – искусство!
    - Можно строить воздушные зАмки
  - Рассуждения при помощи формальных методов

- Software Performance Engineering
  - “Real world strikes back!”
  - Исследуем взаимодействие софта с железом на типичных данных
    - Производительность уже нельзя оценить
    - Производительность можно только измерить
  - Естественно-научные методы
    - Основываемся на эмпирических данных



У меня всё медленно!
Ваш первый шаг?

Первый шаг

- Классические ошибки первого шага
  - “я вижу, что метод foo() реализован неэффективно”
  - "по профилю видно, что метод bar() – горячий и занимает 5%"
  - "по-моему, у нас тормозит БД, и нужно перейти с DB_x на DB_y"

Первый шаг

- Классические ошибки первого шага
  - "я вижу, что метод foo() реализован неэффективно"
  - "по профилю видно, что метод bar() – горячий и занимает 5%"
  - "по-моему, у нас тормозит БД, и нужно перейти с DB_x на DB_y"
- Правильный первый шаг:
  - Выбрать метрику
    - ops/sec, transactions/sec
    - время исполнения
    - время отклика
  - Убедиться в корректности метрики
    - релевантна (учитывает реальный сценарий работы приложения)
    - повторяема

Цель – улучшение метрики!

Метрики

ORACLE

Метрики (1)

- **Throughput, Bandwidth**
  - Количество работы, выполненное за единицу времени
  - Принимает разные формы:
    - MB/sec
    - ops/sec, transactions/sec
    - FPS (frames per second, frags per second)
    - MIPS
    - FLOPS

Метрики (2)

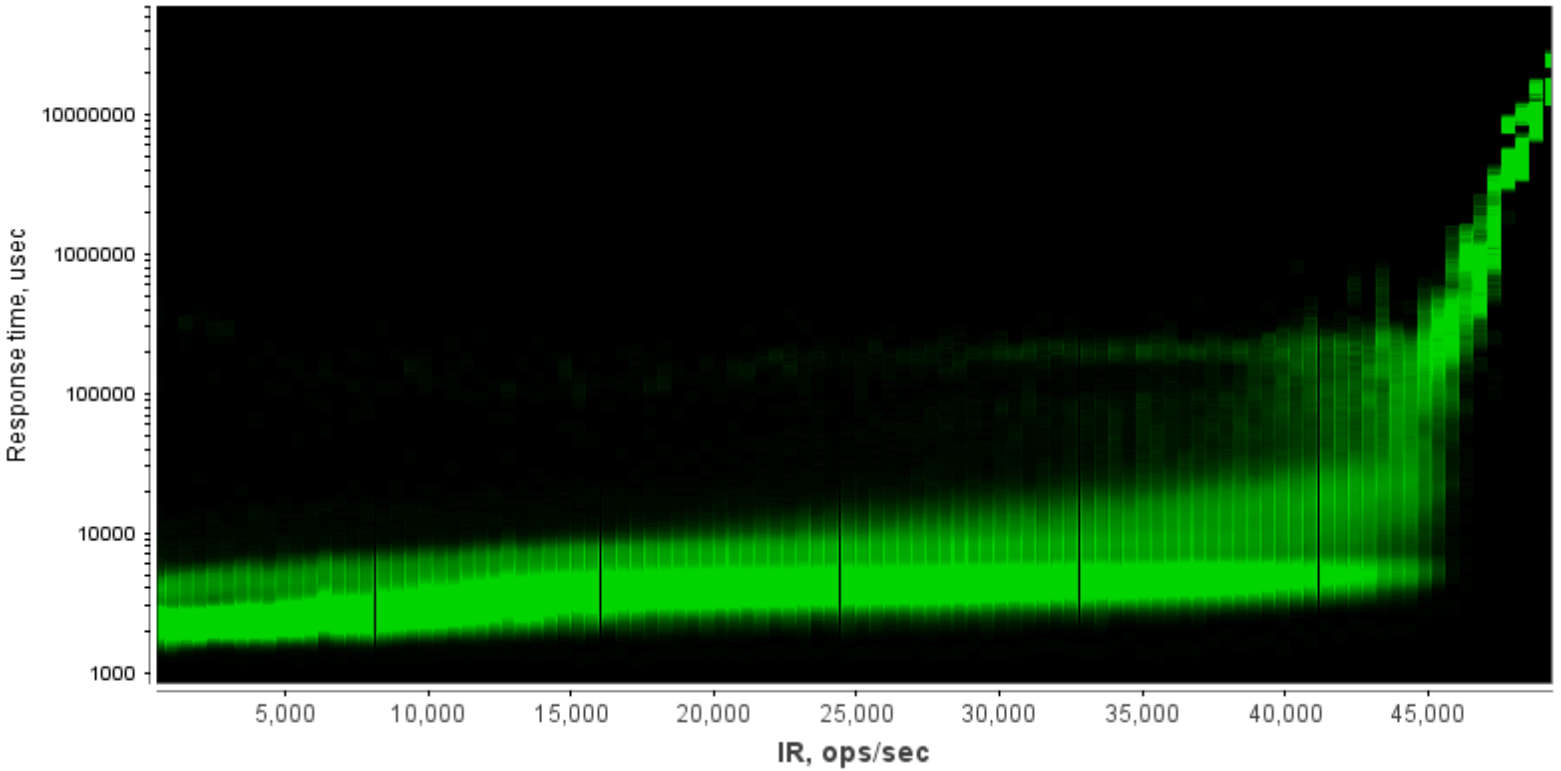
- **Время...**
  - ...работы
    - Execution time: общее время исполнения
  - ...отклика
    - Latency: время отдельной операции
    - Response time: задержка между стимулом и реакцией
  - ...запуска
    - Startup time: время до начала работы
    - Time to performance: время до начала *хорошей* работы
  - etc.

Метрики (3)

- Производительность – композитная метрика
  - throughput vs RT → throughput & RT
- Оказывается, что проще улучшить throughput
  - DDR2 PC2-3200: 3200 Mb/sec, CL 4ns
  - DDR2 PC2-6400: 6400 Mb/sec, CL 5ns
- В большинстве систем, RT тесно связан с throughput
  - Из пункта А в пункт Б один автобус может перевезти сотню человек
  - Достаточно много автобусов перевезут миллион человек
  - Несколько больших автопоездов перевезут миллион человек
    - ...но в обоих случаях будут дикие очереди на погрузку и выгрузку :)

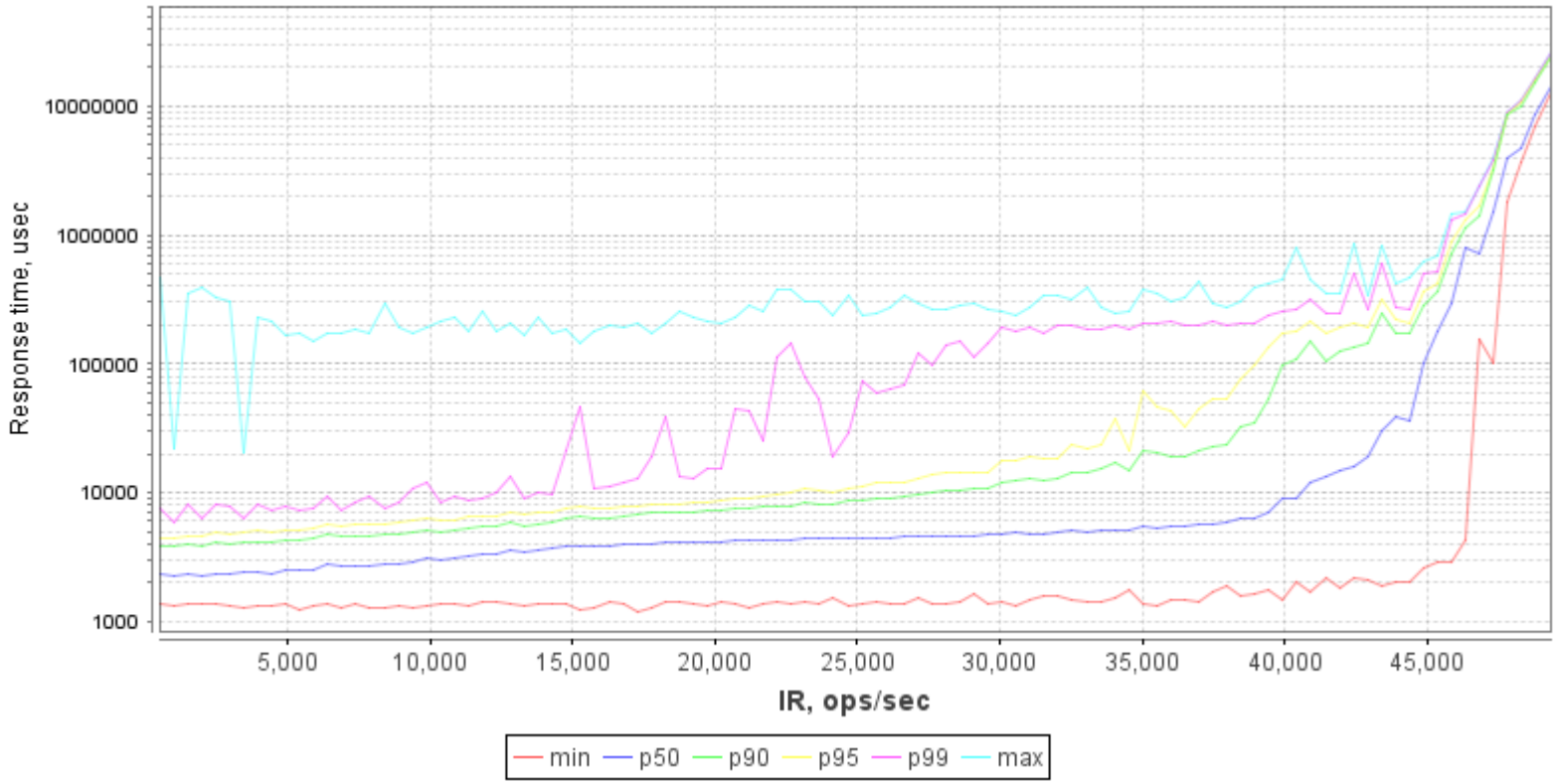
Метрики (4)

- Обычно RT быстро растёт с throughput



Метрики (5)

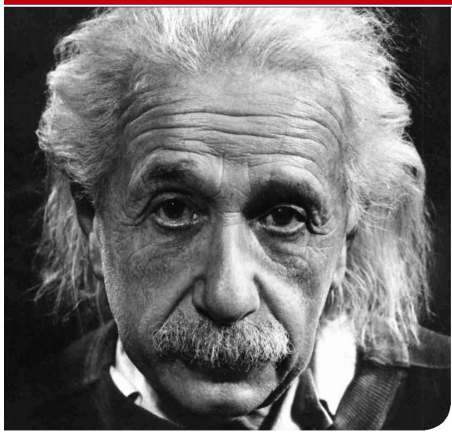
- Обычно RT быстро растёт с throughput



Метрики (6)

- Прочие метрики
  - потребление ресурсов
    - память, сеть, etc.
  - отказоустойчивость
    - MTBF, MTTR
  - потребляемая и отводимая мощность
    - performance per watt
  - TDP
  - etc.

Теория



Утилизация

- Утилизация – насколько ресурс занят?

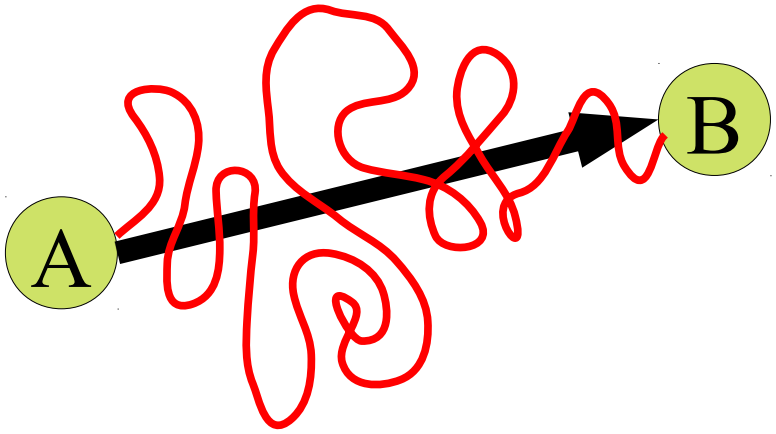
$$Utilization = \frac{ResourceBusyTime}{TotalTime}$$

- Idle – насколько ресурс свободен?

$$Idle = 1 - Utilization$$

Эффективность

- Эффективность (efficiency)
  - Оценка КПД для времени: часть общего времени, потраченная на выполнение полезной работы
  - Субъективно, невозможно строго вычислить
  - NB: Высокая утилизация CPU не означает хорошую эффективность



SpeedUp

- “A в N раз быстрее B” означает

$$SpeedUp = \frac{time(B)}{time(A)} = \frac{throughput(A)}{throughput(B)}$$

Boost%

- “A на n% быстрее B” означает:

$$SpeedUp = 1 + \frac{n}{100\,\%}$$

$$Boost\,\% = (SpeedUp - 1) * 100\,\%$$

$$Boost\,\% = \frac{time(B) - time(A)}{time(A)}$$

$$Boost\,\% = \frac{throughput(A) - throughput(B)}{throughput(B)}$$

Закон Амдала

- Допустим, есть две независимые части:
  - Часть **A** занимает 70% времени, ускорябельна в 2 раза
  - Часть **B** занимает 30% времени, ускорябельна в 6 раз
  - Какую часть ускорять?

ORACLE

Закон Амдала (2)

- Допустим, есть две независимые части:
  - Часть **A** занимает 70% времени, ускорябельна в 2 раза
  - Часть **B** занимает 30% времени, ускорябельна в 6 раз
  - Какую часть ускорять?

Ускорим **B** в 6 раз:

Ускорим **A** в 2 раза:

ORACLE

Закон Амдала (3)

- Каноническое определение:

$$Part(A) = \frac{A}{A+B}$$

$$SpeedUp = \frac{1}{(1 - Part(A)) + \frac{Part(A)}{SpeedUp(A)}}$$

Закон Амдала (4)

- Допустим, есть две независимые части:
  - Часть **A** занимает 70% времени, ускорябельна в 2 раза
  - Часть **B** занимает 30% времени, ускорябельна в 6 раз
  - Какую часть ускорять?

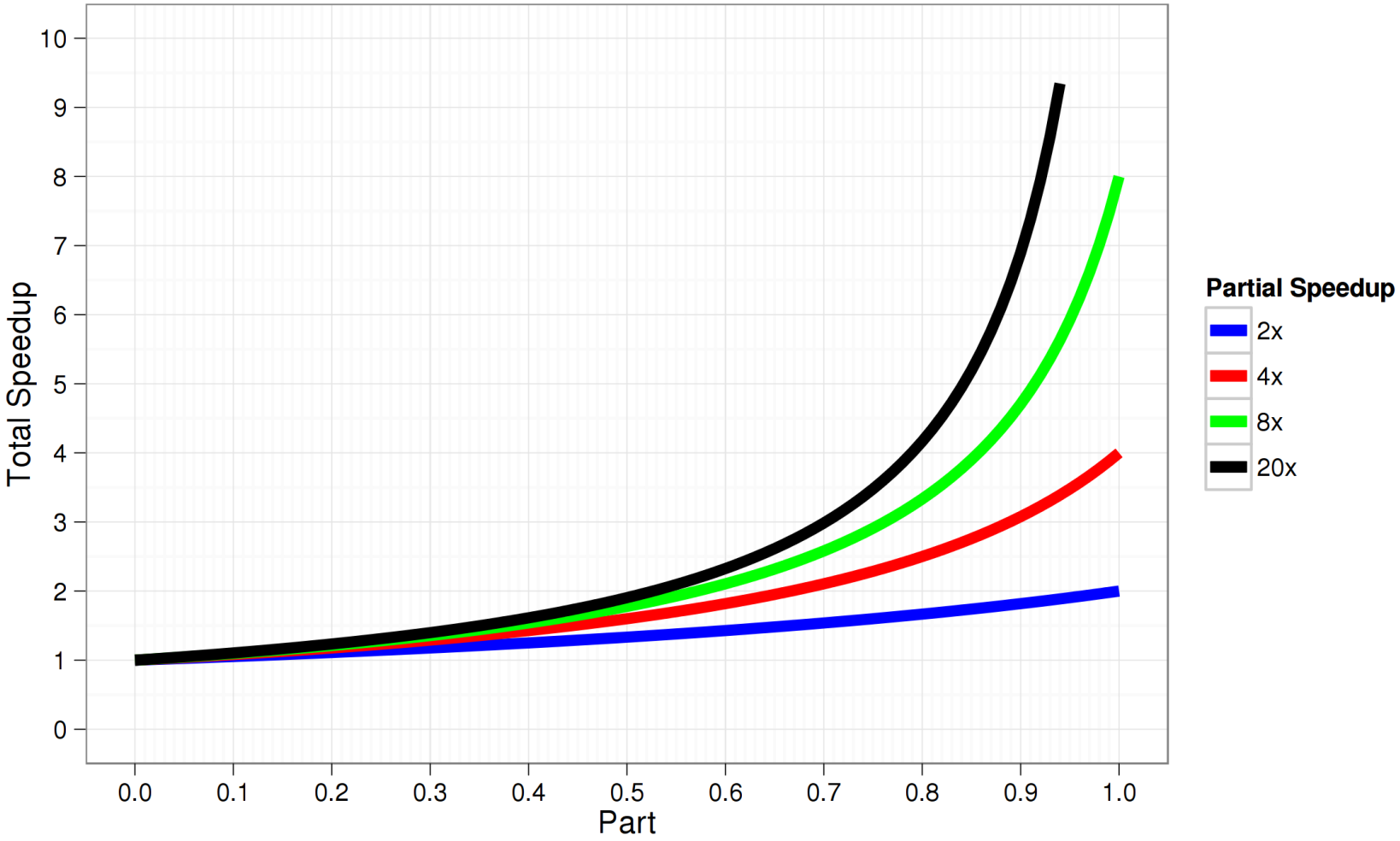
Ускорим **B** в 6 раз:

+33%

Ускорим **A** в 2 раза: +53%

ORACLE

Закон Амдала (5)



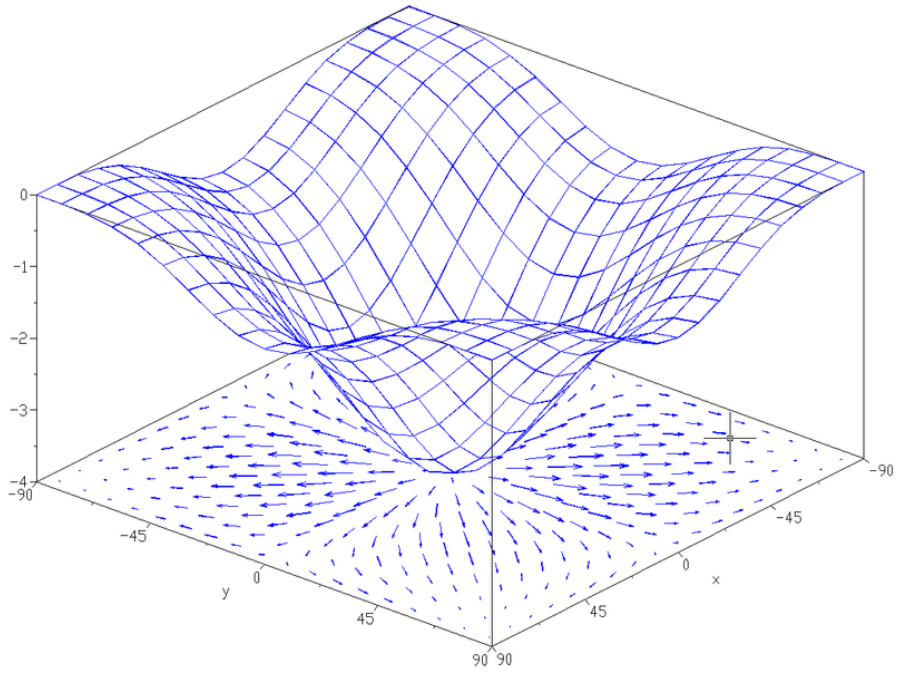
ORACLE

Когда закон Амдала не работает

- Composability
  - Предположим, есть функциональные блоки A и B
  - Разница при последовательном и параллельном исполнении?
- Общая функциональность:
  - Functionality(A ... B) = Functionality(A || B)
  - “Black Box”: поведение одинаково
- Общая производительность:
  - Performance(A ... B) **???** Performance(A || B)
  - Про это сказать толком ничего нельзя
    - A и B соревнуются за аппаратные ресурсы
    - Возможно, > (эффективный размер кеша меньше)
    - Возможно, < (два потока на HT машине)
    - Возможно, = (нет конфликтов)

Scalability

- Пространство ресурсов:
  - В общем случае N-мерное
  - K^n
- Производительность:
  - Скалярное поле
  - $P : K^n \rightarrow \mathbb{R}$
- Масштабируемость:
  - Градиент производительности
  - $S = \nabla P$
- Масштабируемость по ресурсу:
  - Рост P в конкретном направлении
  - $S_i = \frac{\partial P}{\partial R_i}$



ORACLE

Scalability: подходы

- Пространство велико!
  - Полный обход возможен, но не практичен
  - Хотя иногда других вариантов нет
- Случайные блуждания в направлении градиента?
  - Нужна оценка градиента в текущей точке
  - Попробовать дёрнуться по N направлениям
- Локальная оценка!
  - Оценить, будет ли расти P, если добавить ресурса
  - Т.о. оценить, в какую сторону растёт градиент P
  - Для этого нужно уметь диагностировать bottleneck'и

Методология

Второй шаг: ошибки

- «Я вижу, что метод foo() реализован плохо, перепишем и посмотрим, что изменится»

Второй шаг: ошибки

- «Я вижу, что метод foo() реализован плохо, перепишем и посмотрим, что изменится»
  - …а метод не используется вообще.
  - …или используется, но занимает пару миллисекунд
  - …или используется, но *дело не в этом*
- Не самый плохой вариант
  - ...если таких методов мало
  - ...и изменения очень быстрые

Второй шаг: ошибки

- «По профилю видно, что метод bar() – самый горячий и занимает целых 5%, надо его убить»

Второй шаг: ошибки

- «По профилю видно, что метод bar() – самый горячий и занимает целых 5%, надо его убить»
  - …а оказывается, что это 5% от всего приложения, которое занимает 6.25% CPU 16-процессорной системы
  - …или это метод, помогающий всем остальным быть быстрее
  - …или он действительно проблемный, но *дело не в этом*

Второй шаг: ошибки

- «Полносистемная профилировка показывает, что у нас тормозит база данных, и нужно срочно перейти с СУБД$_X$ на СУБД$_Y$»

Второй шаг: ошибки

- «Полносистемная профилировка показывает, что у нас тормозит база данных, и нужно срочно перейти с $СУБД_X$ на $СУБД_Y$»
  - …а вдруг оказывается, что диски слабоваты
  - …или оказывается, что злые админы зашейпили сеть
  - …или просто БД уже давно никто не «пылесосил»
- Мы наблюдали спонтанные переезды с X на Y
  - ...и обратно, немного погодя. А потом опять с X на Y.

Как ускорить приложение?

- К.О. сообщает:
  - “Нужно что-то где-то как-то изменить!”
- Что?
- Где?
- Как?

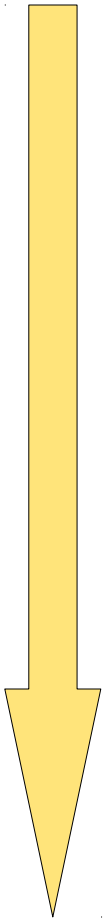
Как ускорить приложение?

- К.О. сообщает:
  - “Нужно что-то где-то как-то изменить!”
- Что мешает работать быстрее?
- Где это находится?
- Как это исправить?

Как ускорить приложение?

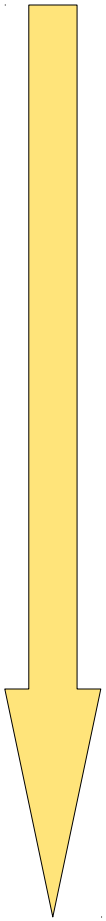
- К.О. сообщает:
  - “Нужно что-то где-то как-то изменить!”
- Что мешает работать быстрее?
  - Используем голову и monitoring tools
- Где это находится?
  - Используем голову и profiling tools
- Как это исправить?
  - Используем голову и прямые руки

Нисходящий подход (классика)

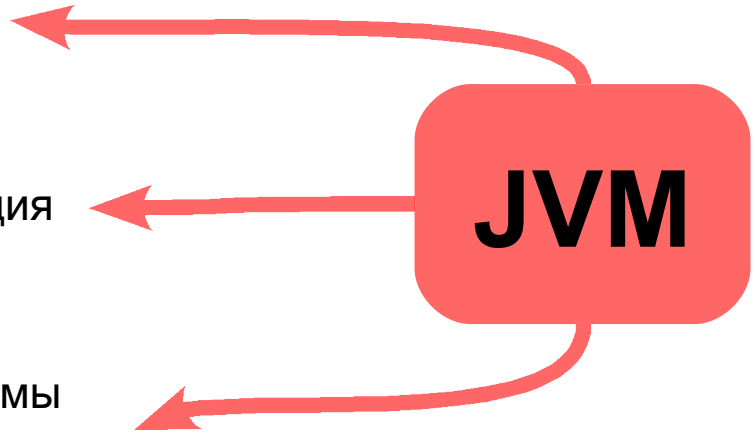


- Уровень системы
  - Сеть
  - Диск
  - Операционная система
  - Процессор/память
- Уровень приложения
  - Блокировки, синхронизация
  - Execution Threads
  - API
  - Алгоритмические проблемы
- Микроархитектурный уровень
  - Code/data alignment
  - Cache optimizations
  - Processor stalls
  - Branch prediction

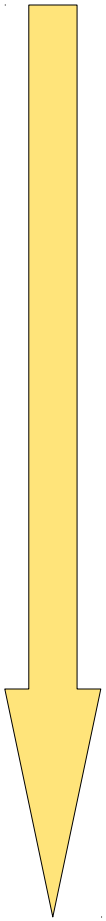
Нисходящий подход (классика)



- Уровень системы
  - Сеть
  - Диск
  - Операционная система
  - Процессор/память
- Уровень приложения
  - Блокировки, синхронизация
  - Execution Threads
  - API
  - Алгоритмические проблемы
- Микроархитектурный уровень
  - Code/data alignment
  - Cache optimizations
  - Processor stalls
  - Branch prediction

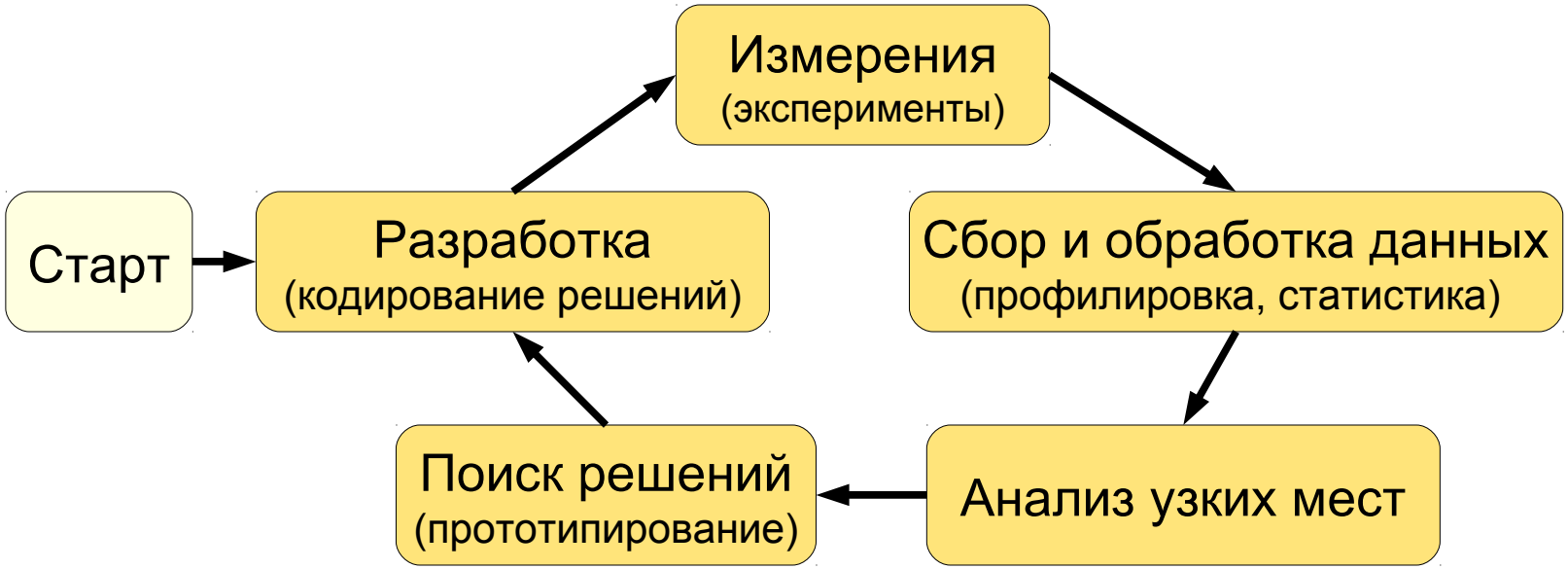


Нисходящий подход (Java)



- Уровень системы
  - Сеть
  - Диск
  - Операционная система
  - Процессор/память
- Уровень JVM
  - Выбор JVM
  - Heap/GC tuning
  - JVM tuning
- Уровень приложения
  - Блокировки, синхронизация
  - Execution Threads
  - API
  - Алгоритмические проблемы
- Микроархитектурный уровень
  - Temporal/spatial data locality
  - Cache optimizations
  - Branches

Итеративный подход



Важно:

- Новая фаза только после пройденных функциональных тестов
- Одно изменение за цикл!
- Документировать все изменения

Детали (#толькохардкор)

ORACLE

Системный уровень

- Выбрали сценарий, метрику
- Запустили нагрузочный тест
- Смотрим на CPU utilization

```
$ mpstat
CPU     %usr    %sys  %iowait   %irq   %idle
all    44.06   21.42    10.71   0.00   23.76
```

- Дальше есть варианты:
  - Много sys%
  - Много irq%, soft%
  - Много iowait%
  - Много idle%
  - Много user%

Много sys%

- Обычно дело не в приложении
- Варианты:
  - Сеть жрёт?
  - Шедулер жрёт?
  - Swapping?
  - Other kernel?

Много sys%: сеть

- Как диагностировать?
  - netstat, sar
  - iptraf, bwm-ng
- Что видно?
  - (как бы очевидно)
- Что можно сделать?
  - Проверь провода, бле%%ь!
  - Реже пишем/читаем
  - Меньше пишем/читаем, сжатие
  - Буферизация, Bandwidth-Delay Product, MTU
  - Быстрее физические интерфейсы
  - Виртуальные интерфейсы

Много sys%: шедулер

- Как диагностировать?
  - vmstat
  - mpstat
- Что видно?
  - Большой runqueue
  - Большой ctxsw/sec
- Что можно сделать?
  - Ограничиваем количество потоков

Много sys%: swapping

- Как диагностировать?
  - top
  - sar
- Что видно?
  - много лежит в swap
  - swap меняется: много page fault'ов
- Что можно сделать?
  - Больше памяти машине
  - Меньше памяти процессу
  - “swappiness”

Много sys%: other kernel

- Как диагностировать?
  - strace
  - perf, oprofile
- Что видно?
  - Какие системные вызовы используются
  - Нужны ли эти системные вызовы?
- Что можно сделать?
  - Тюнить ядро
  - Открывать баги на ядро

Много irq%

- Обычное дело при общении с устройствами
- Как диагностировать?
  - mpstat
  - sar
- Что видно?
  - Какие прерывания используются и кем
- Что можно сделать?
  - Убрать котэ с клавиатуры
  - Умерить таймеры
  - Offload
  - IRQ Balancing

Много iowait%

- Обычное дело для дискового I/O
  - ...или I/O в смонтированные сетевые ФС
- Как диагностировать?
  - iostat
  - sar
- Что видно?
  - Часто лазим или много делаем
  - Не хватает файловых кешей или буферов
- Что можно сделать?
  - Не лазим часто, не пишем много
  - Быстрее диски (особенно для мелких записей)
  - Больше места под дисковые кеши
  - Больше места под дисковые буфера

Много idle%

- Иногда это и неплохо: latency!
- Как диагностировать?
  - vmstat, mpstat
  - jstack
  - -verbose:gc
- Что видно?
  - Недостаточно потоков
  - Недостаточно RUNNABLE потоков
  - Недостаточно потоков в GC
- Что можно сделать?
  - Уровень JVM: настраиваем GC
  - Уровень приложения: ЗАБИВАЙ @ ПАРАЛЛЕЛЬ
  - Уровень приложения: ищем блокировки

Много idle%: wait locks

- Как диагностировать?
  - jrmc и другие lock profilers
  - jstack (много раз)
- Что видно?
  - Много потоков стоят на локах
  - При этом очень мало RUNNABLE
- Что можно сделать?
  - Потихоньку удаляем блокировку за блокировкой
    - Обычно видно только самую толстую блокировку
  - Переезжаем на lock-free алгоритмы и структуры данных
    - Может оказаться, что этим переезжаем в “много user%”

Много user%

- Ура, тут уже есть где развернуться
  - Здесь уже могут понадобиться профайлеры
  - Примерная программа:
    - JVM (GC, JIT, Classload)
    - Алгоритмы
    - Память (TLB, bandwidth, caches, NUMA)
    - Проблемы с CPU

Много user%: выбор JVM

- Держимся за свежие релизы
  - 1.4.2 уже пора закапывать
  - 1.5 уже давно в EOL
  - 1.6 будет в EOL в феврале 2013
  - Оптимизации в старые релизы backport'ятся за большие $$$
- Иногда проще поменять JVM целиком
  - Oracle (Sun) HotSpot
    - Client VM
    - Server VM
  - Oracle (BEA) JRockit
  - [эту JVM нам нельзя упоминать]
  - [и эту тоже]
  - [и эту]

Много user%: JVM, GC

- Как диагностировать?
  - -verbose:gc
  - -XX:+PrintGCDetails
  - VisualGC
- Что видим?
  - Частоту и продолжительность сборок
  - GC pause times
- Что можно сделать?
  - Настраиваем кучу и поколения
  - Выбрать другой GC
  - Thread Stack Size
  - Общий тюнинг GC
    - (поля этого доклада слишком узки)

Много user%: JVM, JIT

- Как диагностировать?
  - -XX:+PrintCompilation
  - MXBeans (Visual VM)
- Что видим?
  - Компиляция долгая и мешает приложению
- Что можно сделать?
  - Выбрать другой режим JVM
  - Ограничить количество потоков
  - Прочая ручная настройка
    - (требуется очень редко)

Много user%: JVM, Classloader

- Как диагностировать?
  - -verbose:class
  - MXBeans (Visual VM)
- Что видим?
  - Загрузка классов занимает много времени
- Что можно сделать?
  - Отключить верификацию
  - Class Data Sharing
  - Увеличить размер system dictionary
    (-XX:PredictedLoadedClassCount)
  - Переупаковка (меньшее количество бОльших JARs)
  - Ждать Jigsaw :)

Много user%: алгоритмы

- Как диагностировать?
  - Любой Java-профайлер на ваш вкус
    - Обязательно на разных наборах входных данных
  - Вдумчивое чтение кода
- Что можно сделать?
  - Алгоритмическая сложность
    - Алгоритмы с меньшей сложностью
    - Алгоритмы с меньшими константами
  - (Анти)кеширование данных
    - Где надо, мемоизируем
    - Где надо, используем свежие объекты
  - Убираем активные ожидания
    - Polling
    - Spinloops

Много user%: память

- Проблемы с памятью и кешами маскируются
  - Видны просто как “занятые” циклы: user%
  - В профайлере видны как медленный код
- Есть много вариантов:
  - TLB
  - Memory bandwidth
  - Caches
    - Capacity
    - Coherency
  - NUMA

Много user%: память, TLB

- TLB = Translation Lookaside Buffer
  - Кэш транслятора виртуальных адресов в физические
  - Каждое обращение к памяти – обращение к TLB
  - Гранулярность TLB – страница памяти
- Как диагностировать?
  - solstudio, vtune, perf: хардварные счётчики
  - Проще попробовать исправить
- Что можно сделать?
  - Меньший working set
  - -XX:+UseLargePages
    - Кое-где требует администраторских привилегий, RTFM

Много user%: память, bandwidth

- Кэши – это хорошо, но в память лазить надо
  - По latency догнать сложно
  - По bandwidth догнать достаточно легко
- Как диагностировать?
  - busstat: статистика по шине
  - Дифференциальный диагноз
- Что можно сделать?
  - Многоканальная память
  - Многоканальные контроллеры памяти
  - Несколько CPU

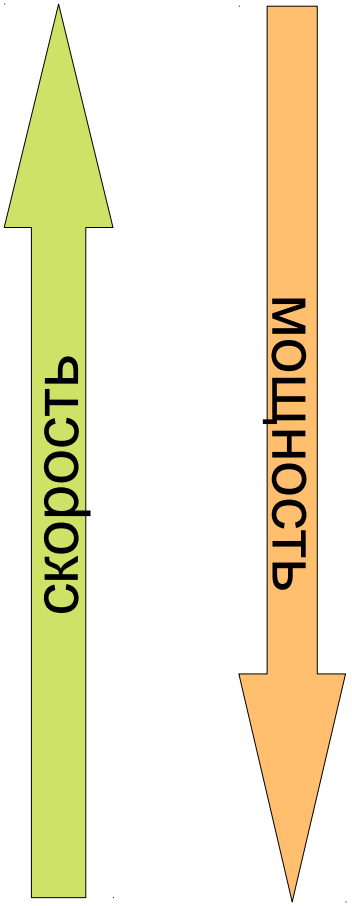
Много user%: caches, capacity

- Кэши не резиновые
  - “Миллионы приложений прямо сейчас страдают от cache miss'ов”
- Как диагностировать?
  - application-профайлеры: медленная работа с памятью
  - solstudio, vtune, perf: хардварные счётчики
- Что можно сделать?
  - Уменьшить working set
    - -XX:+UseCompressedOops
  - Temporal/Spatial locality
    - Блочные декомпозиции
    - Плотнее структуры данных
  - Больше префетчим vs. меньше гадим
    - Включить / выключить HW prefetcher
    - -XX:AllocPrefetchStyle=#
    - Non-temporal операции

Много user%: caches, coherency

- Коммуникация через shared-память
  - Единственный эффективный способ коммуникации на SMP
  - Примитивы синхронизации = примитивы коммуникации
  - Сравнительно просто в железе при плоской иерархии памяти
- Кеши дают иллюзию большой быстрой памяти
  - Часто используемые данные мигрируют между кешами
  - Требуется координация между кешами
- Протоколы когерентности синхронизируют кеши
  - Фактически, message passing, реализованный в хардваре
  - Coherency-трафик – необходимое зло для коммуникации

Caches, coherency: primitives



- Plain unshared memory
- Plain shared memory
  - Коммуникация
- Volatile
  - Плюс видимость
- Atomics (CAS)
  - Плюс атомарность
    - База для wait-free алгоритмов
- Atomic sections (spin-loops)
  - Плюс атомарность группы изменений
    - База для lock-free алгоритмов
- Spin-locks
  - Гарантирует mutual exclusion
    - + появляется ownership
- Wait-locks
  - Плюс блокировка

(как правило; YMMV)

Много user%: caches, coherency

- Как диагностировать?
  - application-профайлеры: горячие места в общих записях
  - solstudio, vtune, perf: хардварные счётчики
- Что можно сделать?
  - Выбрать правильный примитив коммуникации
    - Самый слабый из подходящих
  - Лёгкие проверки
    - Проверить более слабое условие перед тяжёлой операцией
  - Striping
    - Разбить общее место на несколько мелких
  - Отказаться от коммуникации вообще
    - Отдельные копии, thread locals
  - Страхуемся от false sharing
    - Паддинг объектов
    - Разрывание объектов

Много user%: caches, coherency

- Правильные примитивы
  - Выбираем самый слабый примитив для наших требований
    - Он же будет самым быстрым
  - Не волнует видимость?
    - Не нужен volatile!
  - Безопасно опубликовать значение?
    - volatile вместо synchronized
  - Thread-safe счётчик?
    - Atomic* вместо synchronized/Lock
  - Гарантированно короткая критическая секция?
    - CAS-ed loop вместо локов

Много user%: caches, coherency

- Лёгкие проверки
  - Часто можно сделать проверку без тяжёлого действия
  - Если примитивы позволяют легкие операции

```
AtomicBoolean isSet = ...;
if (!isSet.get() &&
     isSet.compareAndSet(false, true) {
    // one-shot action
}
```


Много user%: caches, coherency

- Лёгкие проверки
  - Часто можно сделать проверку без тяжёлого действия
  - Если примитивы позволяют легкие операции

```
ReentrantLock lock = ...;
int count = -LIMIT;
while (!lock.tryLock()) {
   if (count++ > 0) {
      lock.lock();
      break;
   }
}
```


Много user%: caches, coherency

- Striping
  - Иногда удаётся расклеить состояние без нарушения свойств
  - Суперпозиция локальных состояний есть общее состояние
- Пример: эффективный thread-safe счётчик
  - synchronized { i++; }
  - `AtomicInteger.inc(...)`?
    - Вряд ли, одна общая переменная
  - `ThreadLocal<Integer>.set(...)`?
    - Хорошо скалируется, но трудности с агреггацией
  - `AtomicInteger[random.nextInt(count)].inc(...)`
    - Хорошо скалируется: при достаточном $count мало конфликтов
    - Общее состояние: сумма всех счётчиков
    - см. LongAdder @ jsr166e

Много user%: caches, coherency

- Отказываемся от коммуникации
  - Личные копии объектов (= immutability)
  - ThreadLocal
- Пример: ThreadLocalRandom @ JDK7
  - Random имеет одну переменную-состояние
    - Использует CAS, чтобы её обновлять
  - ThreadLocalRandom поддерживает локальные копии
    - По сути, пачка генераторов за одним фасадом

Много user%: caches, coherency

- Квант обмена с кэшами и памятью – cache line (CL)
  - В байтах: 32 (ARM), 64 (x86/SPARC/ARM), 128 (Power)
  - Протоколы когерентности тоже работают на целых CL
  - В MESI поддерживается информация о “владельцах” CL

```
...------][----AA--BB----][-----...
```

- Сильно облегчает работу с памятью
  - Прочитали AA и нужна BB? Она уже прочитана в кеш!
  - HW prefetcher'ы могут затащить и второй cache line

- Бочка дёгтя: False Sharing
  - CPU1 пишет в AA? **Трансфер**. CL теперь владеет CPU1
  - CPU2 пишет в BB? **Трансфер**. CL теперь владеет CPU2

Много user%: caches, coherency

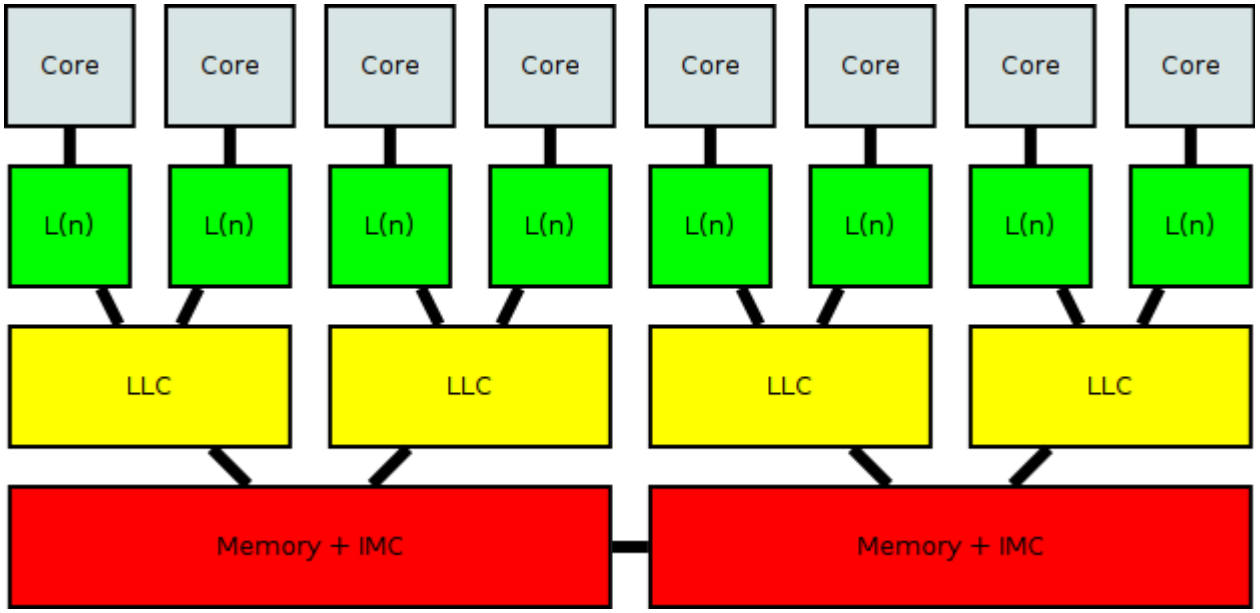
- False Sharing
  - Типичный способ устранить: паддинг объекта
    - Надеемся, JVM не переставляет поля с одинаковыми сигнатурами
    - Правильный Способ™ для нескольких полей: разорвать объект на независимые части и отпаддить по отдельности
  - Размер паддинга зависит от размера CL
    - Объект занимает целиком CL
  - Существенно увеличивает размер объекта
    - Делать только там, где сильно нужно

```
static final class Cell {
    volatile long p0, p1, p2, p3, p4, p5, p6;
    volatile long value1;
    volatile long q0, q1, q2, q3, q4, q5, q6;
}
```

- JEP-142 (@Contended)

Много user%: память, NUMA

- Non Uniform Memory Access
  - “Доступ к памяти занимает разное время”
  - Для SMP память “uniform” и не была – иерархические кеши
  - Тот же принцип: коммуникация – это дорого



Много user%: память, NUMA

- Как диагностировать?
  - busstat: трафик на шине
  - O/S specific инструменты
- Что можно сделать?
  - Ограничиваем коммуникацию
    - На этот раз относится и к *чтению* данных
  - Личные объекты ближе к себе, разделяемые – interleaving
    - -XX:+UseNUMA
    - На Windows работает не полностью, только interleaving
  - Process/Thread Affinity
    - taskset/numactl: JVM можно привязать к конкретным CPU
    - Можно привязать конкретные потоки

Много user%: CPU

- Процессоры кончаются в последнюю очередь
  - Большинство приложений сюда даже не добираются
- Есть несколько вариантов:
  - Не хватает частоты
  - Не хватает блоков исполнения
  - Пределы ILP

Процессорные метрики

- Clocks Per Instruction, CPI
  - Описывает параллелизм команд (чем меньше, тем лучше)
  - Обратная величина: Instructions Per Clock, IPC
- Path Length
  - количество инструкций на пути исполнения
- Время на выполнение:

$$TimeToExecute = P * CPI * Path\ Length$$

Много user%: CPU, частота

- Программа упёрлась в скорость вычислений
  - Path Length уменьшить уже нельзя
  - CPI уменьшить уже нельзя
  - Можно ли уменьшить P?
- Что можно сделать?
  - Оверклокинг
  - Настройки cpufreq
    - Мало ли серверов и датацентров бегает с ondemand?

$$TimeToExecute = P * CPI * Path\ Length$$

Много user%: CPU, вычислялки

- Программа упёрлась в скорость вычислений
  - Разумными средствами Path Length и CPI уже не уменьшить
- Как диагностировать?
  - solstudio, vtune, perf: хардварные счётчики
- Что можно сделать?
  - Больше процессоров!
  - Специализированное оборудование
    - GPU
    - Криптопроцессоры
  - Специализированный код
    - Native-вставки
    - Пишем вставки в JIT (#толькохардкор)

Много user%: CPU, ILP

- Программа упёрлась в скорость вычислений
  - Разумными средствами CPI уже не уменьшить
- Как диагностировать?
  - solstudio, vtune, perf: хардварные счётчики
- Что можно сделать?
  - Больше ILP из кода
    - Меньше зависимостей по данным
  - Branch predictors
    - Кое-где помочь, кое-где запутать
  - Заигрывание с LSD*
    - Мелкие циклы

(*) Loop Stream Detector, а не то, что вы подумали


JavaOne™