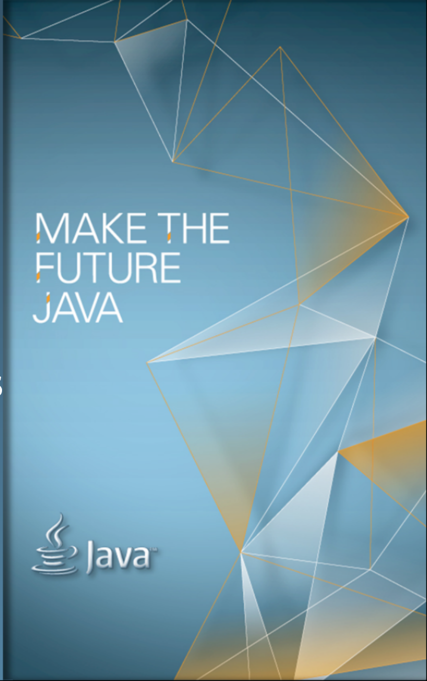
“Quantum” Performance Effects

Сергей Куксенко
sergey.kuksenko@oracle.com, @kuksenk0



Outline

Введение

Core

Not-a-Core

Заключение

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.

Введение

Введение: стандартный дисклеймер

1. Computer Science → Software Engineering
   - Строим приложения по функциональным требованиям
   - В большой степени абстрактно, в “идеальном мире”
   - Рассуждения при помощи формальных методов

2. Performance Engineering
   - “Real world strikes back!”
   - Исследуем взаимодействия софта с железом на типичных данных
   - Эффективно предсказывается уже мало что
   - Рассуждения при помощи формальных методов

Введение: в чем разница?

архитектура vs микроархитектура

Введение: в чем разница?

архитектура vs микроархитектура

x86 AMD64(x86-64/Intel64) ARMv7	Nehalem Sandy Bridge Bulldozer Bobcat Cortex-A9

Введение: SUTs[1]

- Intel® Core™ i5-520M (Westmere) [2.0 GHz] 1x2x2
  - Ubuntu 10.04.4 LTS (32-bits)

[1] System Under Test

Введение: SUTs[1]

- Intel® Core™ i5-520M (Westmere) [2.0 GHz] 1x2x2
  - Ubuntu 10.04.4 LTS (32-bits)
- Samsung Exynos 4412, ARMv7 (Cortex-A9) [1.6 GHz] 1x4x1
  - Linaro 12.11
- AMD Opteron™ 4274HE (Bulldozer/Valencia) [2.5 GHz] 2x8x1
  - Oracle Linux Server release 6.0 (64-bits)
- Intel® Xeon® CPU E5-2680 (Sandy Bridge) [2.70 GHz] 2x8x2
  - Oracle Linux Server release 6.3 (64-bits)

[1] System Under Test

Введение: JVM

- OpenJDK version “1.8.0-ea-lambda” build 83, 32-bits
- OpenJDK version “1.8.0-ea-lambda” build 83, 64-bits
- Java HotSpot(tm) Embedded “1.8.0-ea-b79”

`http://jdk8.java.net/lambda/`

Введение: Demo code

`https://github.com/kuksenko/quantum`

Введение: Demo code

https://github.com/kuksenko/quantum

- Required: JMH (Java Microbenchmark Harness)
  - http://openjdk.java.net/projects/code-tools/jmh/

Core

demo1: double sum

```
private double[] A = new double[2048];

@GenerateMicroBenchmark
public double test1() {
    double sum = 0.0;
    for (int i = 0; i < A.length; i++) {
        sum += A[i];
    }
    return sum;
}

@GenerateMicroBenchmark
public double testManualUnroll() {
    double sum = 0.0;
    for (int i = 0; i < A.length; i += 4) {
        sum += A[i] + A[i + 1] + A[i + 2] + A[i + 3];
    }
    return sum;
}
```

demo1: double sum

```
private double[] A = new double[2048];

@GenerateMicroBenchmark
public double test1() {
    double sum = 0.0;
    for (int i = 0; i < A.length; i++) {
        sum += A[i];
    }
    return sum;
}
```

327 ops/msec

```
@GenerateMicroBenchmark
public double testManualUnroll() {
    double sum = 0.0;
    for (int i = 0; i < A.length; i += 4) {
        sum += A[i] + A[i + 1] + A[i + 2] + A[i + 3];
    }
    return sum;
}
```

699 ops/msec

demo1: looking into asm, test1

```
loop: addsd   0x10(%edi,%eax,8),%xmm0
      addsd   0x18(%edi,%eax,8),%xmm0
      addsd   0x20(%edi,%eax,8),%xmm0
      addsd   0x28(%edi,%eax,8),%xmm0
      addsd   0x30(%edi,%eax,8),%xmm0
      addsd   0x38(%edi,%eax,8),%xmm0
      addsd   0x40(%edi,%eax,8),%xmm0
      addsd   0x48(%edi,%eax,8),%xmm0
      addsd   0x50(%edi,%eax,8),%xmm0
      addsd   0x58(%edi,%eax,8),%xmm0
      addsd   0x60(%edi,%eax,8),%xmm0
      addsd   0x68(%edi,%eax,8),%xmm0
      addsd   0x70(%edi,%eax,8),%xmm0
      addsd   0x78(%edi,%eax,8),%xmm0
      addsd   0x80(%edi,%eax,8),%xmm0
      addsd   0x88(%edi,%eax,8),%xmm0
      add     $0x10,%eax
      cmp     %ebx,%eax
      jl      loop:
```

demo1: looking into asm, testManualUnroll

```
loop: movsd   %xmm0,0x20(%esp)
      movsd   0x48(%eax,%edx,8),%xmm0
      movsd   %xmm0,(%esp)
      movsd   0x40(%eax,%edx,8),%xmm0
      movsd   %xmm0,0x8(%esp)
      movsd   0x78(%eax,%edx,8),%xmm0
      addsd   0x70(%eax,%edx,8),%xmm0
      movsd   0x80(%eax,%edx,8),%xmm1
      movsd   %xmm1,0x10(%esp)
      movsd   0x88(%eax,%edx,8),%xmm1
      movsd   %xmm1,0x18(%esp)
      movsd   0x38(%eax,%edx,8),%xmm4
      addsd   0x30(%eax,%edx,8),%xmm4
      movsd   0x58(%eax,%edx,8),%xmm5
      addsd   0x50(%eax,%edx,8),%xmm5
      movsd   0x28(%eax,%edx,8),%xmm1
      movsd   0x60(%eax,%edx,8),%xmm2
```

```
movsd   0x68(%eax,%edx,8),%xmm3
movsd   0x20(%eax,%edx,8),%xmm7
movsd   0x18(%eax,%edx,8),%xmm6
addsd   0x10(%eax,%edx,8),%xmm6
addsd   0x10(%esp),%xmm0
addsd   %xmm7,%xmm6
addsd   0x18(%esp),%xmm0
addsd   %xmm1,%xmm6
addsd   %xmm2,%xmm5
addsd   0x20(%esp),%xmm6
addsd   %xmm3,%xmm5
addsd   0x8(%esp),%xmm4
addsd   (%esp),%xmm4
addsd   %xmm4,%xmm6
addsd   %xmm6,%xmm5
addsd   %xmm5,%xmm0
add     $0x10,%edx
cmp     %ebx,%edx
jl      loop:
```

demo1: measure time

```
private double[] A = new double[2048];

@GenerateMicroBenchmark(BenchmarkType.AverageTimePerOp)
@OperationsPerInvocation(2048)
public double test1() {
    double sum = 0.0;
    for (int i = 0; i < A.length; i++) {
        sum += A[i];
    }
    return sum;
}

@GenerateMicroBenchmark(BenchmarkType.AverageTimePerOp)
@OperationsPerInvocation(2048)
public double testManualUnroll() {
    double sum = 0.0;
    for (int i = 0; i < A.length; i += 4) {
        sum += A[i] + A[i + 1] + A[i + 2] + A[i + 3];
    }
    return sum;
}
```

demo1: measure time

```
private double[] A = new double[2048];

@GenerateMicroBenchmark(BenchmarkType.AverageTimePerOp)
@OperationsPerInvocation(2048)
public double test1() {
    double sum = 0.0;
    for (int i = 0; i < A.length; i++) {
        sum += A[i];
    }
    return sum;
}
```

1.49 nsec/op

```
@GenerateMicroBenchmark(BenchmarkType.AverageTimePerOp)
@OperationsPerInvocation(2048)
public double testManualUnroll() {
    double sum = 0.0;
    for (int i = 0; i < A.length; i += 4) {
        sum += A[i] + A[i + 1] + A[i + 2] + A[i + 3];
    }
    return sum;
}
```

0.70 nsec/op

demo1: measure time

```java
private double[] A = new double[2048];

@GenerateMicroBenchmark(BenchmarkType.AverageTimePerOp)
@OperationsPerInvocation(2048)
public double test1() {
    double sum = 0.0;
    for (int i = 0; i < A.length; i++) {
        sum += A[i];
    }
    return sum;
}
```

1.49 nsec/op

CPI = $\sim$2.5

```java
@GenerateMicroBenchmark(BenchmarkType.AverageTimePerOp)
@OperationsPerInvocation(2048)
public double testManualUnroll() {
    double sum = 0.0;
    for (int i = 0; i < A.length; i += 4) {
        sum += A[i] + A[i + 1] + A[i + 2] + A[i + 3];
    }
    return sum;
}
```

0.70 nsec/op

CPI = $\sim$0.6

μArch: x86

CISC vs RISC

μArch: x86

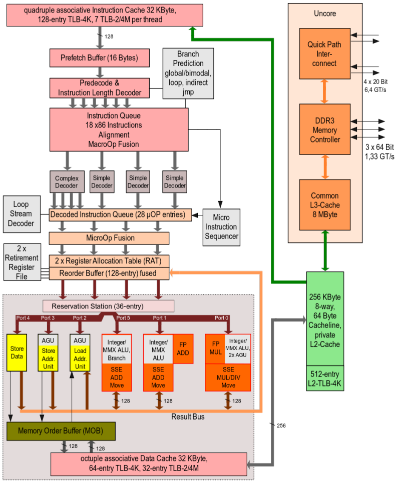
CISC and RISC

современный x86 процессор не тот, кем кажется

Все инструкции (CISC) динамически транслируются в RISC-like микрооперации (μops), которые собственно и исполняются.

μArch: Nehalem собственной персоной

Intel Nehalem microarchitecture



quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (36-entry)
Port 4
Port 3
Port 2
Port 5
Port 1
Port 0
Store Data
AGU
Store Addr. Unit
AGU
Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ MMX ALU
FP ADD
SSE ADD Move
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV Move
128
128
128
Result Bus
Memory Order Buffer (MOB)
128
128
octuple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
256
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1,33 GT/s
Common L3-Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K

GT/s: gigatransfers per second

μArch: упрощенная схема

μArch: looking into instruction tables

Instrustion	Latency	$\frac{1}{Throughput}$
ADDSP r,r	3	1
MULSD r,r	5	1
ADD/SUB r,r	1	0.33
MUL/IMUL r,r	3	1

demo1: test1, looking into asm again

```
loop: addsd   0x10(%edi,%eax,8),%xmm0
      addsd   0x18(%edi,%eax,8),%xmm0
      addsd   0x20(%edi,%eax,8),%xmm0
      addsd   0x28(%edi,%eax,8),%xmm0
      addsd   0x30(%edi,%eax,8),%xmm0
      addsd   0x38(%edi,%eax,8),%xmm0
      addsd   0x40(%edi,%eax,8),%xmm0
      addsd   0x48(%edi,%eax,8),%xmm0
      addsd   0x50(%edi,%eax,8),%xmm0
      addsd   0x58(%edi,%eax,8),%xmm0
      addsd   0x60(%edi,%eax,8),%xmm0
      addsd   0x68(%edi,%eax,8),%xmm0
      addsd   0x70(%edi,%eax,8),%xmm0
      addsd   0x78(%edi,%eax,8),%xmm0
      addsd   0x80(%edi,%eax,8),%xmm0
      addsd   0x88(%edi,%eax,8),%xmm0
      add     $0x10,%eax
      cmp     %ebx,%eax
      jl      loop:
```

1.49 nsec/op

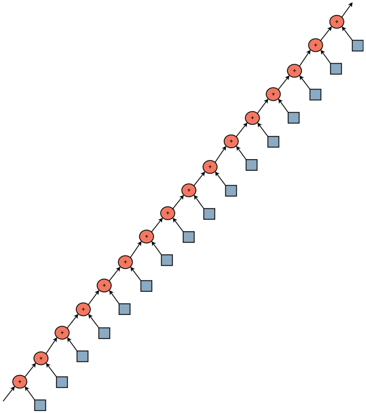
$\sim$ 3 clk/op

unroll by 16

19 insts

CPI $\sim$ 2.5

demo1: test1, other view



1.49 nsec/op

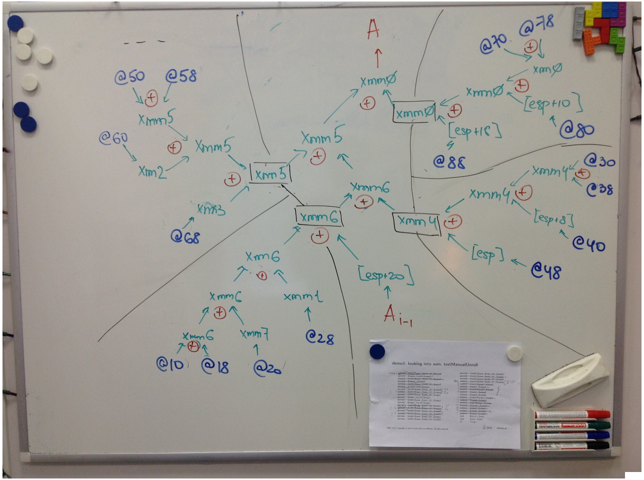
$\sim$ 3 clk/op

unroll by 16

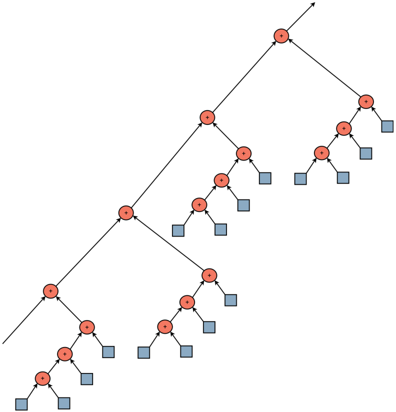
19 insts

CPI $\sim$ 2.5

demo1: testManualUnroll



demo1: testManualUnroll, other view



0.7 nsec/op
$\sim$ 1.4 clk/op
unroll by 4x4
36 insts
CPI $\sim$ 0.6

μArch: Dependences

~~Скорость работы~~ ILP многих программ ограничен естественными для них зависимостями по данным.

μArch: Dependences

~~Скорость работы~~ ILP многих программ ограничен естественными для них зависимостями по данным.

Что делать?

Break Dependency Chains!

demo1 back: рвем цепи “правильно”

demo1 back: рвем цепи “правильно”

```
...
for (int i = 0; i < A.length; i++) {
    sum += A[i];
}
return sum;
```

demo1 back: рвем цепи “правильно”

```
...
for (int i = 0; i < A.length; i++) {
    sum += A[i];
}
return sum;

...
for (int i = 0; i < A.length; i += 2) {
    sum0 += A[i];
    sum1 += A[i + 1];
}
return sum0 + sum1;
```

demo1 back: рвем цепи “правильно”

```
...
for (int i = 0; i < A.length; i++) {
    sum += A[i];
}
return sum;

...
for (int i = 0; i < A.length; i += 2) {
    sum0 += A[i];
    sum1 += A[i + 1];
}
return sum0 + sum1;

...
for (int i = 0; i < array.length; i += 4) {
    sum0 += A[i];
    sum1 += A[i + 1];
    sum2 += A[i + 2];
    sum3 += A[i + 3];
}
return (sum0 + sum1) + (sum2 + sum3);
```

demo1 back: double sum final results

	Nehalem	AMD	ARM
testManualUnroll	0.70	0.45	3.31
test1	1.49	1.50	6.60
test2	0.75	0.79	4.25
test4	0.51	0.43	4.25
test8	0.51	0.25	2.55

time, nsec/op

demo2: results

	Nehalem	AMD	ARM
DoubleMul.test1	3.89	2.52	8.17
DoubleMul.test2	3.59	2.37	4.25
DoubleMul.test4	0.73	0.49	3.15
DoubleMul.test8	0.61	0.30	2.53
IntMul.test1	1.49	1.16	10.04
IntMul.test2	0.75	0.75	7.38
IntMul.test4	0.57	0.67	4.64
IntSum.test1	0.51	0.32	8.92
IntSum.test2	0.51	0.48	6.12

time, nsec/op

Branches: to jump or not to jump

```java
public int absSumBranch(int a[]) {
    int sum = 0;
    for (int x : a) {
        if (x < 0) {
            sum -= x;
        } else {
            sum += x;
        }
    }
    return sum;
}
```

```
loop:  mov    0xc(%ecx,%ebp,4),%ebx
       test   %ebx,%ebx
       jl     L1
       add    %ebx,%eax
       jmp    L2
L1:    sub    %ebx,%eax
L2:    inc    %ebp
       cmp    %edx,%ebp
       jl     loop
```

Branches: to jump or not to jump

```java
public int absSumPredicated(int a[]) {
    int sum = 0;
    for (int x : a) {
        sum += Math.abs(x);
    }
    return sum;
}
```

```
loop:  mov    0xc(%ecx,%ebp,4),%ebx
       mov    %ebx,%esi
       neg    %esi
       test   %ebx,%ebx
       cmovl  %esi,%ebx
       add    %ebx,%eax
       inc    %ebp
       cmp    %edx,%ebp
       jl     Loop
```

demo3: results

Regular Pattern = $(+, -)^*$

	Nehalem	Bulldozer	Cortex-A9	Sandy Bridge
branch_regular	0.87	0.82	5.02	0.51
branch_shuffled	6.23	2.84	9.44	0.97
branch_sorted	0.89	0.99	5.02	0.59
predicated_regular	1.97	0.92	5.33	0.77
predicated_shuffled	1.97	0.96	9.3	0.77
predicated_sorted	1.97	0.96	5.65	0.77

time, nsec/op

demo3: results

Regular Pattern $= (+, +, -, +, -, -, +, -, -, +)^*$

	Nehalem	Bulldozer	Cortex-A9	Sandy Bridge
branch_regular	1.33	0.98	5.02	0.69
branch_shuffled	6.20	2.33	9.53	0.84
branch_sorted	0.89	0.95	5.03	0.59
predicated_regular	1.97	0.95	5.33	0.77
predicated_shuffled	1.97	0.94	9.38	0.77
predicated_sorted	1.97	0.91	5.65	0.77

time, nsec/op

demo4: && vs &

```java
public int countConditional(boolean[] f0, boolean[] f1) {
    int cnt = 0;
    for (int j = 0; j < SIZE; j++) {
        for (int i = 0; i < SIZE; i++) {
            if (f0[i] && f1[j]) {
                cnt++;
            }
        }
    }
    return cnt;
}
```

&&

shuffled	5.7 nsec/op
sorted	1.5 nsec/op

demo4: && vs &

```java
public int countLogical(boolean[] f0, boolean[] f1) {
    int cnt = 0;
    for (int j = 0; j < SIZE; j++) {
        for (int i = 0; i < SIZE; i++) {
            if (f0[i] & f1[j]) {
                cnt++;
            }
        }
    }
    return cnt;
}
```

&&

shuffled	5.7 nsec/op
sorted	1.5 nsec/op

&

shuffled	2.1 nsec/op
sorted	2.1 nsec/op

demo5: стоимость виртуального вызова

```
public interface I            { public int amount(); }
...
public class C0 implements I { public int amount(){ return 0; } }
public class C1 implements I { public int amount(){ return 1; } }
public class C2 implements I { public int amount(){ return 2; } }
public class C3 implements I { public int amount(){ return 3; } }
...
@GenerateMicroBenchmark(BenchmarkType.AverageTimePerOp)
@OperationsPerInvocation(SIZE)
public int sum(I[] a) {
    int s = 0;
    for (I i : a) {
        s += i.amount();
    }
    return s;
}
```

demo5: results

	1 target	2 targets	3 targets	4 targets
sorted	1.0	1.1	7.7	7.8
regular		1.0	7.7	19.0
shuffled		7.4	22.7	24.8

time, nsec/op

Not-a-Core

Not-a-Core: HW Multithreading

- Simultaneous multithreading, SMT
  e.g. Intel® Hyper-Threading Technology

- Fine-grained temporal multithreading
  e.g. CMT, Sun/Oracle ULTRASparc T1, T2, T3, T4, T5 ...

back to demo1: Execution Units Saturation

	1 thread	2 threads	4 threads
DoubleSum.test1	327	654	1279
DoubleSum.test2	647	1293	1865
DoubleSum.test4	957	1916	1866
DoubleSum.testManualUnroll	699	1398	1432

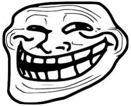
overall throughput, ops/msec

demo6: show

Внимание на экран!

demo6: show

Внимание на экран!



demo6: HDivs.heavy* results on Nehalem

1 thread

int	180
double	90

overall throughput, ops/μsec

	-cpu 1,3	-cpu 2,3	-cpu 3
(int, int)	(180, 180)	(90, 90)	(90, 90)
(double, double)	(90, 90)	(45, 45)	(45, 45)
(double, int)	(90, 180)	(81, 18)	(90, 45)

throughput, ops/μsec

demo6: HDivs.heavy* results on AMD

1 thread

int	128
double	306

overall throughput, ops/μsec

	-cpu 0,1	-cpu 0,2	-cpu 0,8	-cpu 0
(int, int)	(92, 92)	(127, 127)	(128, 132)	(63, 63)
(double, double)	(151, 153)	(304, 304)	(313, 314)	(154, 155)
(double, int)	(278, 119)	(290, 127)	(313, 129)	(122, 64)

throughput, ops/μsec

Заключение

Заключение: Учиться, учиться и учиться!

Читаем:

- “Computer Architecture: A Quantitative Approach”
  John L. Hennessy, David A. Patterson
- http://www.agner.org/optimize/
- Intel® 64 and IA-32 Architectures Software Developer Manuals
- Software Optimization Guide for AMD Family 10h Processors