“Quantum” Performance Effects:

Beyond The Core

ORACLE CODE

developer.oracle.com

Sergey Kuksenko
Java Platform Group, Oracle

October, 2018

Live for the Code

ORACLE

Safe Harbor Statement

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, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

About me

- Java/JVM Performance Engineer at Oracle, @since 2010
- Java/JVM Performance Engineer, @since 2005
- Java/JVM Engineer, @since 1996

System Under Test

- Intel® Core™ i5-5300U [2.3 GHz] 1x2x2
  - *μ*arch: Haswell
  - launched: Q1'2015s
- OS: Xubuntu 18.04 (64-bits) (4.15.0-36-generic)
- Java 8 (64-bits)
- Java 11 (64-bits)

Demo code

```
https://github.com/kuksenko/quantum2
```

Demo code

`https://github.com/kuksenko/quantum2`

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

Demo 1: How to copy 2 Mbytes.

Demo 1

```java
int[] a = new int[512*1024];
int[] b = new int[512*1024];

 @Benchmark
 public void arraycopy() {
     System.arraycopy(a, 0, b, 0, a.length);
 }

 @Benchmark
 public void reversecopy() {
     for(int i = a.length - 1; i >= 0; i--) {
         b[i] = a[i];
     }
 }
```

Demo 1

```
int[] a = new int[512*1024];
int[] b = new int[512*1024];

 @Benchmark
 public void arraycopy() {
     System.arraycopy(a, 0, b, 0, a.length);
 }

 @Benchmark
 public void reversecopy() {
     for(int i = a.length - 1; i >= 0; i--) {
         b[i] = a[i];
     }
 }
```

740 μs

??

300 μs

* Using Java 8

Conclusions?

- Oracle engineers - rubbish!
  - I know how to copy faster!

Conclusions?

- ~~Oracle engineers - rubbish!~~
  - ~~I know how to copy faster!~~

Shared results within team

- What I got:
  - `arraycopy vs reversecopy:` 740 vs 300 μs

ORACLE

Shared results within team

- What I got:
  - `arraycopy vs reversecopy:` 740 vs 300 μs
- What Bob got *(on some MacBook Pro)*:
  - `arraycopy vs reversecopy:` 190 vs 185 μs

Shared results within team

- What I got:
  - `arraycopy vs reversecopy:` **740 vs 300** μs
- What Bob got *(on some MacBook Pro)*:
  - `arraycopy vs reversecopy:` **190 vs 185** μs
- What Alice got *(she already migrated to JDK11)*:
  - `arraycopy vs reversecopy:` **270 vs 280** μs

Shared results within team

- What I got:
  - `arraycopy vs reversecopy:` **740 vs 300** μs
- What Bob got *(on some MacBook Pro)*:
  - `arraycopy vs reversecopy:` **190 vs 185** μs
- What Alice got *(she already migrated to JDK11)*:
  - `arraycopy vs reversecopy:` **270 vs 280** μs
- What if copy less data "2Mbytes - 32 bytes":
  - `arraycopy vs reversecopy:` **280 vs 720** μs

spent a billion on research

- MacOS doesn't support "Large Pages"!
  - Ubuntu - "Transparent Huge Pages"
- G1 is default GC since Java 9!
  - Java 8 default GC - "ParallelOld"

spent a billion on research

- MacOS doesn't support "Large Pages"!
  - Ubuntu - "Transparent Huge Pages"
- G1 is default GC since Java 9!
  - Java 8 default GC - "ParallelOld"

Conclusions:

- Large Pages - Rubbish!
- G1 GC - Cool!

spent a billion on research

- MacOS doesn't support "Large Pages"!
  - Ubuntu - "Transparent Huge Pages"
- G1 is default GC since Java 9!
  - Java 8 default GC - "ParallelOld"

Conclusions:

- ~~Large Pages – Rubbish!~~
- ~~G1 GC – Cool!~~

To Be Continued …

Demo 2: How many data?

Demo 2: The Last ~~Jedi~~ Refactoring

```java
public class MyData {

    private byte[] bytes;
    private int length;

    public MyData(int length) {
        this.bytes = new byte[length];
        this.length = length;
    }

    public int length() { return length; }

    public byte[] bytes() { return bytes; }
}
```

Demo 2: dataSize(**MyData**)

```java
MyData[] data = new MyData[256];

@Setup
public void setup() {
    Random rnd = new Random();
    Arrays.setAll(data, i -> new MyData(512 * 1024 + rnd.nextInt(64 * 1024)));
}

@Benchmark
public int dataSize() {
    int s = 0;
    for (MyData a : data) {
        s += a.length();
    }
    return s;
}
```

Demo 2: dataSize(byte[])

```
byte[][] data = new byte[256][];

@Setup
public void setup() {
    Random rnd = new Random();
    Arrays.setAll(data, i -> new byte[512 * 1024 + rnd.nextInt(64 * 1024)]);
}

@Benchmark
public int dataSize() {
    int s = 0;
    for (byte[] a : data) {
        s += a.length;
    }
    return s;
}
```

Demo 2: results (Java 8)

`DataSize(MyData)`	**145** ns
`DataSize(byte[])`	**200** ns

?

Demo 2: results (Java 8)

`DataSize(MyData)`	**145** ns
`DataSize(byte[])`	**200** ns

?

What if turn on G1? (`-XX:+UseG1GC`)

Demo 2: results (Java 8)

`DataSize(MyData)`	**145** ns
`DataSize(byte[])`	**200** ns

?

What if turn on G1? (`-XX:+UseG1GC`)

`DataSize(MyData)`	**145** ns
`DataSize(byte[])`	**13045** ns

???

Demo 2: results

What if turn off ”Large Pages”?

ParallelOld GC:	
`DataSize(MyData)`	**145** ns
`DataSize(byte[])`	**250** ns
G1 GC:	
`DataSize(MyData)`	**145** ns
`DataSize(byte[])`	**635** ns

??

Demo 2: Conclusions

Conclusions:

- Large Pages - Rubbish!
- G1 GC - Rubbish!

Demo 2: Conclusions

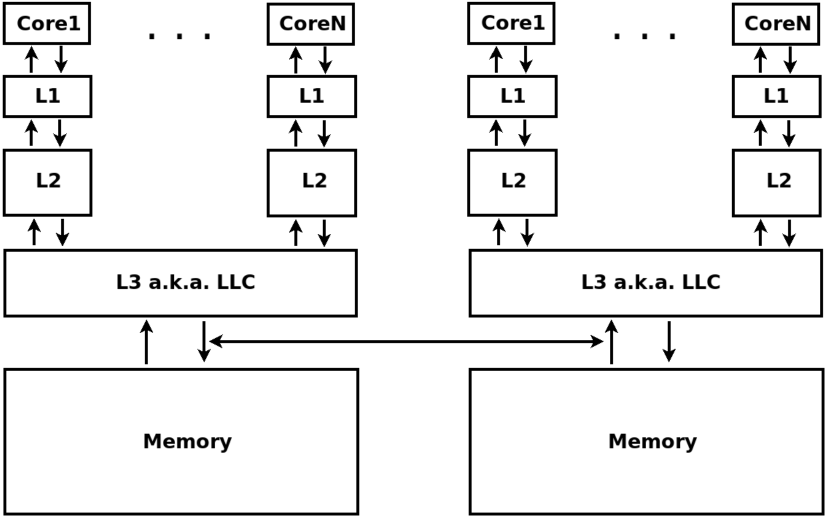
Conclusions:

- ~~Large Pages - Rubbish!~~
- ~~G1 GC - Rubbish!~~

To Be Continued …

Why we are here?

Caches, caches everywhere



Caches in numbers **(Intel Core i5-5300U)**

L1 - 32K, 8-way, latency: 4 cycles

L2 - 256K, 8-way, latency: 12 cycles

L3 - 3M, 12-way, latency: 35(and more) cycles

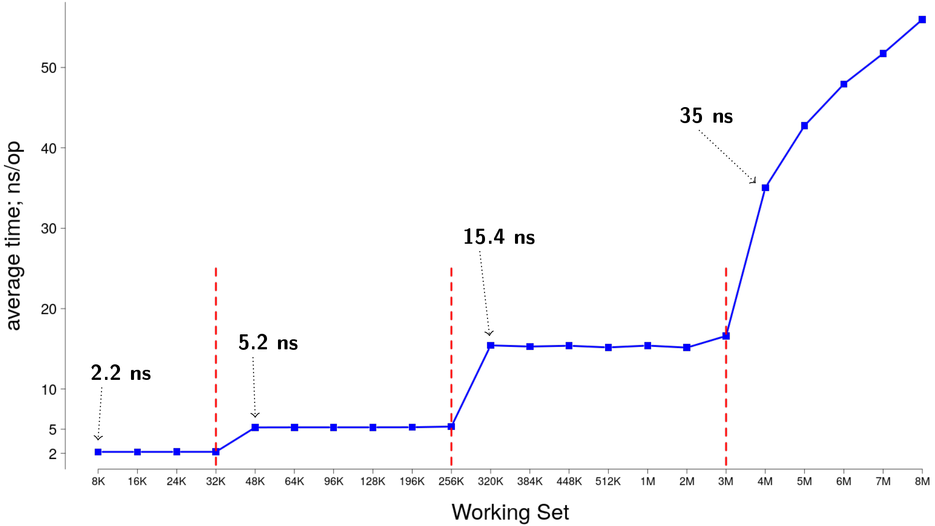
- cache line - 64 bytes

Demo 3: memory access cost.

Demo 3: walking on memory

```
Node root;

@Benchmark
@OperationsPerInvocation(COUNT)
public int walk() {
    return forward(root, COUNT);
}

public int forward(Node node, int cnt) {
    for(int i=0; i < cnt; i++) {
        node = node.next;
    }
    return node.value;
}
```

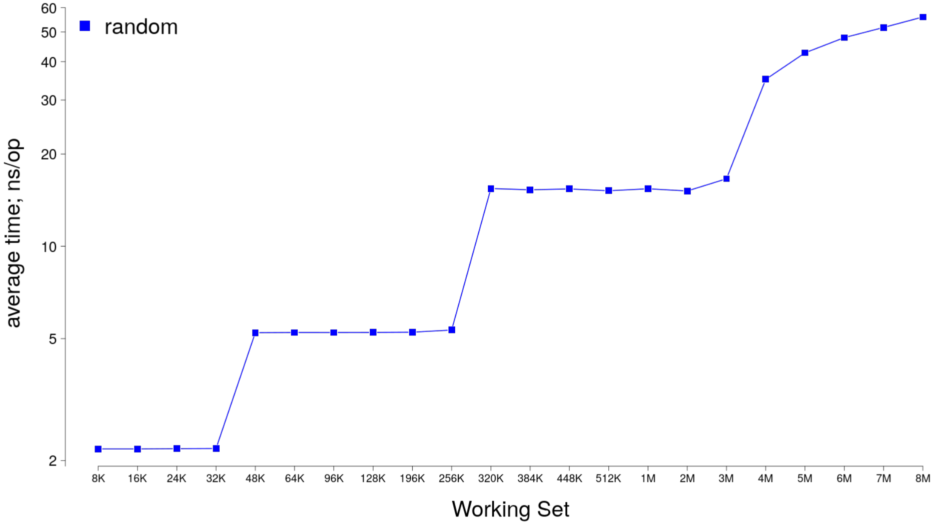
Demo 3: walking on memory



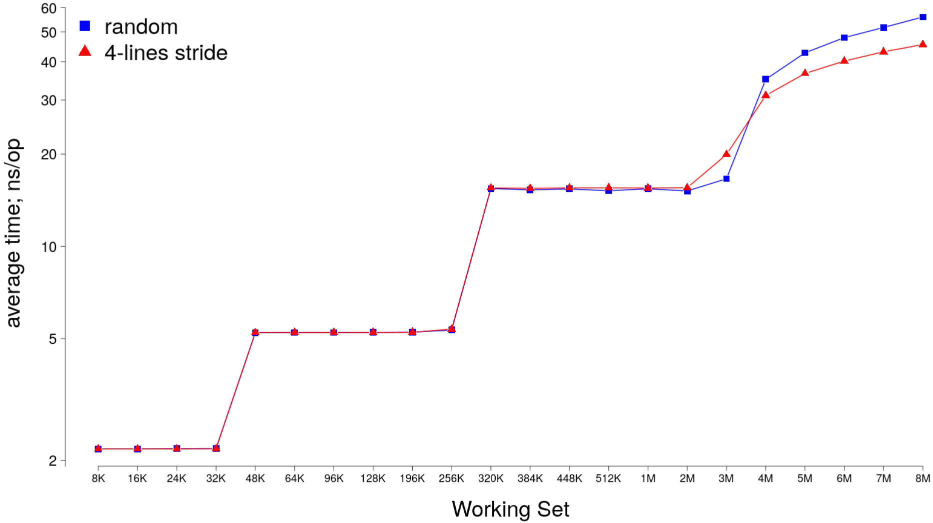
Demo 3: walking on memory

What about HW prefetching?

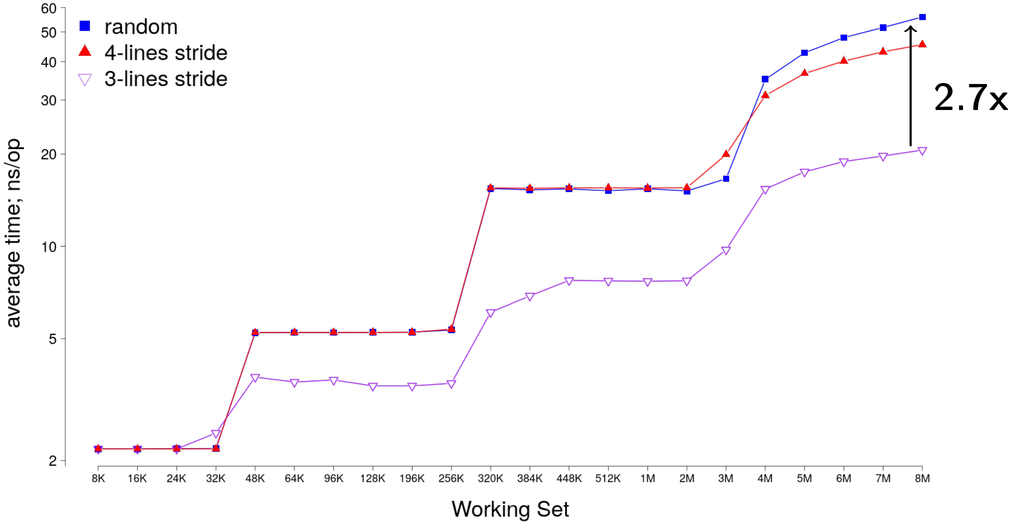
Demo 3: different mix



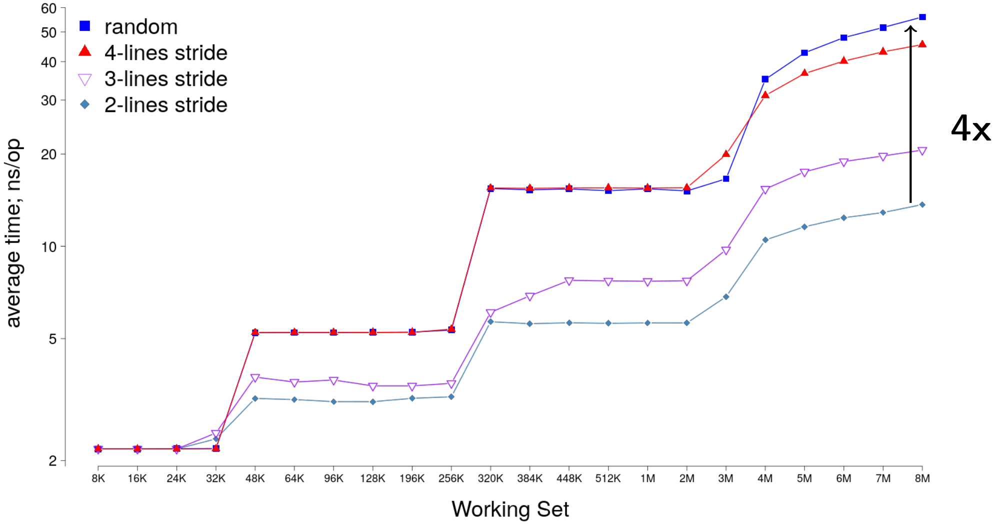
Demo 3: different mix



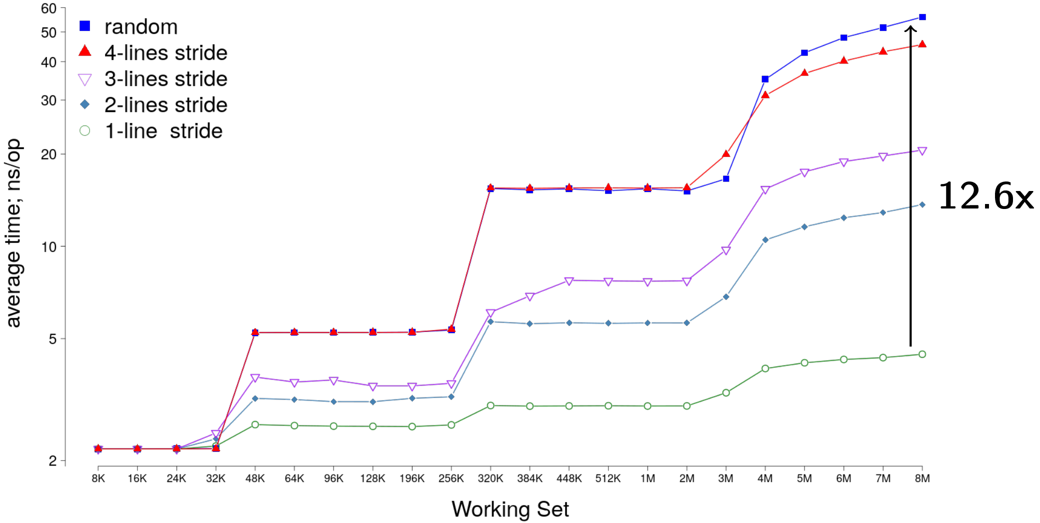
Demo 3: different mix



Demo 3: different mix



Demo 3: different mix



Demo 4: to split or not to split?

Demo 4: Good old Unsafe!

```
Unsafe UNSAFE;

long from;                     // page alignment

@Param({"-8", "-4", "-2", "0", "2", "4", "8" })
int offset;                    // offset in bytes

@Benchmark
public long getlong() {
    return UNSAFE.getLong(a, from + offset);
}

@Benchmark
public void putlong() {
    UNSAFE.putLong(a, from + offset, 42L);
}
```

Demo 4: Results

offset	*getlong*	*putlong*
-8	5.0	1.8
-4	19.1	17.8
0	5.0	1.8
60	5.2	2.5
64	5.0	1.8
	time, ns/op	

Demo 4: Results

unaligned data:

offset	*getlong*	*putlong*
-8	5.0	1.8
-4	19.1	17.8
0	5.0	1.8
60	5.2	2.5
64	5.0	1.8
	time, ns/op	

Page Split!

Line Split!

Demo 4: Misalignment

But wait!

Java doesn't have misaligned data!

Demo 4: Misalignment

But wait!
Java doesn't have misaligned data!

There are no misaligned data,
but there are misaligned operations.

Demo 4: Misalignment

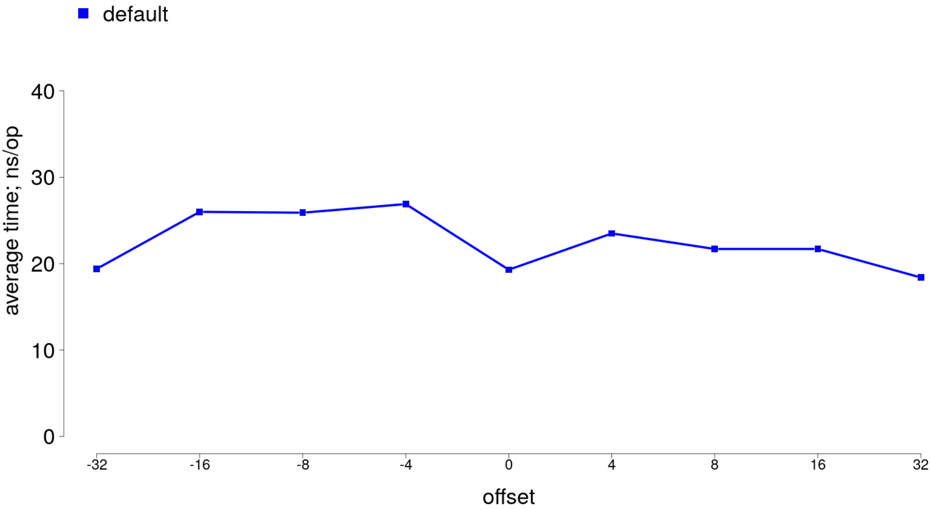
Java misaligned access:

- `Unsafe/VarHandle`
  - Buffers
  - Offheap
- SIMD instructions (SSE, AVX ...)
  - HotSpot intrinsics (`System.arraycopy, Arrays.fill ...`)
  - Automatic vectorization

Demo 4: Arrays.fill

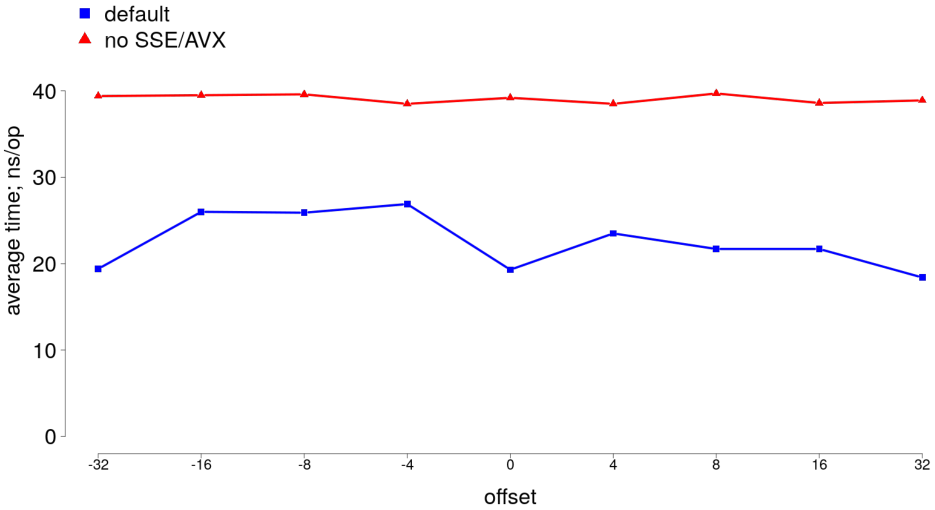
```
int from;         // alignment to page boundary

int size;

int offset;

byte[] a;

@Benchmark
public void fill() {
    Arrays.fill(a, from + offset, from + offset + size, (byte)42);
}
```

Demo 4: Arrays.fill, 512 bytes



ORACLE®

Demo 4: Arrays.fill, 512 bytes



Demo 5: upside down

Demo 5: matrix transpose

```
int size;

double[][] matrix = new double[size][size];

@Benchmark
public void transpose() {
    for (int i = 1; i < size; i++) {
        for (int j = 0; j < i; j++) {
            double tmp = matrix[i][j];
            matrix[i][j] = matrix[j][i];
            matrix[j][i] = tmp;
        }
    }
}
```

Demo 5: results (NxN)

N	
N+0	
N+1	
N+2	
N+3	**80** μs

Demo 5: results (NxN)

N	
N+0	
N+1	
N+2	**94** μs
N+3	**80** μs

Demo 5: results (NxN)

N	
N+0	**88** μs
N+1	
N+2	**94** μs
N+3	**80** μs

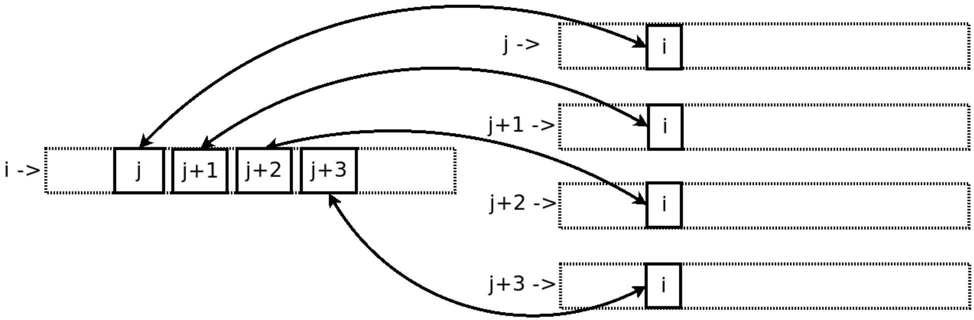
Demo 5: results (NxN)

N	
N+0	**88** μs
N+1	**350** μs
N+2	**94** μs
N+3	**80** μs

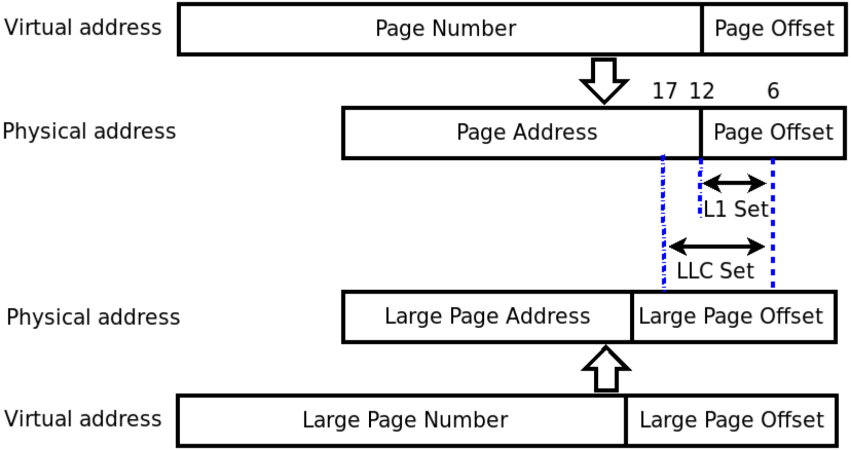
Demo 5: results (NxN)

N	
253	**88** μs
254	**350** μs
255	**94** μs
256	**80** μs

Demo 5: matrix transpose



Cache Associativity



Critical Stride

$$\langle \mathit{Critical\ Stride} \rangle = \frac{\langle \mathit{Cache\ Size} \rangle}{\langle \mathit{Associativity} \rangle}$$

- L1 (32K, 8-way) ⇒ 4K
- L2 (256K, 8-way) ⇒ 32K
- L3 (3M, 12-way) ⇒ 256K

Demo 2: How many data?(cont.)

”critical stride” hit

Let’s count:

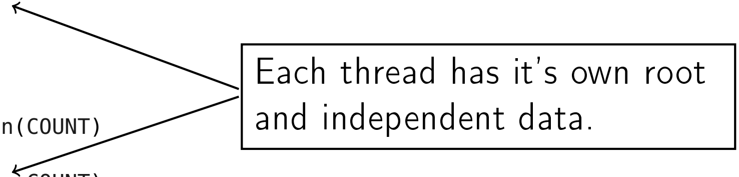
- G1 GC
  - all arrays are aligned to 1M (256K, 32K, 4K)
- ParallelOld GC
  - 256 arrays ⇒ 254 different ”index sets” в L3
  - 256 arrays ⇒ 251 different ”index sets” в L2
  - 256 arrays ⇒ 62 different ”index sets” в L1
  - number of hits to L1 index sets:

    10, 9, 8, 8, 8, 7, 7...

Demo 6: the rich get richer

Demo 6: Walking ~~dead~~ threads

```
@Benchmark
@Group("pair")
@OperationsPerInvocation(COUNT)
public int bob() {
    return forward(root, COUNT);
}

@Benchmark
@Group("pair")
@OperationsPerInvocation(COUNT)
public int alice() {
    return forward(root, COUNT);
}
```



Each thread has it's own root and independent data.

Demo 6: 128K per thread

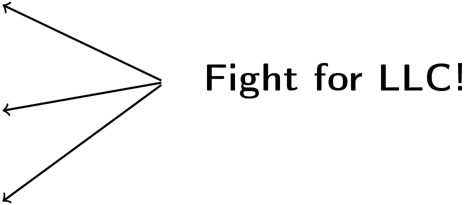
```
Iteration   1:
              bob:   5.246 ns/op
              alice: 5.241 ns/op

Iteration   2:
              bob:   5.254 ns/op
              alice: 5.272 ns/op

Iteration   3:
              bob:   5.233 ns/op
              alice: 5.244 ns/op

Iteration   4:
              bob:   5.244 ns/op
              alice: 5.232 ns/op
```

Demo 6: 1M per thread

```
Iteration   1:
              bob:   14.495 ns/op
              alice: 14.614 ns/op

Iteration   2:
              bob:   14.289 ns/op
              alice: 14.331 ns/op

Iteration   3:
              bob:   14.242 ns/op
              alice: 14.296 ns/op

Iteration   4:
              bob:   14.332 ns/op
              alice: 14.332 ns/op
```

Demo 6: 2M per thread

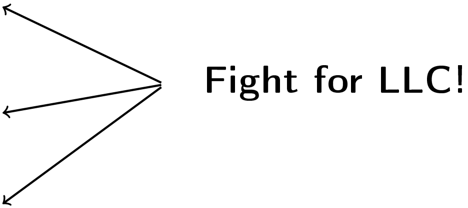
```
Iteration   1:
              bob:   17.199 ns/op
              alice: 48.845 ns/op

Iteration   2:
              bob:   46.777 ns/op
              alice: 20.850 ns/op

Iteration   3:
              bob:   17.046 ns/op
              alice: 48.686 ns/op

Iteration   4:
              bob:   46.422 ns/op
              alice: 20.704 ns/op
```



Fight for LLC!

Demo 6: 2M per thread

```
Iteration   1:
              bob:    17.199 ns/op
              alice:  48.845 ns/op

Iteration   2:
              bob:    46.777 ns/op
              alice:  20.850 ns/op

Iteration   3:
              bob:    17.046 ns/op
              alice:  48.686 ns/op

Iteration   4:
              bob:    46.422 ns/op
              alice:  20.704 ns/op
```



Fight for LLC!

$$\sim 1 \leq \frac{\langle Total\ Working\ Set \rangle}{\langle LLC\ size \rangle} \leq \sim 2.5$$

Demo 7: Bytes histogram

Demo 7: count bytes frequency

```
byte[] source;  // SIZE == 16 * K;

@Benchmark
public int[] count1() {
    int[] table = new int[256];
    for (byte v : source) {
        table[v & 0xFF]++;
    }
    return table;
}
```

Demo 7: count bytes frequency

```
byte[] source;  // SIZE == 16 * K;

@Benchmark
public int[] count1() {
    int[] table = new int[256];
    for (byte v : source) {
        table[v & 0xFF]++;
    }
    return table;
}
```

13.7 μs

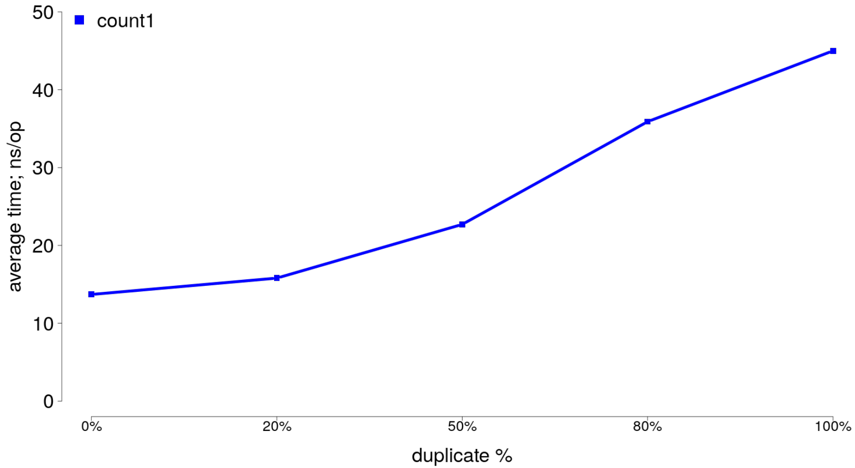
Demo 7: count bytes frequency

```
byte[] source;  // SIZE == 16 * K;

@Benchmark
public int[] count1() {
    int[] table = new int[256];
    for (byte v : source) {
        table[v & 0xFF]++;
    }
    return table;
}
```

$13.7\ \mu s$

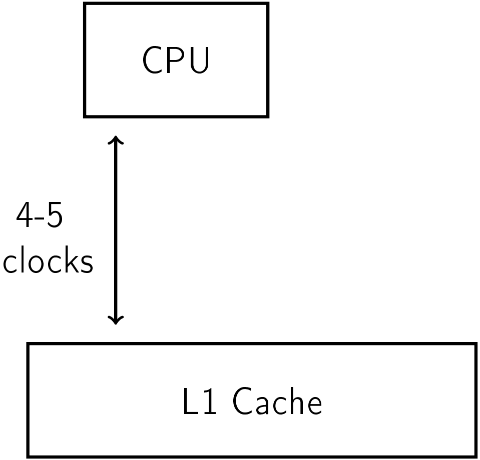
What if the data is unevenly distributed?

Results

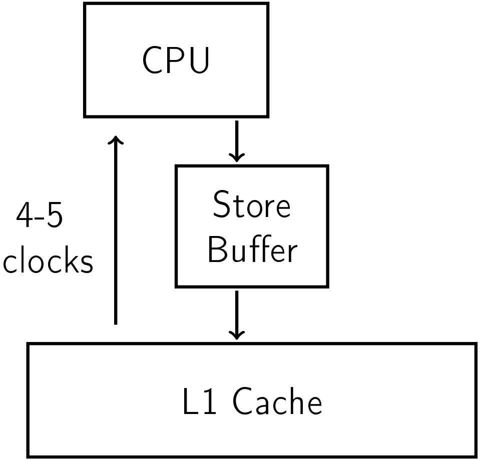


count1
50
40
30
20
10
0
average time; ns/op
0%
20%
50%
80%
100%
duplicate %

Store Buffer



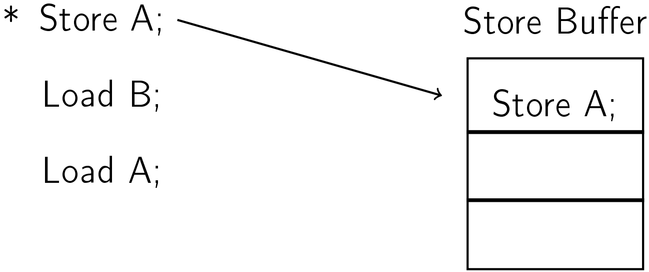
Store Buffer



Store Forwarding

```
Store A;

Load B;

Load A;
```

Store Forwarding



* Store A;

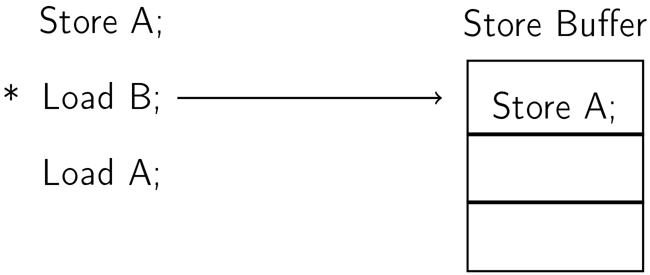
Load B;

Load A;

Store Buffer

Store A;

Store Forwarding



Store A;

* Load B;

Load A;

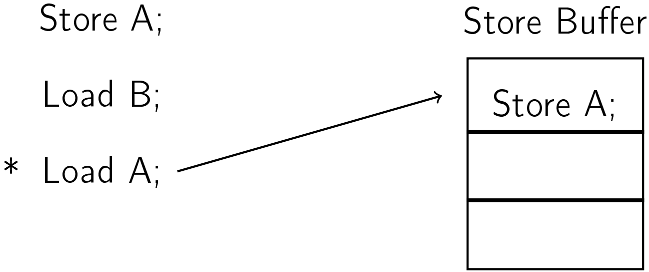
Store Buffer

Store A;

No “B” in Store Buffer

Execute!
even before Store

Store Forwarding



Store A;

Load B;

* Load A;

Store Buffer

Store A;

“A” exists in
Store Buffer

What to do?

Hit to “Store Buffer”

- Wait until “Store A” reaches L1 (expensive)
- Take value from Store Buffer (a.k.a. “Store Forwarding”)

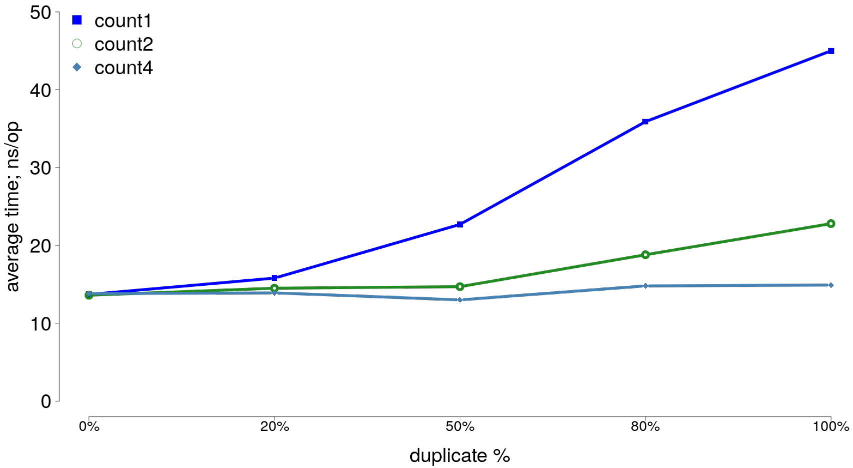
Let’s do this

```
@Benchmark
public int[] count2() {
    int[] table0 = new int[256];
    int[] table1 = new int[256];
    for (int i = 0; i < source.length; ) {
        table0[source[i++] & 0xFF]++;
        table1[source[i++] & 0xFF]++;
    }
    for (int i = 0; i < 256; i++) {
        table0[i] += table1[i];
    }
    return table0;
}
```

… and this

```
@Benchmark
public int[] count4() {
    int[] table0 = new int[256];
    int[] table1 = new int[256];
    int[] table2 = new int[256];
    int[] table3 = new int[256];
    for (int i = 0; i < source.length; ) {
        table0[source[i++] & 0xFF]++;
        table1[source[i++] & 0xFF]++;
        table2[source[i++] & 0xFF]++;
        table3[source[i++] & 0xFF]++;
    }
    for (int i = 0; i < 256; i++) {
        table0[i] += table1[i] + table2[i] + table3[i];
    }
    return table0;
}
```

Results



50
40
30
20
10
0
average time; ns/op
■ count1
○ count2
◆ count4
0%
20%
50%
80%
100%
duplicate %

Demo 8: bytes ⇔ int

Demo 8: bytes ⟺ int

```
ByteBuffer buf = ByteBuffer.allocateDirect(4);

@Benchmark
public int bytesToInt() {
    buf.put(0, b0);
    buf.put(1, b1);
    buf.put(2, b2);
    buf.put(3, b3);
    return buf.getInt(0);
}

@Benchmark
public int intToBytes() {
    buf.putInt(0, i0);
    return buf.get(0) +  buf.get(1) +
           buf.get(2) +  buf.get(3);
}
```

ORACLE

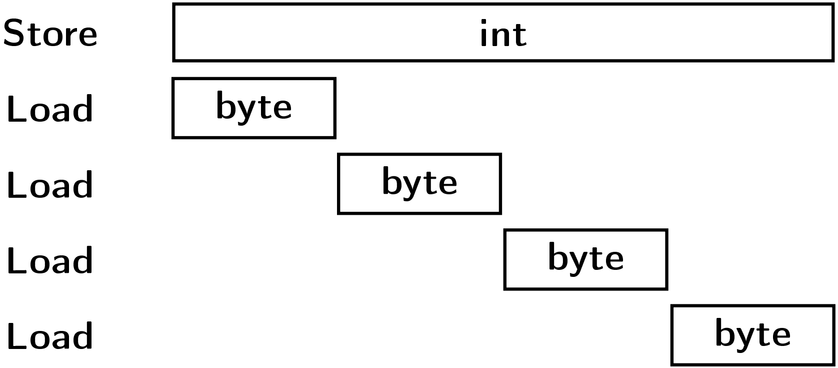
Demo 8: bytes ⇔ int

```
ByteBuffer buf = ByteBuffer.allocateDirect(4);

@Benchmark
public int bytesToInt() {
    buf.put(0, b0);
    buf.put(1, b1);
    buf.put(2, b2);
    buf.put(3, b3);
    return buf.getInt(0);
}

@Benchmark
public int intToBytes() {
    buf.putInt(0, i0);
    return buf.get(0) +  buf.get(1) +
           buf.get(2) +  buf.get(3);
}
```

13.2 ns

7.9 ns

Demo 8: Store Forwarding success



Store int

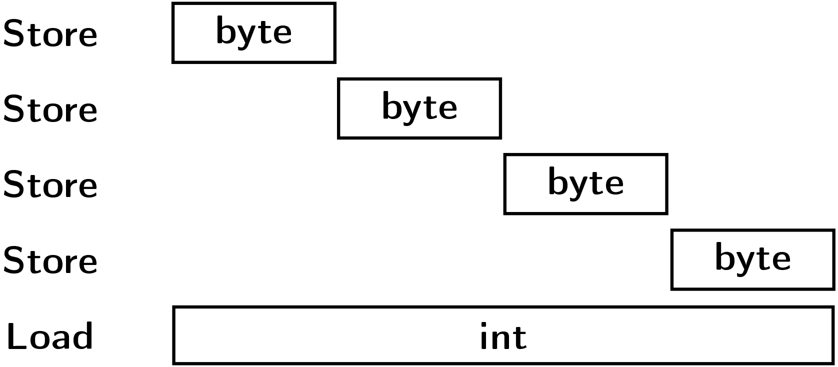
Load byte

Load byte

Load byte

Load byte

Demo 8: Store Forwarding fail



Store byte

Store byte

Store byte

Store byte

Load int

Demo 1: back to `arraycopy`

Demo 1: looking into asm

arraycopy

```
loop: vmovdqu -0x38(%rdi,%rdx,8),%ymm0
      vmovdqu %ymm0,-0x38(%rsi,%rdx,8)
      vmovdqu -0x18(%rdi,%rdx,8),%ymm1
      vmovdqu %ymm1,-0x18(%rsi,%rdx,8)
      add    $0x8,%rdx
      jle    loop
```

reversecopy

```
loop: vmovdqu -0xc(%r8,%rbx,4),%ymm0
      vmovdqu %ymm0,-0xc(%r10,%rbx,4)
      add    $0xfffffff8,%ebx
      cmp    $0x6,%ebx
      jg     loop
```

Demo 1: What about memory layout?

- ParallelOld GC
  - `AddressOf(a) == 0x76d890628`
  - `AddressOf(b) == 0x76da90638`
  - `AddressOf(b) – AddressOf(a) == 2Mb + 16`
- G1 GC
  - `AddressOf(a) == 0x6c7200000`
  - `AddressOf(b) == 0x6c7500000`
  - `AddressOf(b) – AddressOf(a) == 3Mb`

Demo 1: What about memory layout?

- ParallelOld GC
  - AddressOf(a) == 0x76d890628
  - AddressOf(b) == 0x76da90638
  - AddressOf(b) – AddressOf(a) == 2Mb + 16
- G1 GC
  - AddressOf(a) == 0x6c7200000
  - AddressOf(b) == 0x6c7500000
  - AddressOf(b) – AddressOf(a) == 3Mb

Demo 1: 4K-aliasing

HW uses **12** lower bits of address to detect Store Buffer conflicts.

- address difference 4K (12 bit)
- “Load” can't bypass “Store”
- “Store Forwarding” can't help - different addresses.

HW recovery:

- wait until “Store” is finished
- “clear pipeline” in case of speculation

Demo 1: arraycopy trace

```
Load  A;
Store B;

Load  A + 32;
Store B + 32;

Load  A + 64;
Store B + 64;

...
```

Demo 1: arraycopy trace

$$B == A + 2M + 16;$$

```
Load  A;
Store A + 2M + 16;

Load  A + 32;
Store A + 2M + 48;

Load  A + 64;
Store A + 2M + 80;

...
```

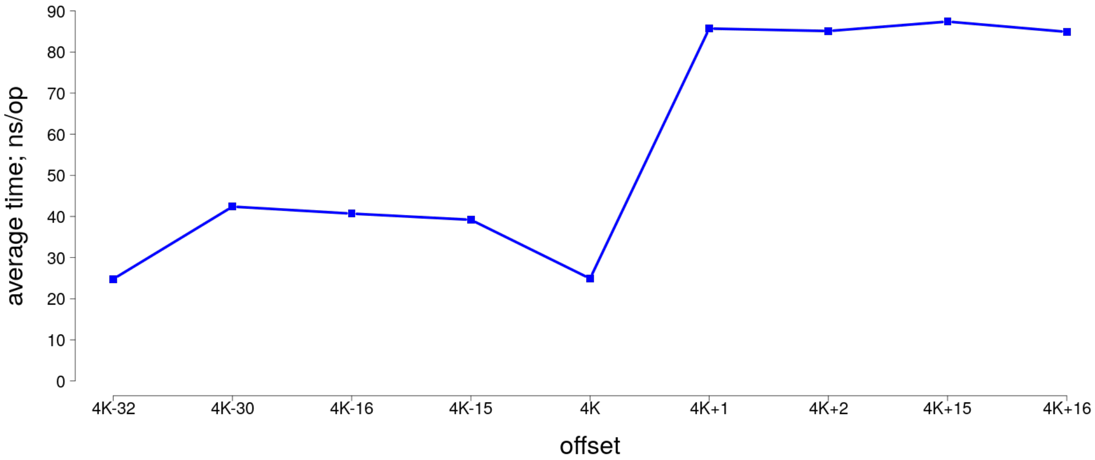
Demo 1: arraycopy trace

$$B == A + 2M + 16;$$

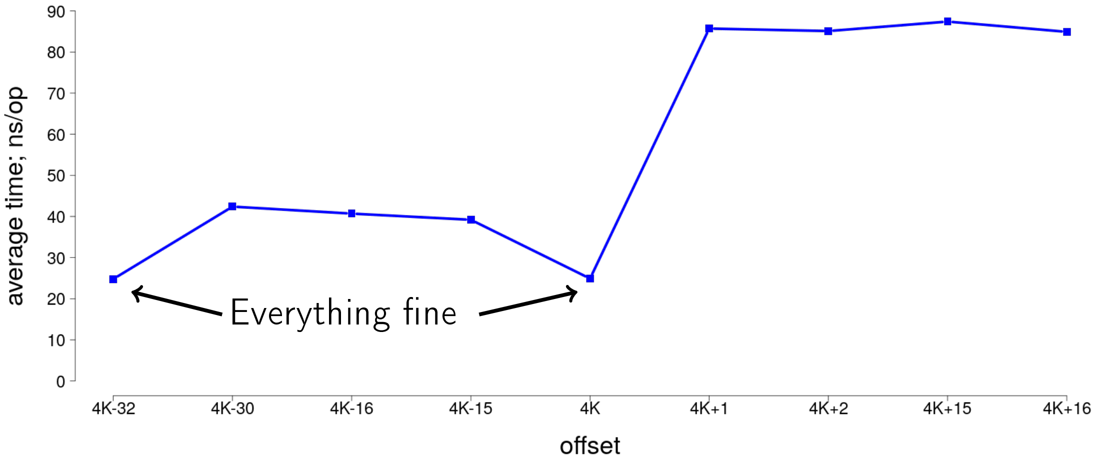
```
                                      address % 4096
Load  A;                                     0
Store A + 2M + 16;                          16

Load  A + 32;                               32
Store A + 2M + 48;                          48

Load  A + 64;                               64
Store A + 2M + 80;                          80

...
```

4K-aliasing

Demo 1: 1K copying

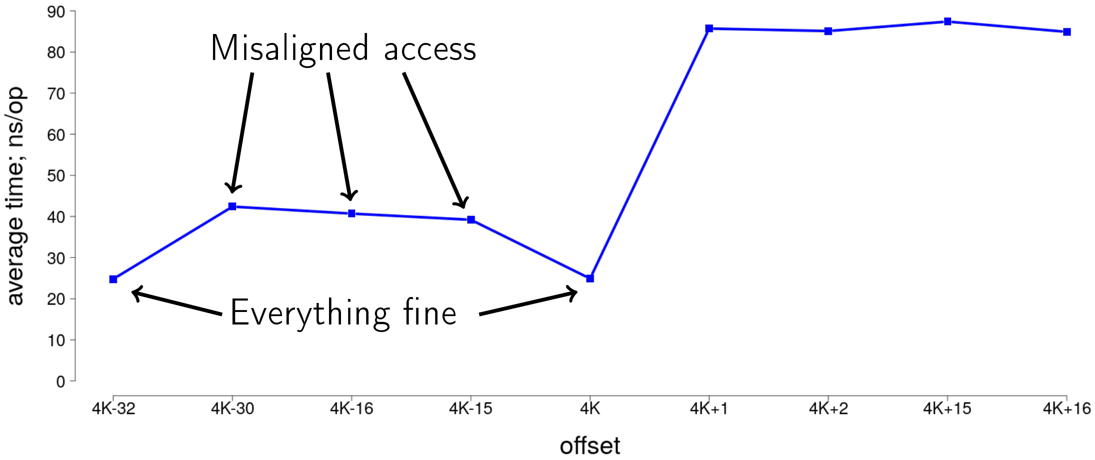


Demo 1: 1K copying



Everything fine

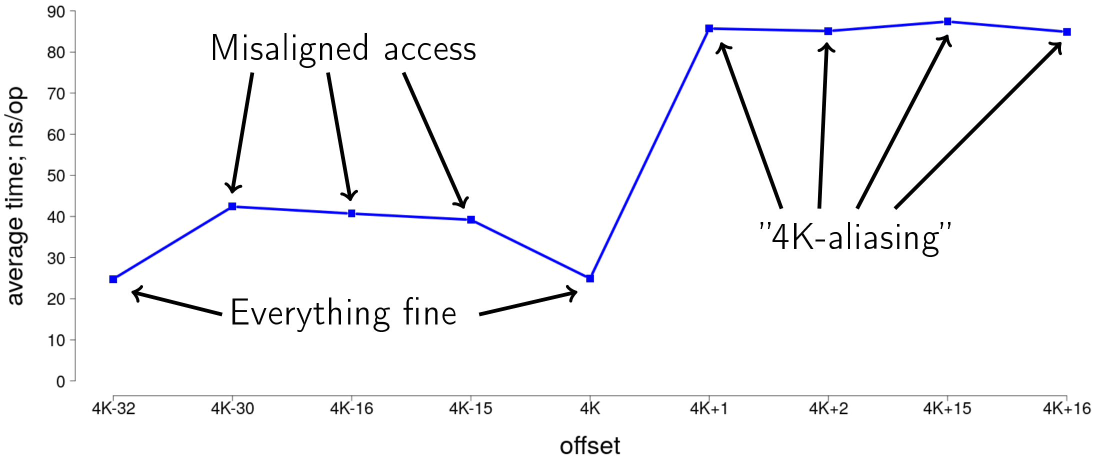
Demo 1: 1K copying



Misaligned access

Everything fine

Demo 1: 1K copying

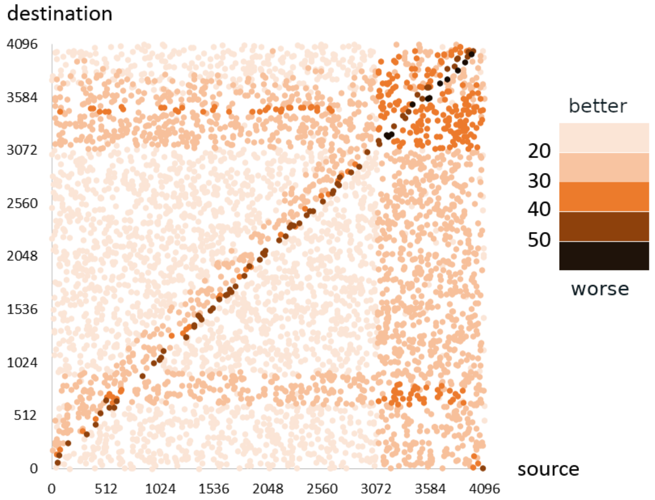


Misaligned access

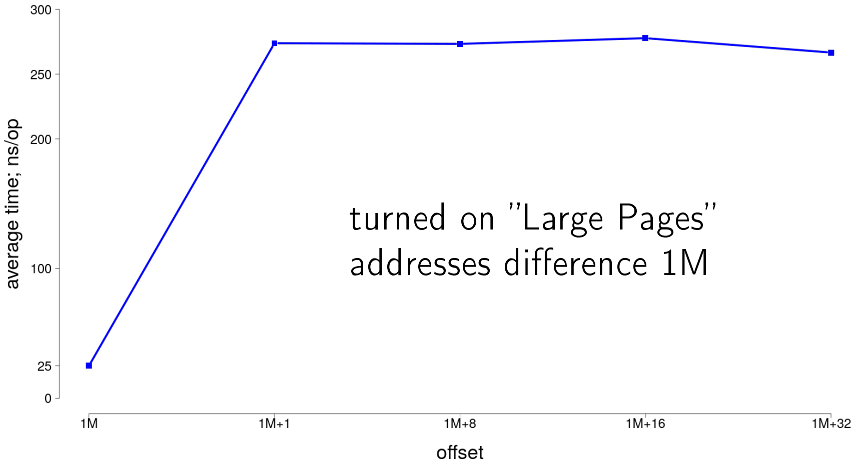
Everything fine

"4K-aliasing"

Demo 1: too many details



Demo 1: It’s not the end



turned on "Large Pages"
addresses difference 1M

Demo 1: All together

Data copying performance depends on how data located in memory

- Line split
- Page split
- 4K-aliasing
- ”1M & large pages aliasing” (still didn’t find an explanation)

ORACLE

Conclusion

To read!

- “What Every Programmer Should Know About Memory”
  Ulrich Drepper
- “Computer Architecture: A Quantitative Approach”
  John L. Hennessy, David A. Patterson
- CPU vendors documentation
- `http://www.agner.org/optimize/`
- etc.

Thank you!

Q & A ?