Updated 11 December, 2014

Branch mispredictions don't affect mergesort

Amr Elmasry[1], **Jyrki Katajainen**[2,3], **Max Stenmark**[3]

[1] Department of Computer Engineering and Systems, Alexandria University

[2] Department of Computer Science, University of Copenhagen

[3] Jyrki Katajainen and Company

These slides are available at http://www.cphstl.dk

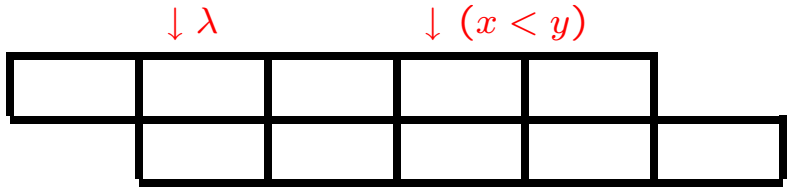
Problem: Expensive conditional branches

Code

if $(x < y)$ **goto** λ;
I_1;
I_2;
...
λ:
J_1;
J_2;
...

Pipelined execution

if $(x < y)$ **goto** λ;
I_1 or J_1**?**



Here instructions are carried out in five steps:

- Instruction fetch
- Register read
- Execution
- Data access
- Register write

History table $\longrightarrow$ prediction $\longrightarrow$ speculation $\overset{\text{if wrong}}{\longrightarrow}$ cycles wasted

Research question

Input: A random permutation of the integers $\{0, 1, \ldots, n-1\}$ in an array

Task: Sort these integers in increasing order

In-situ: Use $O(\lg n)$ words of extra memory

Question: Does there exist a faster in-situ sorting algorithm than quicksort with skewed pivots for this particular type of input?

[Kaligosi & Sanders 2006]

Related work

Mergesort: $O(n \lg n)$ work, $n \lg n + O(n)$ element comparisons, $O(n)$ extra space, and $O(n)$ branch mispredictions
[Mortensen 2001; Master's Thesis]

Samplesort: $O(n \lg n)$ work, $n \lg n + O(n)$ element comparisons, $O(n)$ extra space, and $O(n)$ branch mispredictions **on an average**
[Sanders & Winkel 2004]

Quicksort: A skewed pivot-selection strategy can lead to a better performance than the exact-median pivot-selection strategy
[Kaligosi & Sanders 2006]

Heapsort: $O(n \lg n)$ work, $2n \lg n + O(n)$ element comparisons, $O(1)$ extra space, and $O(1)$ branch mispredictions

Mergesort: $O(n \lg n)$ work, $n \lg n + O(n)$ element comparisons, $O(n)$ extra space, and $O(1)$ branch mispredictions
[Elmasry & Katajainen 2012]

Preliminary experiments

`std::sort` ≡ **introsort**

n	Time	Branches	Mispredicts
2^{10}	3.6	1.55	0.45
2^{15}	3.5	1.55	0.43
2^{20}	3.4	1.54	0.43
2^{25}	3.4	1.54	0.43

`std::stable_sort` ≡ **bottom-up mergesort**

n	Time	Branches	Mispredicts
2^{10}	3.7	2.11	0.14
2^{15}	3.6	2.06	0.09
2^{20}	3.7	2.05	0.07
2^{25}	3.7	2.04	0.05

All numbers are divided by $n \lg n$; time is in nanoseconds.

Janus: processor: Intel® Core™ i5-2520M CPU @ 2.50GHz × 4; word size: 64 bits; main memory: 3.8 GB; L3 cache: 3 MB, 12-way associative; cache line: 64 B. operating system: Ubuntu 12.04; Linux kernel: 3.2.0-24-generic; compiler: `g++` version 4.6.3; compiler options: `-O3 -Wall`

Secret behind mergesort

Element comparisons are decoupled from conditional branches!

C++ code

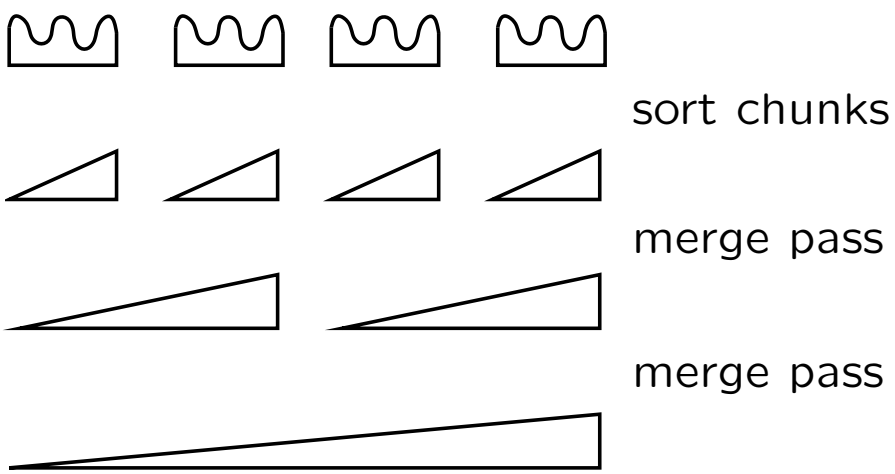
```
1   if (less(*q, *p)) {
2     *r = *q;
3     ++q;
4   }
5   else {
6     *r = *p;
7     ++p;
8   }
9   ++r;
```

Assembly-language code

```
 1  movl   (%eax), %edx
 2  leal   4(%eax), %edi
 3  movl   (%ebx), %ecx
 4  leal   4(%ebx), %ebp
 5  cmpl   %ecx, %edx
 6  cmovge %ecx, %edx
 7  cmovge %ebp, %ebx
 8  cmovl  %edi, %eax
 9  movl   %edx, (%esi)
10  addl   $4, %esi
```

Aha! Conditional move `if (c) x = y`

Tuned mergesort



opt$_1$: Instead of using insertionsort, sort each chunk of size four with straight-line code that has no conditional branches.

opt$_2$: Unroll the main loop in the merge routine by moving four elements to the output area in each iteration.

Our first result

Branch mispredictions don't affect mergesort!

Mergesort (opt_1)

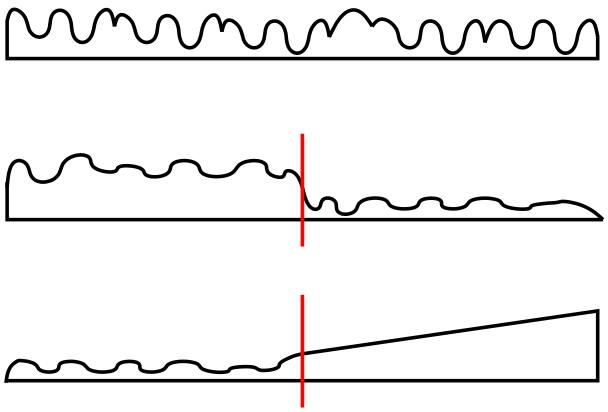
n	Time	Branches	Mispredicts
2^{10}	2.9	1.70	0.04
2^{15}	3.0	1.80	0.03
2^{20}	3.1	1.85	0.02
2^{25}	3.2	1.88	0.02

Mergesort (opt_1 & opt_2)

n	Time	Branches	Mispredicts
2^{10}	3.0	0.85	0.06
2^{15}	3.0	0.73	0.03
2^{20}	3.2	0.67	0.03
2^{25}	3.3	0.64	0.02

NB: # branches $< n \lg n$

Tuned in-situ mergesort



median finding

partitioning

mergesort

recur until $\leq n/\lg(2+n)$ elements

```
 1 template <typename iterator, typename comparator>
 2 void sort(iterator p, iterator r, comparator less) {
 3   typedef typename std::iterator_traits<iterator>::difference_type index;
 4   index n = r - p;
 5   index threshold = n / ilogb(2 + n);
 6   while (n > threshold) {
 7     iterator q_1 = p + n / 2;
 8     iterator q_2 = r - n / 2;
 9     converse_relation<comparator> greater(less);
10     std::nth_element(p, q_1, r, greater);
11     mergesort(p, q_1, q_2, less);
12     r = q_1;
13     n = r - p;
14   }
15   std::sort(p, r, less);
16 }
```

[Katajainen, Pasanen & Teuhola 1996]

Our second result

Branch mispredictions don't affect in-situ mergesort!

In-place `std::stable_sort`

n	Time	Branches	Mispredicts
2^{10}	17.3	9.0	2.05
2^{15}	20.6	10.9	2.36
2^{20}	22.7	12.2	2.51
2^{25}	24.5	13.3	2.60

Tuned in-situ mergesort

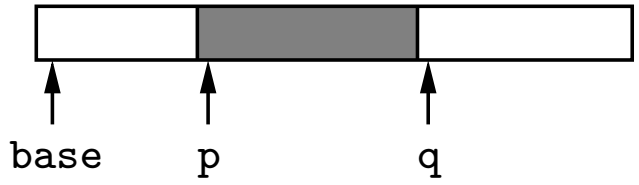
n	Time	Branches	Mispredicts
2^{10}	4.2	1.98	0.26
2^{15}	4.2	1.95	0.15
2^{20}	4.2	1.94	0.11
2^{25}	4.3	1.93	0.08

NB: The library routine runs in $O(n(\lg n)^2)$ time.

NB: Sorting is no more stable with our routine.

Our third result

We could reproduce the results of Kaligosi & Sanders for quicksort.



1) `pivot = (p - base) + α * (q - p)`

2) Hoare's partitioning

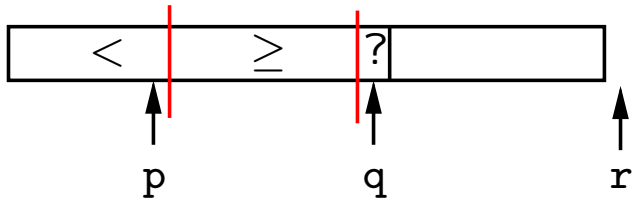
Quicksort $\alpha = \frac{1}{2}$

n	Time	Branches	Mispredicts
2^{10}	3.6	1.33	0.45
2^{15}	3.5	1.30	0.47
2^{20}	3.6	1.29	0.48
2^{25}	3.6	1.28	0.48

Quicksort $\alpha = \frac{1}{5}$

n	Time	Branches	Mispredicts
2^{10}	3.0	1.56	0.37
2^{15}	3.0	1.58	0.36
2^{20}	2.9	1.58	0.35
2^{25}	3.0	1.59	0.34

Tuned quicksort (not in the proceedings)



Lomuto's partitioning

```
 1 q = p;
 2 --p;
 3 while (q < r) {
 4   x = *q;
 5   if (less(x, pivot)) {
 6     ++p;
 7     *q = *p;
 8     *p = x;
 9   }
10   ++q;
11 }
12 return ++p;
```

Lean version

```
 1 q = p;
 2 while (q < r && ! less(*q, pivot)) {
 3   ++q;
 4 }
 5 if (q == r) {
 6   return p;
 7 }
 8 std::iter_swap(p, q);
 9 ++q;
10 while (q < r) {
11   x = *q;
12   smaller = less(x, pivot);
13   p += smaller;
14   delta = smaller * (q - p);
15   s = p + delta;
16   t = q - delta;
17   *s = *p;
18   *t = x;
19   ++q;
20 }
21 return ++p;
```

Aha! A mixture of `int`s and `bool`s

Our fourth result

Branch mispredictions don't affect quicksort!

Quicksort with skewed pivots

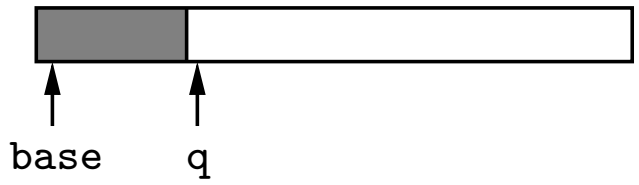
n	Time	Branches	Mispredicts
2^{10}	3.0	1.56	0.37
2^{15}	3.0	1.58	0.36
2^{20}	2.9	1.58	0.35
2^{25}	3.0	1.59	0.34

Tuned quicksort

n	Time	Branches	Mispredicts
2^{10}	2.7	1.23	0.14
2^{15}	2.6	1.21	0.09
2^{20}	2.6	1.20	0.07
2^{25}	2.6	1.19	0.05

NB: In tuned quicksort, the median-of-three pivot-selection strategy is used.

Curiosity: median-for-free in-situ mergesort



1) `median = 0.5 * (q - base)`

2) Lean partitioning

n	Time	Branches	Mispredicts
2^{10}	3.4	1.56	0.06
2^{15}	3.9	1.71	0.05
2^{20}	4.1	1.76	0.03
2^{25}	4.3	1.82	0.03

Curiosity: median-for-free quicksort

1) `pivot = (p - base) + 0.5 * (q - p)`

2) Lean partitioning

n	Time	Branches	Mispredicts
2^{10}	2.5	1.21	0.13
2^{15}	2.3	1.14	0.09
2^{20}	2.3	1.10	0.07
2^{25}	2.3	1.08	0.05

Results of the race

4. tuned mergesort★★★
5. `std::sort` ≡ introsort★★★
6. `std::stable_sort`★★★
7. median-for-free in-situ mergesort★★★
8. tuned in-situ mergesort★★★★
9. in-place `std::stable_sort`★★

★ general purpose
★ in-situ
★ $O(n \lg n)$ worst case
★ $O(n)$ branch mispredictions

Teaching quicksort

You can tell

- the truth of Kaligosi & Sanders [2006] **or**
- our truth **or**
- both **or**
- the incorrect old story **or**
- something else. **What?**

Discussion

1) A conditional move in C++ was not always converted to a conditional-move instruction by the compiler.

2) Assembly-language code written by us was slower than the code generated by the compiler.

3) Our results are architecture-dependent. **How much?**

4) In instruction-rate calculations we should have taken into account the linear terms, too.

5) Referee: You should have also measured the number of clock cycles used. Agreed!

Advice for practitioners

- Write programs as before if speed is not primary concern.
- Keep easy-to-predict branches since they have small overhead on modern processors.
- Eliminate hard-to-predict branches if the elimination will not cause too much overhead.