Branchless Search Programs

Amr Elmasry[1]

Jyrki Katajainen[2,3]

[1] Alexandria University
[2] University of Copenhagen
[3] Jyrki Katajainen and Company

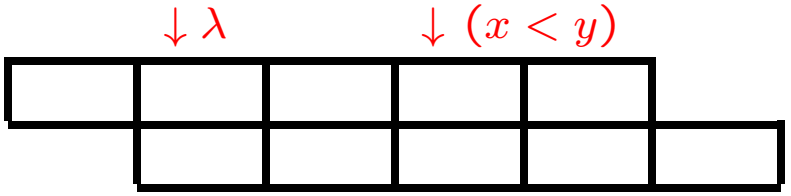
Cause of the troubles: Conditional branches

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 $\rightarrow$ prediction $\rightarrow$ speculation $\overset{\text{if wrong}}{\rightarrow}$ cycles wasted

Symptoms

Quicksort

A skewed pivot-selection strategy can lead to a better performance than the exact-median pivot-selection strategy

[Kaligosi & Sanders 2006]

Search trees

Skewed search trees can perform better than perfectly balanced search trees

[Brodal & Moruz 2006]

And, yes, we have been able to reproduce these results!

Proposed medication

Quicksort

– select the $\lfloor \alpha N \rfloor$th smallest element as the pivot (e.g. for $\alpha = \frac{1}{5}$).

[Kaligosi & Sanders 2006]

– select the median as the pivot

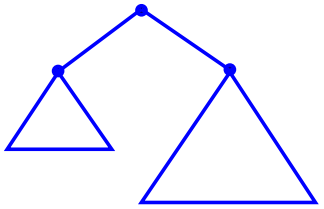
– program the partitioning routine without **if** statements

[Elmasry et al. 2012]

Search trees

– build skewed search trees ($weight(\texttt{(*x).left()}) = \lfloor \alpha weight(\texttt{x}) \rfloor$, $0 < \alpha \leq \frac{1}{2}$)

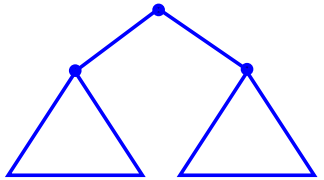
[Brodal & Moruz 2006]



– build balanced search trees

– program the search routine without **if** statements

[this paper]



Research question

Data: A collection of N integers

Queries: Support **random** membership searches as efficiently as possible.

Updates: None; the collection is **static**.

Question: What is the best data representation in this particular case?

[Brodal & Moruz 2006]

Branchless search program

Original

```
1 bool is_member(V const & v) {
2   N* y = nullptr; // candidate node
3   N* x = root; // current node
4   while (x != nullptr) {
5     if (less(v, (*x).element())) {
6       x = (*x).left();
7     }
8     else {
9       y = x;
10      x = (*x).right();
11    }
12  }
13  if (y == nullptr || less((*y). ↩
      element(), v)) {
14    return false;
15  }
16  return true;
17 }
```

[Bottenbruch 1962]

Branch optimized

```
1 N* choose(bool c, N* x, N* y) {
2   return (N*)((char*) y + c * ((char ↩
      *) x - (char*) y));
3 }
4
5 bool is_member(V const & v) {
6   N* y = nullptr; // candidate node
7   N* x = root; // current node
8   while (x != nullptr) {
9     bool c = less(v, (*x).element());
10    y = choose(c, y, x);
11    x = choose(c, (*x).left(), (*x). ↩
        right());
12  }
13  if (y == nullptr || less((*y). ↩
      element(), v)) {
14    return false;
15  }
16  return true;
17 }
```

Experimental environment for sanity checks

Processor

Intel® Core™ i5-2520M CPU @ 2.50GHz × 4

Memory system

12-way-associative L3 cache: 3 MB

cache lines: 64 B

main memory: 3.8 GB

Operating system

Ubuntu 12.04 (Linux kernel 3.2.0-29-generic)

Compiler

`g++` compiler (`gcc` version 4.6.3) with optimization `-O3`

Profiler

`valgrind` simulators (version 3.7.0)

Sorted array vs. red-black tree

Standard benchmark

r random `is_member` queries, $r = 10^6$

Input data

All elements are of type `int`

Reported value

Measurement result divided by $r \times \lg N$

Search time [ns]

N	Sorted array	Red-black tree
2^{10}	6.5	5.6
2^{15}	8.5	11.3
2^{20}	14.5	36.1
2^{25}	34.1	67.0

Performance of branchless search

Theorem.

- N elements
- $\sim \lg N$ element comparisons
- $\sim \lg N$ branches ($\lessdot$)
- $O(1)$ mispredictions

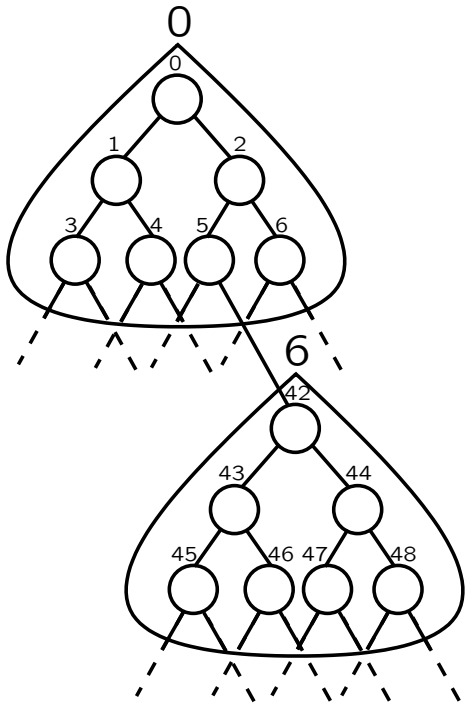
Search time [ns]

N	Skewed Original $\alpha = \frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	Skewed Branchless $\alpha = \frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$
2^{10}	5.6	5.8	5.7	5.1	6.6	6.3
2^{15}	10.7	10.5	11.3	7.4	10.4	12.2
2^{20}	41.3	40.4	44.2	38.1	42.5	51.7
2^{25}	79.0	81.7	91.2	75.3	85.5	96.7

Branch behaviour

N	Balanced $\alpha = \frac{1}{2}$ Original $\lessdot$	Mispred.	Skewed $\alpha = \frac{1}{3}$ Original $\lessdot$	Mispred.	Skewed $\alpha = \frac{1}{4}$ Original $\lessdot$	Mispred.	Balanced $\alpha = \frac{1}{2}$ Branchless $\lessdot$	Mispred.
2^{10}	2.20	0.61	2.34	0.57	2.57	0.52	1.20	0.10
2^{15}	2.13	0.57	2.28	0.52	2.53	0.46	1.13	0.07
2^{20}	2.10	0.55	2.26	0.50	2.52	0.44	1.10	0.05
2^{25}	2.08	0.54	2.24	0.49	2.51	0.42	1.08	0.04

Local search tree



$F = 7$

[Oksanen & Malmi 1995]

Pointer-based representation

$left\text{-}child(x)$
    return $(*x).left()$

Performance of local search trees

Theorem.

- N elements
- $\sim \lg N$ element comparisons
- $\sim \lg N$ branches
- $O(1)$ mispredictions
- $O(\log_B N)$ cache I/Os

(B: # elements in a cache line)

Search time [ns]

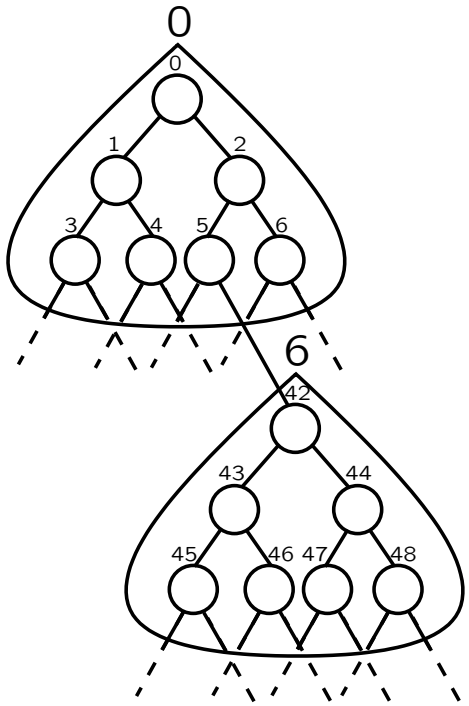
N	Red-black	Local	
		Orig.	Branchless
2^{10}	5.6	5.0	5.9
2^{15}	11.3	8.1	6.0
2^{20}	36.1	21.4	20.7
2^{25}	67.0	32.8	35.1

Cache behaviour

All values are divided by $r \times \log_B n$ ($B = 16$ in our test)

N	Red-black			Local		
	Refs.	I/Os	Misses	Refs.	I/Os	Misses
2^{10}	9.39	0.60	0.00	10.16	0.10	0.00
2^{15}	9.00	2.73	0.00	9.19	2.43	0.00
2^{20}	8.77	3.52	1.55	8.60	3.27	1.32
2^{25}	8.64	3.93	2.45	8.84	3.77	2.27

Implicit local search tree



$F = 7$

No pointers

$left\text{-}child(i)$
  $j = i/F$
  if $i < \lfloor F/2 \rfloor + j * F$
    return $2 * i - j * F + 1$
  else
    return $F * (2 * i - (1 - F) * j - F + 2)$

Performance of implicit local search trees

Theorem.

- N elements
- $\sim \lg N$ element comparisons
- $\sim \lg N$ branches
- $O(1)$ mispredictions
- $\sim \log_B N$ cache I/Os

(B: # elements in a cache line)

Search time [ns]; $F = 15$

N	Sorted array	Implicit local
2^{10}	6.5	15.0
2^{15}	8.5	15.0
2^{20}	14.5	16.3
2^{25}	34.1	20.1

Cache behaviour

All values are divided by $r \times \log_B N$ ($B = 16$ in our test)

N	Sorted array			Implicit local		
	Refs.	I/Os	Misses	Refs.	I/Os	Misses
2^{10}	7.40	0.00	0.00	10.56	0.00	0.00
2^{15}	6.93	2.00	0.00	9.46	0.39	0.00
2^{20}	6.70	3.20	0.73	8.80	0.81	0.12
2^{25}	6.56	3.37	3.01	9.00	1.01	0.51

Performance of branchless programs

Conditional branches

$$\sim 2 \lg N \longrightarrow \sim \lg N$$

Branch mispredictions

$$\sim 0.5 \lg N \longrightarrow O(1)$$

Search time [ns]; $F = 15$

N	Sorted array		Implicit local	
	Orig.	Branchless	Orig.	Branchless
2^{10}	6.5	5.8	15.0	15.0
2^{15}	8.5	6.9	15.0	14.2
2^{20}	14.5	21.1	16.3	15.6
2^{25}	34.1	48.5	20.1	22.8

Branch behaviour

N	Sorted array Original		Sorted array Branchless		Implicit local Original		Implicit local Branchless	
	⩽	Mispred.	⩽	Mispred.	⩽	Mispred.	⩽	Mispred.
2^{10}	2.20	0.62	1.20	0.10	3.56	0.89	1.32	0.11
2^{15}	2.07	0.57	1.13	0.07	3.21	0.86	1.12	0.07
2^{20}	2.05	0.55	1.10	0.05	3.05	0.84	1.05	0.05
2^{25}	2.04	0.54	1.08	0.04	3.18	0.82	1.09	0.04

Unrolling the loop

```
1 bool is_member(V const & v) {
2   N* y = nullptr; // candidate node
3   N* x = root; // current node
4   bool c;
5   switch (height) {
6   case 31:
7     c = less(v, (*x).element());
8     y = choose(c, y, x);
9     x = choose(c, (*x).left(), (*x).right());
  ⋮
125 case 1:
126   c = less(v, (*x).element());
127   y = choose(c, y, x);
128   x = choose(c, (*x).left(), (*x).right());
129 default:
130   c = (x == nullptr) || less(v, (*x).element());
131   y = choose(c, y, x);
132 }
133 if ((y == nullptr) || less((*y).element(), v)) {
134   return false;
135 }
136 return true;
137 }
```

Theorem.

- N elements
- $\sim \lg N$ element comparisons
- $O(1)$ branches
- $O(1)$ mispredictions

No improvement in practice

Conclusions

- Branch optimization is only effective for small problem instances.
- There is no reason to remove easy-to-predict conditional branches.
- It would be cool if branch optimization was done automatically by the compiler.
- Is branch optimization important in industrial applications?
- How architecture-dependent are the results?
- There are still phenomena that we do not understand—please explain.

Williams' heapsort

```
if (less(a[j], a[j + 1]) {
  j = j + 1;
}
```

$\longrightarrow$

```
j = j + less(a[j], a[j + 1]);
```

Running time/$N \lg N$ [ns]

N	Original	Optimized
2^{10}	5.7	3.8
2^{15}	5.6	4.2
2^{20}	6.7	7.2
2^{25}	12.3	25.8

The field is now open

Considered by us

- binary heaps
- weak heaps
- search trees
- heapsort
- mergesort
- quicksort

Relevant papers

- Amr, Jyrki: Lean programs, branch mispredictions, and sorting, FUN 2012
- Amr, Jyrki, Max: Branch mispredictions don't affect mergesort, SEA 2012
- Amr, Jyrki: Branchless search programs, SEA 2013
- Amr, Jyrki, Stefan: Weak heaps engineered, Journal of Discrete Algorithms (to appear)