London, 21 July 1999

Performance engineering case study: heap construction

Jesper Bojesen[1]

Jyrki Katajainen[2]

Maz Spork[3]

[1]UNI•C, Danish Computing Centre for Research and Education
Technical University of Denmark

[2]Department of Computing
University of Copenhagen

[3]Visionik A/S, Copenhagen

Keywords: hacking, code tuning, memory tuning, performance engineering

Problem: Heap Construction

Input: A sequence $\mathbf{a}[0\,..\,N)$ of N items specified by two random access iterators, and a strict weak ordering $\lessdot$ on the set of items.

Task: Put the sequence in **heap order**, i.e., in such a form that

$$\forall i \in [1\,..\,N)\colon \mathbf{a}[i] \lessdot \mathbf{a}[\lfloor (i-1)/2 \rfloor] \text{ is false.}$$

Requirement: Use only $O(1)$ extra space if possible.

Our Result

Our program is two times faster than Floyd's.

– Items: 4-byte integers.

– The cache performance is improved by a factor $c \cdot \log_2 B$, where c is a positive constant and B the size of the cache line measured in items.

Casts Gave 20% of the Speed

Original code:

```
int n;
node hole, last_leaf;
hole = hole + (hole - base);
n = n + 1;
last_leaf = base + n;
```

Generated code:

```
int n;
node hole, last_leaf;
hole = hole + ((int) hole - (int) base) / sizeof(item));
n = n + 1;
node last_leaf = (node) ((int) base + n * sizeof(item));
```

Modified code:

```
int n;
node hole, last_leaf;
hole = (node) ((int) hole) + (int) hole - (int) base);
n = n + sizeof(item);
node last_leaf = (node) ((int) base + n);
```

The overhead caused by memory delays can be significant.

Sum of N Integers

Input: An array of N integers.

Output: The sum of these integers (modulo word size).

```
unsigned int sum(unsigned int* a,
                 unsigned int mask, unsigned int step) {
  // Preconditions: mask is a power of 2 minus one
  //                gcd(step, mask + 1) = 1
  unsigned int i;
  unsigned int result = a[0];
  for (i = step; i != 0; i = (i + step) & mask)
    result += a[i];
  return result;
}
```

Theory: For $\texttt{mask} = N - 1$ the running time of this program is $\Theta(N)$ if $\gcd(\texttt{step}, N) = 1$.

Practice: Caching Effects

```
IBM 66 MHz Power2
Stride   1: size 524288 Time(s): 1.75000e+00
Stride 631: size 524288 Time(s): 1.28800e+01
Ratio = 7.4

IBM 160 MHz P2SC
Stride   1: size 524288 Time(s): 7.60000e-01
Stride 631: size 524288 Time(s): 9.95000e+00
Ratio = 13.1

IBM 330 MHz PPC (F50)
Stride   1: size 524288 Time(s): 1.11000e+00
Stride 631: size 524288 Time(s): 1.42000e+01
Ratio = 12.8

IBM 160 MHz PPC (E30)
Stride   1: size 524288 Time(s): 2.41000e+00
Stride 631: size 524288 Time(s): 1.83100e+01
Ratio = 7.6

IBM 34H 42 MHz Power
Stride   1: size 524288 Time(s): 5.04000e+00
Stride 631: size 524288 Time(s): 3.03300e+01
Ratio = 6.0

DECstation 400 MHz
Stride   1: size 524288 Time(s): 4.83314e-01
Stride 631: size 524288 Time(s): 2.76656e+00
Ratio = 5.7
```

```
DEC DS20 500 MHz
Stride   1: size 4194304 Time(s): 1.20162e+01
Stride 631: size 4194304 Time(s): 8.61799e+01
Ratio = 7.2

HP 9000/735 99 MHz
Stride   1: size 524288 Time(s): 3.96000e+00
Stride 631: size 524288 Time(s): 1.65300e+01
Ratio = 4.2

HP 9000/C160 160 MHz
Stride   1: size 524288 Time(s): 1.88000e+00
Stride 631: size 524288 Time(s): 1.59100e+01
Ratio = 8.5

SGI PowerChallenge 90 MHz
Stride   1: size 2097152 Time(s): 1.85000e+00
Stride 631: size 2097152 Time(s): 1.91600e+01
Ratio = 10.4

CRAY C90
Stride   1: size 524288 Time(s): 2.05746e+00
Stride 631: size 524288 Time(s): 2.04852e+00
Ratio = 1.0
```

Recursion can be faster than iteration.

Floyd's bottom-up method:

```
template<class node, class ordering>
void make_heap(node first, node last, ordering less) {
    for (node top = parent(last - 1); top >= first; --top)
        item_down(first, top, last, less);
}
```

Top-down method:

```
template <class node, class ordering>
void make_heap(node first, node top, node last,
               ordering less) {
    if (right_child(top) <= parent(last - 1))
        make_heap(first, right_child(top), last, less);
    if (left_child(top) <= parent(last - 1))
        make_heap(first, left_child(top), last, less);
    item_down(first, top, last, less);
}

template <class node, class ordering>
void make_heap(node first, node last, ordering less) {
    if (last - first < 2) return;
    make_heap(first, first, last, less);
}
```

Execution times on a 500 MHz Compaq:

```
Floyd's bottom-up method          Top-down method
N: 1000000                        N: 1000000
Time (s): 0.122217                Time (s): 0.0944407
```

Any Experience in C++?

```
toke> g++ -O4 heap_construction.cc
heap_construction.cc: In function
‘void make_heap<int *, less<int>>(int *, int *,
struct less<int>)’:
heap_construction.cc:381:   instantiated from here
heap_construction.cc:234: Internal compiler error 980519.
heap_construction.cc:234: Please submit a full bug report
to ‘egcs-bugs@egcs.cygnus.com’.
heap_construction.cc:234: See <http://egcs.cygnus.com/
faq.html#bugreport> for details.
```

Divide-and-conquer algorithms lead to fast programs when run on a computer with a hierarchical memory.

Current computers have many parallel components which makes the prediction of execution times difficult.

Pure C Machine Model

```
register const c = ...;
register int x, y, z;
register int* p;
```

Memory-to-register assignments "`x = *p;`"

Register-to-memory assignments "`*p = x;`"

Constant-to-register assignments "`x = c;`"

Register-to-register assignments "`x = y;`"

Unary arithmetic assignments "`x =` $\ominus$`y;`", where $\ominus \in \{$`-`, `~`, `!`$\}$

Binary arithmetic assignments "`x = y` $\oplus$ `z;`", where $\oplus \in \{$`+`, `-`, `*`, `/`, `&`, `|`, `^`, `<<`, `>>`$\}$

Conditional branches "`if (x` $\triangleleft$ `y) goto` λ`;`", where $\triangleleft \in \{$`<`, `<=`, `==`, `>`, `>=`$\}$

Unconditional branches "`goto` λ`;`"

Penalty Cost Model

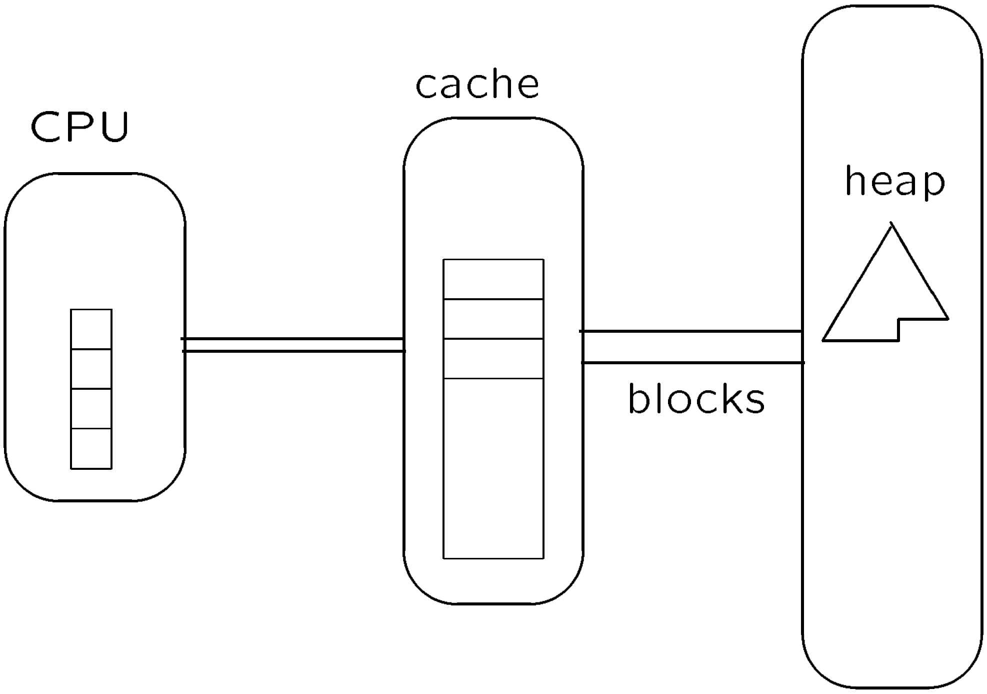
1. The cost of all pure C operations is assumed to be the same τ.

2. Each load and store operation has an extra penalty $\tau_i^\star$ if it incurs a miss at level i of the memory hierarchy, $i \in \{1, 2, \ldots, \ell\}$.

Now, if a program executes n pure C operations and incurs m_i misses at memory level i, $i \in \{1, 2, \ldots, \ell\}$, its execution time might be expressed as the sum

$$n \cdot \tau + \sum_{i=1}^{\ell} m_i \cdot \tau_i^\star \, .$$

Cache Parameters



B**:** the size of blocks (in items)

M**:** the capacity of the cache (in items)

Cache-line replacement strategy: LRU

Associativity: Fully associative

Alignment: Heap is B**-aligned**

Our Results Revisited

Repeated insertion:

τ-cost: $5N \log_2 N + 8N + O(1)$

$\tau^\star$-cost: $5\frac{N}{B} + o(N)$ if $\frac{M}{B} \geq 5 \log_2 N$

lower bound $\frac{3}{2}\frac{N}{B} - \frac{M}{B}$

Repeated merging:

(as proposed by Floyd)

τ-cost: $12.5N + O(1)$

$\tau^\star$-cost: lower bound $\frac{1}{2}\frac{N}{B} \log_2 B$ if $\frac{M}{B} \leq \frac{1}{4}\frac{N}{B}$

Repeated merging:

(a "new" program)

τ-cost: $13N + O(\log_2 N)$

$\tau^\star$-cost: close to $\frac{N}{B}$ if $\frac{M}{B} \gg 2B$

Layerwise construction:

τ-cost: $O(N)$

$\tau^\star$-cost: $\exists c \geq 1, c\frac{N}{B}$ if $\frac{M}{B} \leq O(1)$

Experimental Results