How to Write Fast Code
18-645, spring 2008
27th Lecture, Apr. 23rd

Instructor: Markus Püschel
TAs: Srinivas Chellappa (Vas) and Frédéric de Mesmay (Fred)
Course Evaluations

- Are open now
- Please fill it out
Research Project

- Project expectations
- Paper templates and instructions on the website
- Poster template will be uploaded tonight

- Today
- Papers due (6 pm)
- Last class: poster session
  Scaife Hall
  5:30 – 8:30 pm
- Due:
  - Final papers
  - Final code
Today

- Sorting, part 2
  (Example of a non-numerical problem)
Sorting large arrays

Quicksort

(Multiway-) Mergesort

- Temporal and spatial locality
- Simple, array based (no complicated data structures)

Sorting small arrays

Insertion sort

• Good for “almost sorted” list

Sorting Networks

• Suitable for unrolling
Radix Sort

- **Basic idea**

- **Second iteration:**
  - Sort for next digit
  - \( D \rightarrow S \)

- **Discussion:** blackboard

Cache-Conscious (CC) Radix Sort  (Jimenez et al. 2003)

- **Basic idea: Blackboard**
- **Pseudocode (Bucket = array)**

\[
\text{CC-Radix}(\text{bucket}, b) \\
\text{begin} \\
\quad \text{if fits\_in\_cache\_L_i}(\text{bucket}) \text{ then} \\
\quad \quad \text{Radix\_sort}(\text{bucket}, b) \\
\text{else} \\
\quad \text{sub\_buckets} = \text{Reverse\_sorting}(\text{bucket}, b) \\
\quad \text{for each sub\_bucket in sub\_buckets} \\
\quad \quad \text{CC-Radix}(\text{sub\_bucket}, b - b_r) \\
\quad \text{endfor} \\
\text{endif} \\
\text{end}
\]

**Choose to avoid TLB misses**

**1 step w.r.t. most significant bits b_r**

CC Radix Sort: Results

<table>
<thead>
<tr>
<th>1M keys</th>
<th>CSE</th>
<th>L2 misses</th>
<th>TLB misses</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC-Radix sort</td>
<td>89.0</td>
<td>305,242</td>
<td>1,762</td>
</tr>
<tr>
<td>EBT (11)</td>
<td>132.0</td>
<td>434,000</td>
<td>121,000</td>
</tr>
<tr>
<td>PLSB (11)</td>
<td>159.6</td>
<td>744,638</td>
<td>10,278</td>
</tr>
<tr>
<td>LSB sort (6)</td>
<td>163.0</td>
<td>779,329</td>
<td>172,000</td>
</tr>
<tr>
<td>MSB-Radix sort</td>
<td>203.2</td>
<td>796,891</td>
<td>974,008</td>
</tr>
<tr>
<td>Radix sort</td>
<td>282.0</td>
<td>502,883</td>
<td>2,607,023</td>
</tr>
</tbody>
</table>

Evaluation: Quicksort, Mergesort, CC-Radix

So everything solved?

Performance versus Standard Deviation

- Performance may depend on
  - the distribution of input data
  - the computing platform

CC-Radix: Smaller stddev = data distributes over fewer buckets = more steps to fit into cache

Adaptive Sorting (Li et al.)

- **Basic idea:** Adapt algorithm to platform and input data

- **Algorithm space and parameters:**
  - Quicksort recursively, once data sets < t, use insertion or sorting network
  - CC-Radix recursively, once data sets < u, use insertion or sorting network
  - Multiway-mergesort (one step) with p subsets and fanout f then CC-radix as above

- **Input characteristics:** Use entropy E (of digits)

- **At installation time:**
  - find t and u
  - Use machine learning to learn a decision function: decision: (N, E) → \{Q, CC, MM(f, p)\}

Example Result (Sorting 12M Records)

<table>
<thead>
<tr>
<th>Cache optimization</th>
<th>MMM Atlas</th>
<th>Sparse MVM Sparsity/Bebop</th>
<th>DFT FFTW</th>
<th>Sorting Adaptive sorting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocking</td>
<td>Blocking</td>
<td>Blocking (rarely useful)</td>
<td>recursive FFT, fusion of steps</td>
<td>Recursive, array-based sorting algorithms</td>
</tr>
<tr>
<td>Register optimization</td>
<td>Blocking</td>
<td>Blocking (sparse format)</td>
<td>Scheduling small FFTs</td>
<td>Scheduling sorting networks</td>
</tr>
<tr>
<td>Optimized basic blocks</td>
<td></td>
<td>Unrolling, instruction ordering, scalar replacement, simplifications (for FFT), different algorithm (for sorting)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other optimizations</td>
<td>—</td>
<td>—</td>
<td>Precomputation of constants</td>
<td>Sorting specific</td>
</tr>
<tr>
<td>Adaptivity</td>
<td>Search: blocking parameters</td>
<td>Search: register blocking size</td>
<td>Search: recursion strategy</td>
<td>Search: recursion strategy</td>
</tr>
</tbody>
</table>
Course Summary:

What I hope you have learned
Understand the problem (symptoms)

- Minimizing operations count ≠ minimizing runtime (and not even close)
- A straightforward implementation is usually 10-100x suboptimal
- Optimal performance on one machine does mean optimal performance on another
- End of automatic speedup for legacy software is near
And the Cause

- Evolution of computing platforms:
  - End of CPU frequency scaling (power density)
  - Deep memory hierarchies
  - Vector instructions
  - Multiple cores

![Chart showing evolution of CPU performance](chart.png)
Understand what to optimize for

- First remove obvious performance killers
- Then memory hierarchy
- And only then vector instructions and multithreading
Understand how to optimize given code

- Proper timing of code
- Find runtime bottleneck
- Analyze cost (cost measure)
- Determine performance and percentage of peak (efficiency)
- Understand cache behavior of code (walking through the code)
- Apply techniques from class
- Repeat procedure

- Understand inherent limitations (degree of reuse, temporal/spatial locality, memory bound/CPU bound)
Understand how to write fast code

- Start with the right algorithm (proper structure)!!!!!!
- Continue as in previous slide
Understand how to benchmark and how to report it

- Precise description of procedure
- Correct
- Fair
- Proper analysis
- And if the plots are nice even better 😊