External-Memory and Cache-Oblivious Algorithms: Theory and Experiments

Gerth Stølting Brodal

maDAlGO
CENTER FOR MASSIVE DATA ALGORITHMIC
Motivation

1. Datasets get MASSIVE

2. Computer memories are HIERARCHICAL

New Algorithmic Challenges 😊

....both theoretical and experimental
Massive Data

More New Information Over Next 2 Years Than in All Previous History

Examples (2002)
- **Phone**: AT&T 20TB phone call database, wireless tracking
- **Consumer**: WalMart 70TB database, buying patterns
- **WEB/Network**: Google index $8 \cdot 10^9$ pages, internet routers
- **Geography**: NASA satellites generate TB each day
Computer Hardware
A Typical Computer
## Customizing a Dell 650

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor speed</td>
<td>2.4 – 3.2 GHz</td>
</tr>
<tr>
<td>L3 cache size</td>
<td>0.5 – 2 MB</td>
</tr>
<tr>
<td>Memory</td>
<td>1/4 – 4 GB</td>
</tr>
<tr>
<td>Hard Disk</td>
<td>36 GB– 146 GB</td>
</tr>
<tr>
<td>CD/DVD</td>
<td>7.200 – 15.000 RPM</td>
</tr>
<tr>
<td>L2 cache size</td>
<td>256 – 512 KB</td>
</tr>
<tr>
<td>L2 cache line size</td>
<td>128 Bytes</td>
</tr>
<tr>
<td>L1 cache line size</td>
<td>64 Bytes</td>
</tr>
<tr>
<td>L1 cache size</td>
<td>16 KB</td>
</tr>
</tbody>
</table>
Pentium® 4
Processor Microarchitecture

Intel Technology Journal, 2001
## Memory Access Times

<table>
<thead>
<tr>
<th></th>
<th>Latency</th>
<th>Relative to CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>0.5 ns</td>
<td>1</td>
</tr>
<tr>
<td>L1 cache</td>
<td>0.5 ns</td>
<td>1-2</td>
</tr>
<tr>
<td>L2 cache</td>
<td>3 ns</td>
<td>2-7</td>
</tr>
<tr>
<td>DRAM</td>
<td>150 ns</td>
<td>80-200</td>
</tr>
<tr>
<td>TLB</td>
<td>500+ ns</td>
<td>200-2000</td>
</tr>
<tr>
<td>Disk</td>
<td>10 ms</td>
<td>$10^7$</td>
</tr>
</tbody>
</table>

Increasing
Hierarchical Memory Basics

Increasing access time and space
A Trivial Program

for (i=0; i+d<n; i+=d) A[i]=i+d;
A[i]=0;

for (i=0, j=0; j<8*1024*1024; j++) i = A[i];
A Trivial Program — d=1

RAM : \( n \approx 2^{25} = 128 \text{ MB} \)
A Trivial Program — d=1

L1 : $n \approx 2^{12} = 16 \text{ KB}$
L2 : $n \approx 2^{16} = 256 \text{ KB}$
A Trivial Program — $n=2^{24}$
Experiments were performed on a DELL 8000, PentiumIII, 850MHz, 128MB RAM, running Linux 2.4.2, and using gcc version 2.96 with optimization -O3

L1 instruction and data caches
• 4-way set associative, 32-byte line size
• 16KB instruction cache and 16KB write-back data cache

L2 level cache
• 8-way set associative, 32-byte line size
• 256KB

www.Intel.com
Algorithmic Problem

- Modern hardware is not uniform — *many* different parameters
  - Number of memory levels
  - Cache sizes
  - Cache line/disk block sizes
  - Cache associativity
  - Cache replacement strategy
  - CPU/BUS/memory speed
- Programs should ideally run for many different parameters
  - by knowing many of the parameters at runtime, or
  - by knowing few essential parameters, or
  - ignoring the memory hierarchies
- Programs are executed on unpredictable configurations
  - Generic portable and scalable software libraries
  - Code downloaded from the Internet, e.g. Java applets
  - Dynamic environments, e.g. multiple processes
Memory Models
### Hierarchical Memory Model

- many parameters

Limited success because to complicated

Diagram showing CPU connected to L1, L2, L3, RAM, and Disk, with increasing access time and space.
External Memory Model—two parameters
Aggarwal and Vitter 1988

- Measure number of block transfers between two memory levels
- Bottleneck in many computations
- Very successful (simplicity)

Limitations
- Parameters $B$ and $M$ must be known
- Does not handle multiple memory levels
- Does not handle dynamic $M$
Ideal Cache Model—no parameters !?

Frigo, Leiserson, Prokop, Ramachandran 1999

- Program with only one memory
- Analyze in the I/O model for
- Optimal off-line cache replacement
- strategy arbitrary $B$ and $M$

Advantages
- Optimal on arbitrary level $\rightarrow$ optimal on all levels
- Portability, $B$ and $M$ not hard-wired into algorithm
- Dynamic changing parameters
Justification of the Ideal-Cache Model
Frigo, Leiserson, Prokop, Ramachandran 1999

**Optimal replacement** $\text{LRU} + 2 \times \text{cache size} \rightarrow \text{at most } 2 \times \text{cache misses}$

Sleator and Tarjan, 1985

**Corollary**

$T_{M,B}(N) = O(T_{2M,B}(N))$  #cache misses using LRU is $O(T_{M,B}(N))$

**Two memory levels**

Optimal cache-oblivious algorithm satisfying $T_{M,B}(N) = O(T_{2M,B}(N))$

$\rightarrow$ optimal #cache misses on each level of a multilevel LRU cache

**Fully associativity cache**

Simulation of LRU

• Direct mapped cache

• Explicit memory management

• Dictionary (2-universal hash functions) of cache lines in memory

• Expected $O(1)$ access time to a cache line in memory
Basic External-Memory and Cache-Oblivious Results
Scanning

\[
\begin{align*}
\text{sum} &= 0 \\
\text{for } i = 1 \text{ to } N \text{ do } \text{sum} = \text{sum} + A[i]
\end{align*}
\]

Corollary

External/Cache-oblivious selection requires $O(N/B)$ I/Os

Hoare 1961 / Blum et al. 1973
Merging \( k \) sequences with \( N \) elements requires \( O(N/B) \) IOs provided \( k \leq M/B - 1 \)
External-Memory Sorting

• $\theta(M/B)$-way MergeSort achieves optimal

$$O(\text{Sort}(N))=O\left(\frac{N}{B} \cdot \log_{M/B}(\frac{N}{B})\right)$$

I/Os

Aggarwal and Vitter 1988
Cache-Oblivious Merging

Frigo, Leiserson, Prokop, Ramachandran 1999

• $k$-way merging (lazy) using binary merging with buffers
• tall cache assumption $M \geq B^2$
• $O(N/B \cdot \log_{M/B} k)$ IOs
Cache-Oblivious Sorting – FunnelSort
Frigo, Leiserson, Prokop and Ramachandran 1999

- Divide input in $N^{1/3}$ segments of size $N^{2/3}$
- Recursively FunnelSort each segment
- Merge sorted segments by an $N^{1/3}$-merger

**Theorem** Provided $M \geq B^2$ performs optimal $O(\text{Sort}(N))$ I/Os
Sorting (Disk)

Brodal, Fagerberg, Vinther 2004

Walltime/n*log n

log n
Sorting (RAM)

Brodal, Fagerberg, Vinther 2004

![Graph showing walltime/n * log n vs log n for various sorting algorithms.]

- Funnelsort2
- Funnelsort4
- Mix
- msort-c
- msort-m
- Rmerge
- GCC
- TPIE
External Memory Search Trees

- B-trees  Bayer and McCreight 1972
- Searches and updates use $O(\log_B N)$ I/Os
Cache-Oblivious Search Trees

- Recursive memory layout (van Emde Boas)

  Binary tree

  Searches use $O(\log_B N)$ I/Os

- Dynamization (several papers) – ”reduction to static layout”
Cache-Oblivious Search Trees

Brodal, Jacob, Fagerberg 1999
Matrix Multiplication
Frigo, Leiserson, Prokop and Ramachandran 1999

Average time taken to multiply two $N \times N$ matrices divided by $N^3$

Iterative: $O(N^3)$ I/O
Recursive: $O\left(\frac{N^3}{B \sqrt{M}}\right)$ I/Os
The Influence of other Chip Technologies...

...why some experiments do not turn out as expected
Translation Lookaside Buffer (TLB)

- translate **virtual addresses** into physical addresses
- small table (full associative)
- TLB miss requires lookup to the page table

![Diagram of TLB](wikipedia.org)

- **Size:** 8 - 4,096 entries
- **Hit time:** 0.5 - 1 clock cycle
- **Miss penalty:** 10 - 30 clock cycles
TLB and Radix Sort

Rahman and Raman 1999

Time for one permutation phase

- 4-bit radix
- 5-bit radix
- 6-bit radix
- 7-bit radix
- 8-bit radix
- 11-bit radix
- 16-bit radix

Diagram showing the time per key (microseconds) for different radices (4-bit to 16-bit) as a function of the number of keys (from 0 to 3.5e+07).
### TLB and Radix Sort

Rahman and Raman 1999

<table>
<thead>
<tr>
<th>$n$</th>
<th>EPLSB11</th>
<th>PLSB11</th>
<th>EBT11</th>
<th>LSB556</th>
<th>LSB6</th>
<th>FLSB11</th>
<th>QSort</th>
<th>MSort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M</td>
<td>0.46</td>
<td>0.47</td>
<td>0.54</td>
<td>0.57</td>
<td>0.64</td>
<td>0.90</td>
<td>0.70</td>
<td>1.02</td>
</tr>
<tr>
<td>2M</td>
<td>0.89</td>
<td>0.92</td>
<td>1.09</td>
<td>1.12</td>
<td>1.28</td>
<td>1.86</td>
<td>1.50</td>
<td>2.22</td>
</tr>
<tr>
<td>4M</td>
<td>1.74</td>
<td>1.82</td>
<td>2.20</td>
<td>2.20</td>
<td>2.56</td>
<td>3.86</td>
<td>3.24</td>
<td>4.47</td>
</tr>
<tr>
<td>8M</td>
<td>3.53</td>
<td>3.64</td>
<td>4.49</td>
<td>4.35</td>
<td>5.09</td>
<td>7.68</td>
<td>6.89</td>
<td>9.71</td>
</tr>
<tr>
<td>16M</td>
<td>7.48</td>
<td>7.85</td>
<td>8.81</td>
<td>8.57</td>
<td>10.22</td>
<td>15.23</td>
<td>14.65</td>
<td>19.47</td>
</tr>
<tr>
<td>32M</td>
<td>14.96</td>
<td>15.66</td>
<td>17.55</td>
<td>17.52</td>
<td>20.45</td>
<td>31.71</td>
<td>31.69</td>
<td>41.89</td>
</tr>
</tbody>
</table>

TLB optimized
Cache Associativity

Execution times for scanning $k$ sequences of total length $N=2^{24}$ in round-robin fashion (SUN-Sparc Ultra, direct mapped cache)

<table>
<thead>
<tr>
<th>$k$</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>0.52</td>
<td>4.03</td>
<td>3.99</td>
<td>4.02</td>
<td>4.04</td>
<td>4.01</td>
<td>5.6</td>
<td>5.58</td>
<td>5.6</td>
<td>5.53</td>
<td>5.55</td>
</tr>
</tbody>
</table>

Cache associativity

TLB misses

Sanders 1999
Prefetching vs. Caching

Pan, Cherng, Dick, Ladner 2007

All Pairs Shortest Paths (APSP)
Organize data so that the CPU can prefetch the data → computation (can) dominate cache effects

Prefetching disabled

Prefetching enabled
Branch Prediction vs. Caching

**QuickSort**  Select the pivot biased to achieve subproblems of size $\alpha$ and $1-\alpha$

+ reduces # branch mispredictions
- increases # instructions and # cache faults

Kalogi and Sanders 2006
Branch Prediction vs. Caching

**Skewed Search Trees**  Subtrees have size $\alpha$ and $1-\alpha$

- reduces # branch mispredictions
- increases # instructions and # cache faults

Brodal and Moruz 2006

---

Graph showing the relationship between cache misses and running time as a function of $\alpha$. The graph illustrates the trade-off between branch prediction and caching, with different algorithms (e.g., DFS, VEB) showing varied performance under different values of $\alpha$. The graphs suggest that choosing an optimal $\alpha$ can improve overall performance by balancing the reduction in branch mispredictions with the increase in instructions and cache faults.
Another Trivial Program
— the influence of branch predictions

for (i=0; i<n; i++)
    A[i] = rand() % 1000;

for (i=0; i<n; i++)
    if (A[i] > threshold)
        g++;
    else
        s++;
Branch Mispredictions

- Worst-case number of mispredictions for threshold=500

```c
for(i=0;i<n;i++)
a[i]=rand()%1000;
for(i=0;i<n;i++)
  if(a[i]>threshold)
    g++;
  else
    s++;
```
Running Time

\[
\text{for}(i=0; i<n; i++) \\
\text{\quad } a[i] = \text{rand}() \% 1000; \\
\text{for}(i=0; i<n; i++) \\
\text{\quad if}(a[i] > \text{threshold}) \\
\text{\quad \quad } g++; \\
\text{\quad else} \\
\text{\quad \quad } s++; \\
\]

- Prefetching disabled  
  \(\rightarrow 0.3 - 0.7\) sec
- Prefetching enabled  
  \(\rightarrow 0.15 - 0.5\) sec
L2 Cache Misses

```c
for(i=0;i<n;i++)
a[i]=rand()% 1000;
for(i=0;i<n;i++)
  if(a[i]>threshold)
    g++;
  else
    s++;
```

- Prefetching disabled
  → 2,500,000 cache misses
- Prefetching enabled
  → 40,000 cache misses
L1 Cache Misses

```
for(i=0; i<n; i++)
    a[i] = rand() % 1000;

for(i=0; i<n; i++)
    if(a[i] > threshold)
        g++;
    else
        s++;
```

- Prefetching disabled
  → $4 - 16 \times 10^6$ cache misses
- Prefetching enabled
  → $2.5 - 5.5 \times 10^6$ cache misses
Summary

• Be conscious about the presence of memory hierarchies when designing algorithms

• Experimental results not often quite as expected due to neglected hardware features in algorithm design

• External memory model and cache-oblivious model adequate models for capturing disk/cache bottlenecks
What did I not talk about...

- non-uniform memory
- parallel disks
- parallel/distributed algorithms
- graph algorithms
- computational geometry
- string algorithms
- and a lot more...

THE END