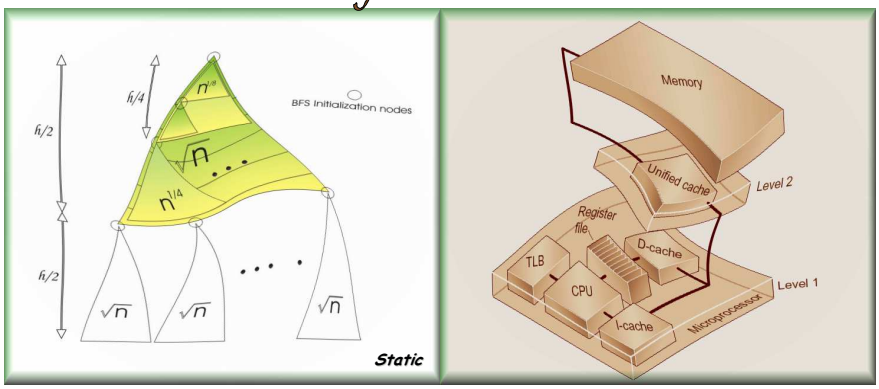
Cache Oblivious Algorithms
Theory & Practice



Research Proficiency Examination

Piyush Kumar
Department of Computer Science
Advisor: Joseph S.B. Mitchell

CO Algorithms: Brief History

- Frigo, Leiserson, Prokop, Ramachandran (FOCS 99)
  Cache Oblivious Algorithms
- Harold Prokop's **Thesis**
- Bender, Demaine, Farch-Coltun (FOCS 00)
  Cache Oblivious B-Trees
- …
- Arge, Bender, Demaine et.al. (STOC02)
  CO Priority Queue

Talk Outline…

- **Motivation**
  - ④ Matrix Multiplication/Transposition
  - ④ Static Searches in Bal. Bin. Trees
- **The Model**
- **CO-Sorting**
- **Some Analysis**
- CO-Sorting **Experiments**
- Do's and Don'ts of the model
- Future work

Workstations

SUN UltraSparc 2:
UltraSparc 16kB L1, 512kB L2.

SGI Visual Workstation 540:
Quad-Pentium III 32kB L1, 1024kB L2.

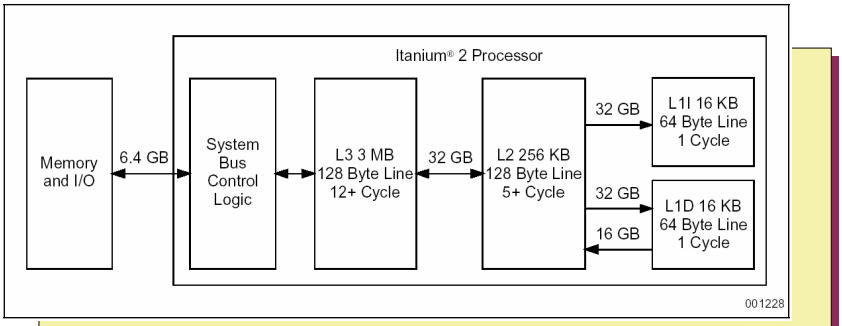
Dell Precision:
Dual-Pentium III 32kB L1 512kB L2.

IBM ThinkPad 600:
Pentium II 32kB L1, 256kB L2.

Compaq Presario:
AMD K6-III 64kB L1, 256kB L2, 1024kB L3.

How can we write portable code that runs efficiently on different multilevel caching architectures?

Intel Itaniums



Matrix Multiplication (MM)

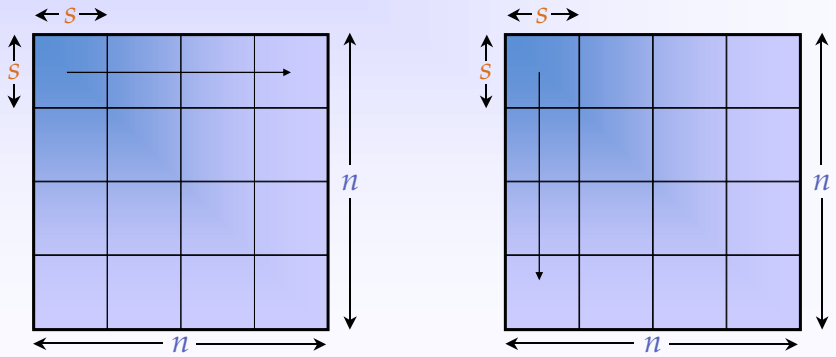
$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

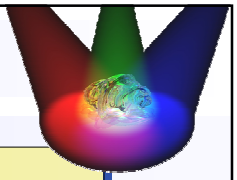
C A B

Cache-Aware MM

```
BLOCK-MULT (A,B,C,n)
1  for i ← 1 to n/s
2      do for j ← 1 to n/s
3          do for k ← 1 to n/s
4              do ORD-MULT (A_ik,B_kj,C_ij,s)
```

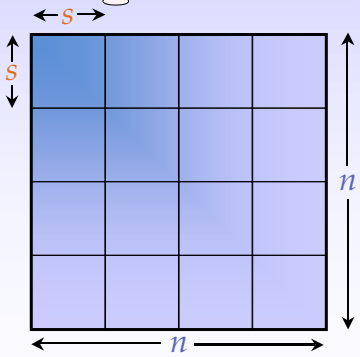


Cache-Aware MM



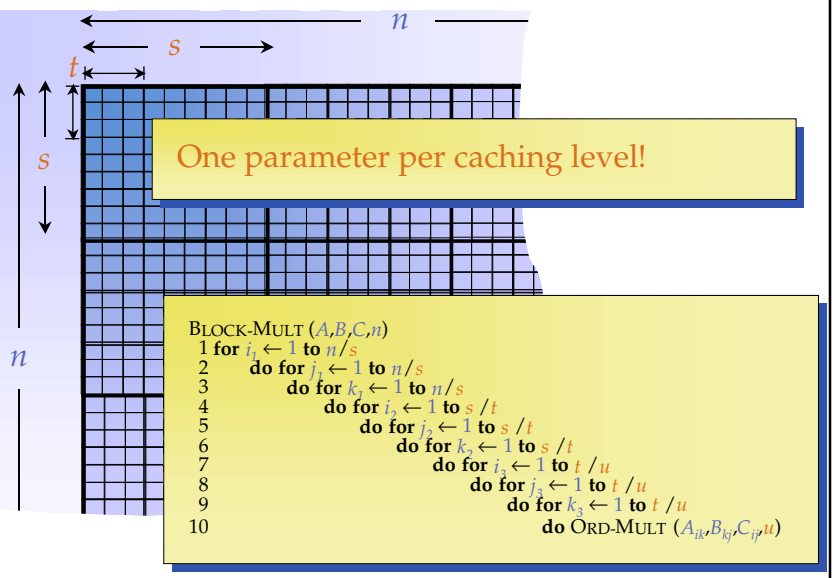
```
BLOCK-MULT (A,B,C,n)
1  fo
2
3
4                    ULT (A_ik,B_kj,C_ij,s)
```

Oracle?!



- Tune s so that A_{ik}, B_{kj}, and C_{ij} just fit into cache? $s = \Theta(\sqrt{Z})$
- If $n > s$, then
$$Q(n) = \Theta((n/s)^3(s^2/L)) = \Theta(n^3/L\sqrt{Z}).$$
- Optimal [HK81].

Three-Level Cache




```
BLOCK-MULT (A,B,C,n)
 1 for i1 ← 1 to n/s
 2     do for j1 ← 1 to n/s
 3         do for k1 ← 1 to n/s
 4             do for i2 ← 1 to s/t
 5                 do for j2 ← 1 to s/t
 6                     do for k2 ← 1 to s/t
 7                         do for i3 ← 1 to t/u
 8                             do for j3 ← 1 to t/u
 9                                 do for k3 ← 1 to t/u
10                                     do ORD-MULT (Aik,Bkj,Cij,u)
```

Recursive Matrix Multiplication

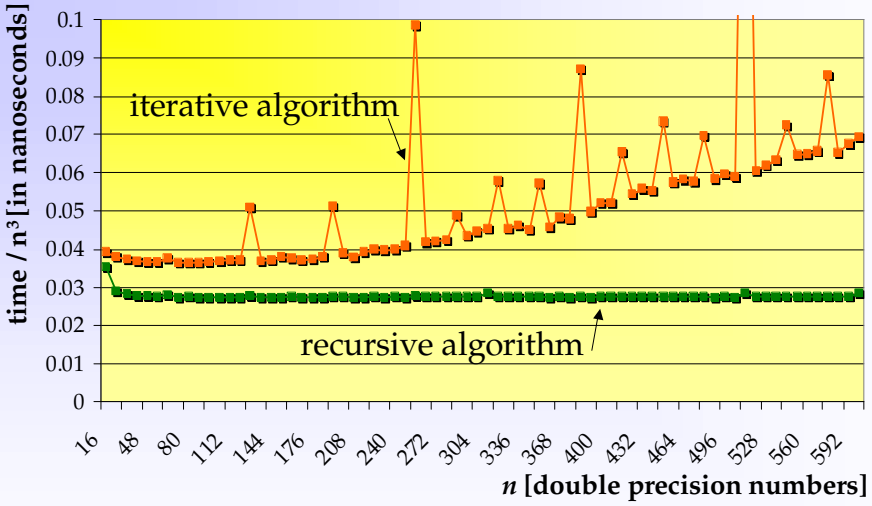
Divide and conquer on $n \times n$ matrices.

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \times \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$= \begin{pmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{21}B_{11} & A_{21}B_{12} \end{pmatrix} + \begin{pmatrix} A_{12}B_{21} & A_{12}B_{22} \\ A_{22}B_{21} & A_{22}B_{22} \end{pmatrix}$$

8 multiplications of $(n/2) \times (n/2)$ matrices.
1 addition of $n \times n$ matrices.

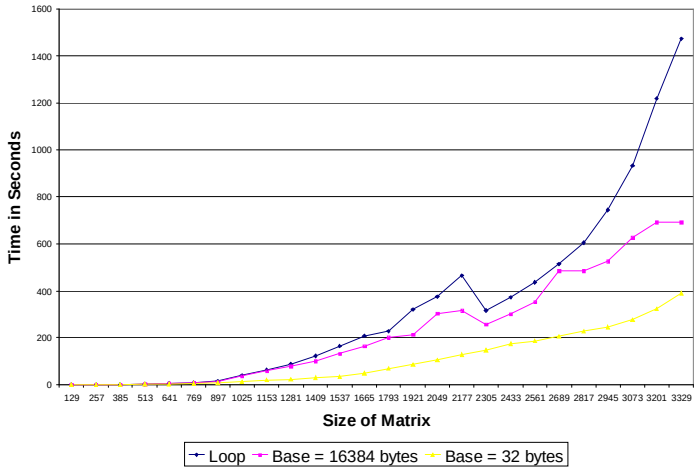
$\Theta(n^3)$-*Matrix Multiplication (n,n) × (n,n)*



450-MHz AMD K6-III processor with 32kB L1-cache, 64kB L2-cache, and 1MB L3-cache.

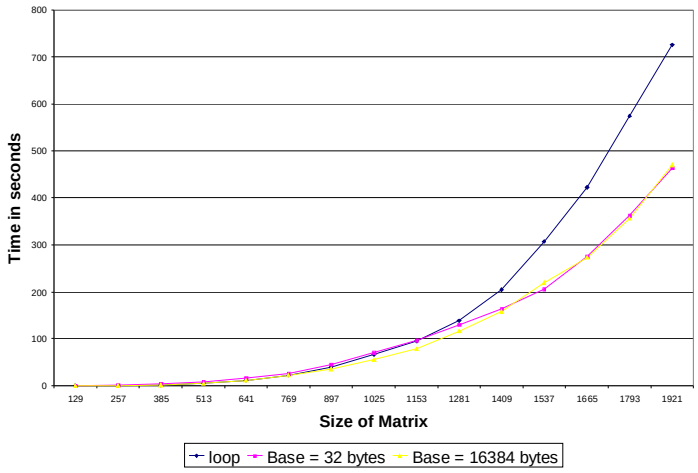
Experiments: MM

- Linux Athlon 1Ghz/1Gb/g++ -O3



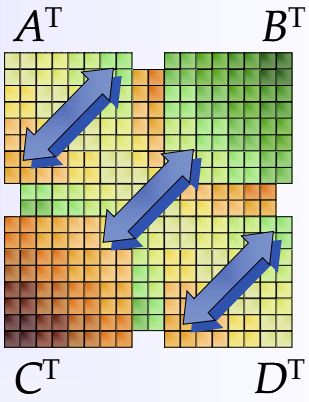
Experiments: MM

- Linux/Itanium/2GB/g++ -O3



Recursive Transpose

1. Partition matrix in 4 submatrices A, B, C, and D
2. Recursively transpose A.
3. Recursively transpose and swap B and C.
4. Recursively transpose D.



$\Theta(n^2 / L)$ cache misses, which is optimal. Used as a subroutine in our optimal cache-oblivious FFT [HK81].

Code: The InPlace Loop

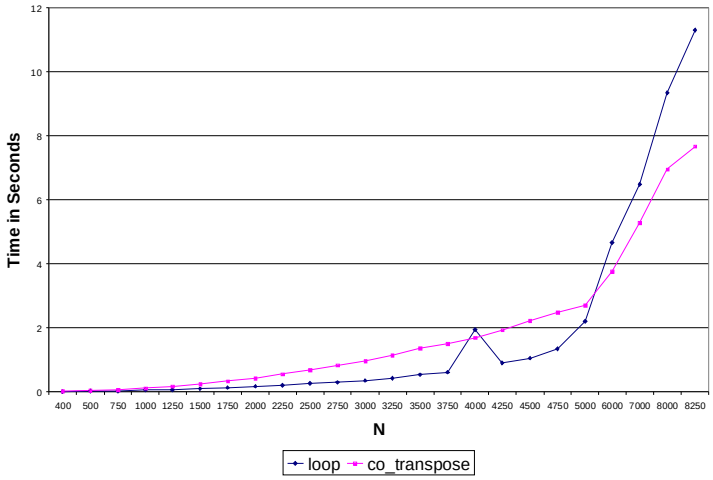
```
for (i = 0; i < N; i++)
    for (j = i+1; j < N; j++)
        swap(A[i][j], A[j][i])
```

Code : Co Transpose

```
void transpose(int x, int delx, int y, int dely,
               ElementType I[N][P], ElementType O[P][N]){
        // Base Case of recursion
        // Should be tailored for specific machines if one wants
        // this code to perform better.
        if((delx == 1) && (dely == 1)) {
                O[y][x] = I[x][y];
                return;
        }
        // Divide the transposition into two sub transposition
        // problems, depending upon which side of the matrix is
        // bigger.
        if(delx >= dely){
                int xmid = delx / 2;
                transpose(x,xmid,y,dely,I,O);
                transpose(x+xmid,delx-xmid,y,dely,I,O);
                return;
        }
        // Similarly cut from ymid into two subproblems
        ...
}
```

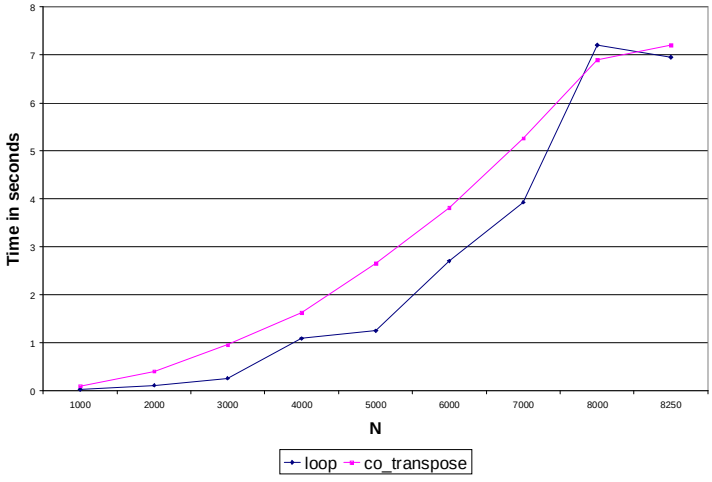
Experiments: MT

- Notebook, Windows 2k/512Mb/PIII 1GHz/g++ -O3



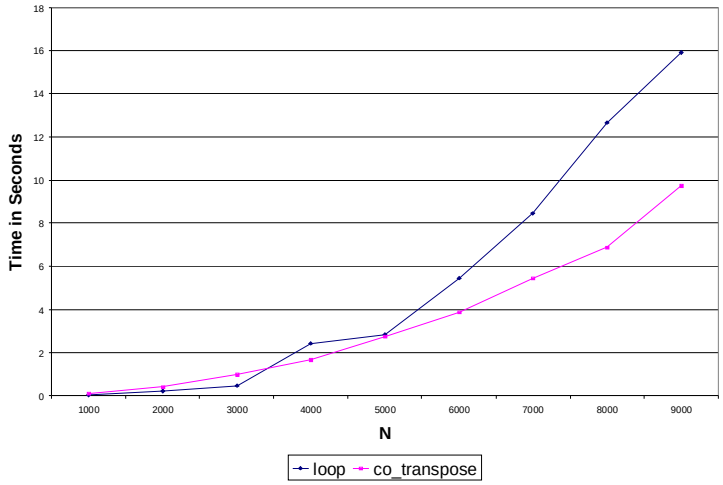
Experiments: MT

- Notebook, Windows 2k/512Mb/PIII 1GHz/g++ -O3



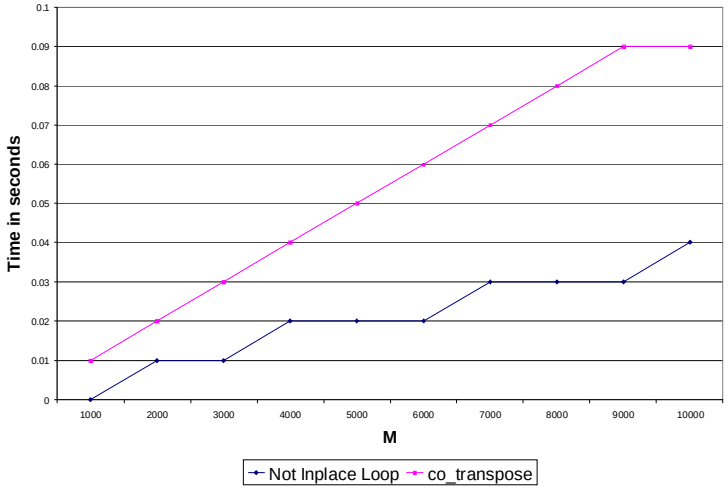
Experiments: MT

- Linux Athlon 1Ghz/1Gb/g++ -O3



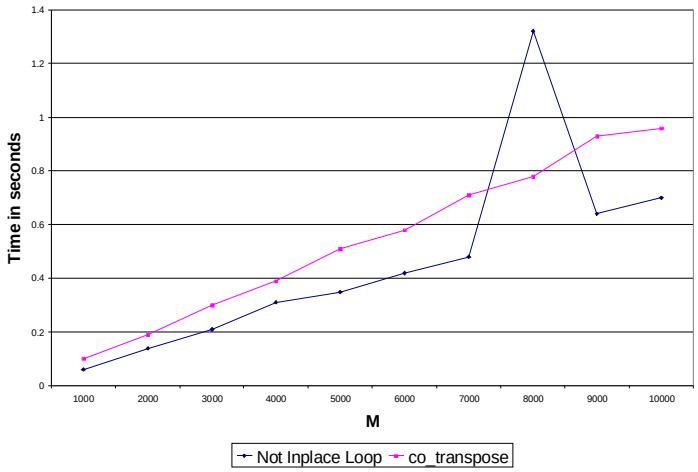
Experiments: MT

- Linux Athlon 1Ghz/1Gb/g++ -O3/ Size =N x (P =100) , tall matrices

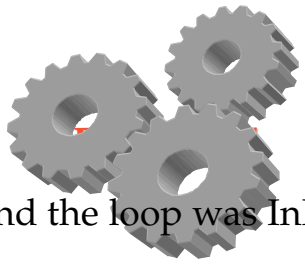


Experiments: MT

- Linux Athlon 1Ghz/1Gb/g++ -O3/ Size = N x (P =1000)



What went Wrong?

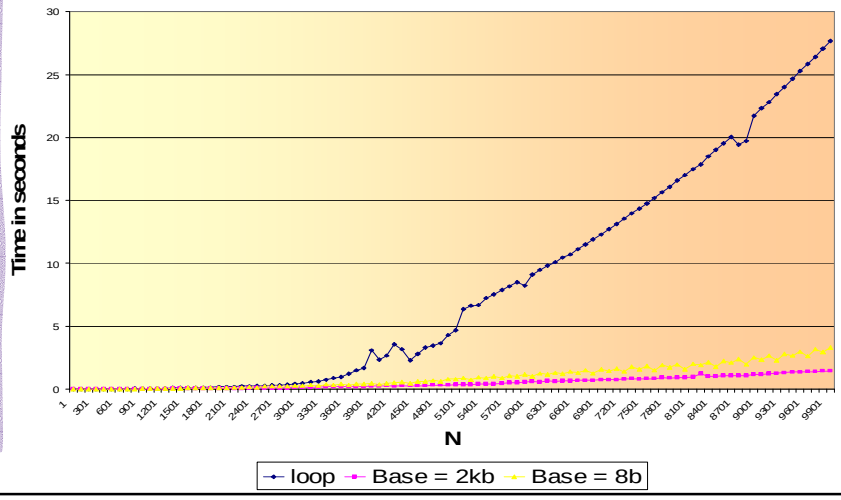


And the loop was InPlace!

Experiments: MT

Loop not Inplace

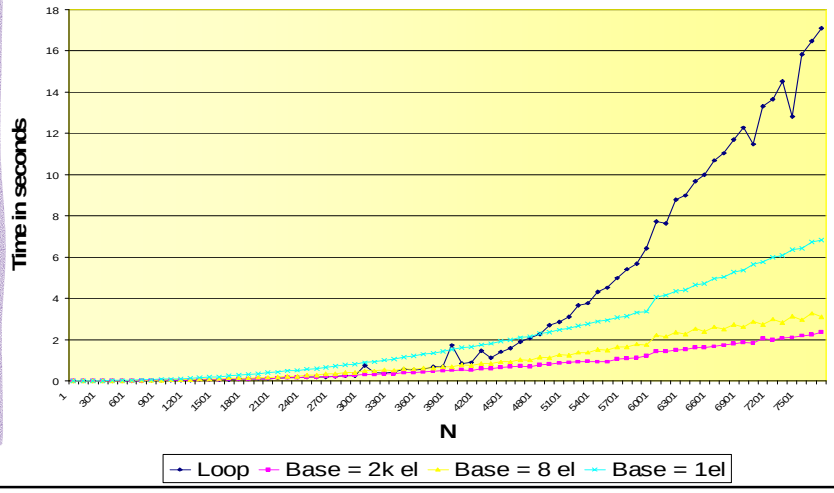
- Linux Athlon 1Ghz/1Gb/g++ -O3/ Size = N x N



Experiments: MT

Loop not Inplace

- Notebook, Windows 2k/512Mb/PIII 1GHz/g++ -O3



Did we miss something?

- Alg 1: Naïve Algorithm
- Alg 2: Simple blocking using fixed B
- Alg 3: Half Copy
- Alg 4: Full Copy
- Alg 5: CO
- Alg 6: Morton Ordering

Chatterjee & Sen HPCA 00

Did we miss something?

Running time (seconds), block size = 2^5						
$\log_2 N$	Alg.1	Alg. 2	Alg. 3	Alg. 4	Alg. 5	Alg. 6
10	0.21	0.10	0.06	0.06	0.08	0.03
11	0.86	0.49	0.39	0.35	0.45	0.14
12	3.37	1.63	1.05	1.14	2.16	0.54
13	13.56	6.38	4.55	4.99	6.69	2.13

Running time (seconds), block size = 2^6						
$\log_2 N$	Alg.1	Alg. 2	Alg. 3	Alg. 4	Alg. 5	Alg. 6
10	0.13	0.08	0.06	0.05	0.08	0.03
11	0.85	0.42	0.34	0.28	0.45	0.13
12	3.38	1.58	0.89	0.97	1.97	0.52
13	13.51	5.99	3.58	3.91	7.00	2.09

Running time (seconds), block size = 2^7						
$\log_2 N$	Alg.1	Alg. 2	Alg. 3	Alg. 4	Alg. 5	Alg. 6
10	0.14	0.12	0.05	0.05	0.09	0.03
11	0.87	0.42	0.36	0.24	0.47	0.20
12	3.36	1.46	0.85	0.88	2.03	0.59
13	13.46	5.74	3.12	3.35	6.86	2.35

Static Searches

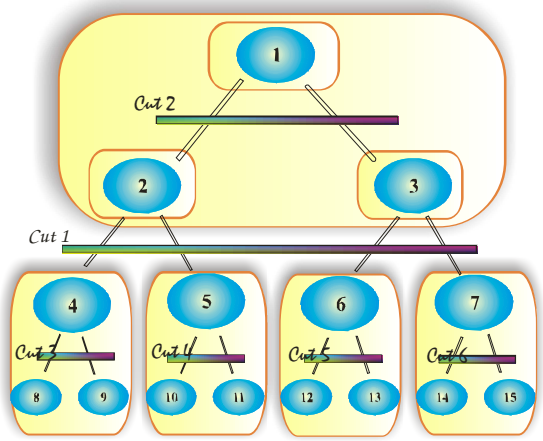
- Only for balanced binary trees
- Assume there are no insertions and deletions
- Only searches

Can we speed it up?

What is a layout?

- Mapping of nodes of a tree to the Memory
- Different kinds of layouts
  - In-order
  - Post-order
  - Pre-order
  - Van Emde Boas
- *Main Idea* : Store Recursive subtrees in contiguous memory

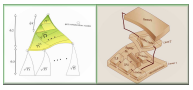
Example of Van Emde Boas

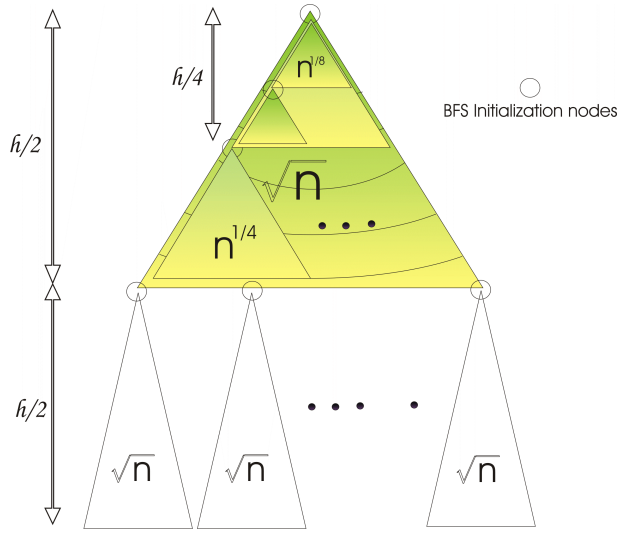


ACTUAL LAYOUT OF TREE IN MEMORY:

1, 2, 3, 4, 8, 9, 5, 10, 11, 6, 12, 13, 7, 14, 15

Another View





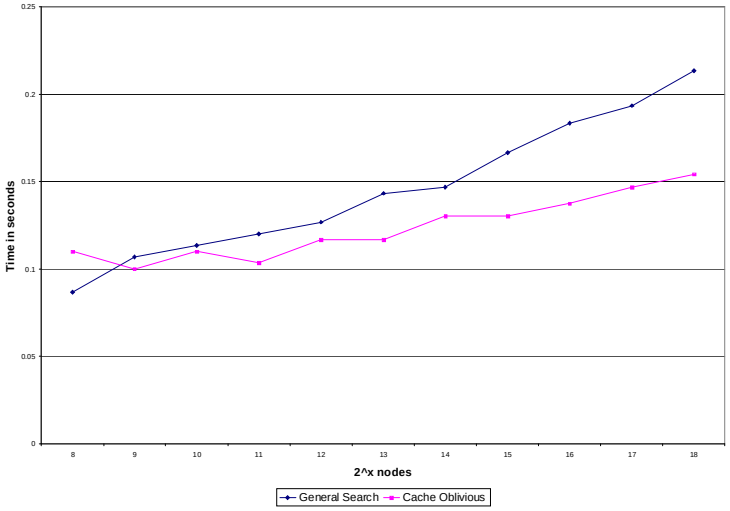
Theoretical Guarantees?

- Cache Complexity Q(n) = $O(\log_L n)$
- Work Complexity W(n) = $O(\log\ n)$

From Prokop's Thesis

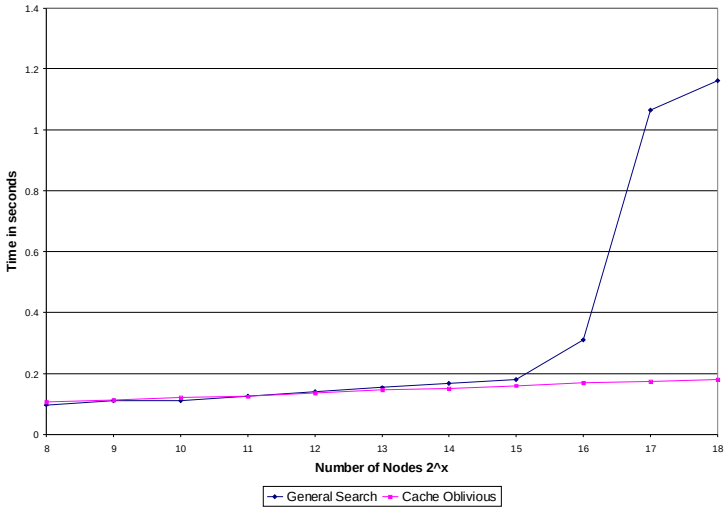
In Practice??

- Windows notebook/512MB/PIII 1Gz/256 byte nodes



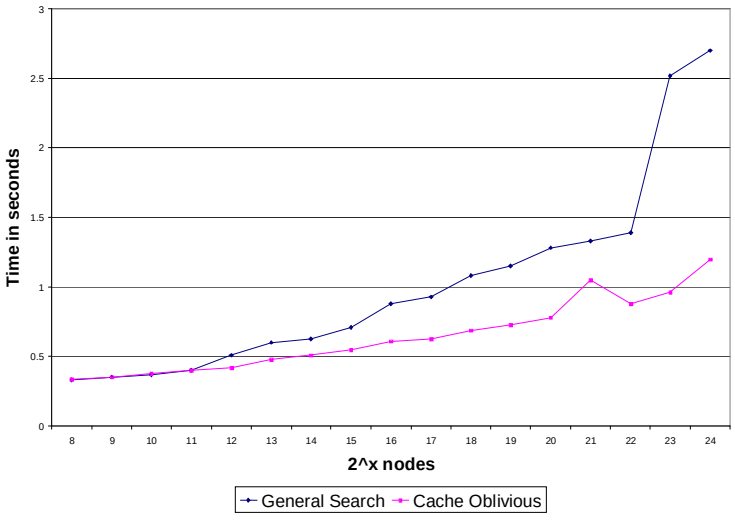
In Practice II

- Windows notebook/512MB/PIII 1Gz/32 byte nodes



In Practice III

- Linux/Itanium/2GB/g++ -O3/ 48 byte nodes



In Practice!

- Matrix Operations by Morton Ordering By David S. Wise

  (Cache oblivious Practical Matrix operation results)
- Bender, Duan, Wu

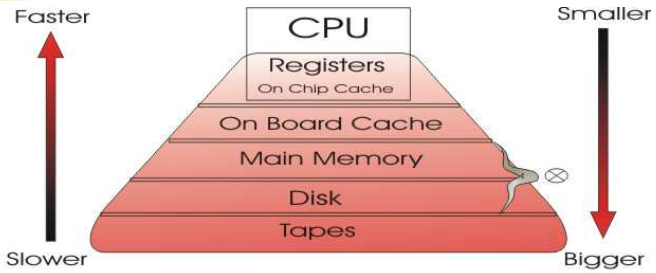
  (Cache oblivious dictionaries)
- Rahman, Cole, Raman

  (CO B-Trees)

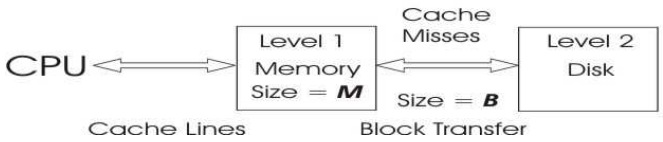
Talk outline…

- Motivation (Searching BBT)
- *The Model*
- CO-Sorting
- …

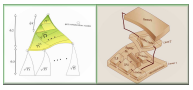
(M,B) Ideal Cache Model



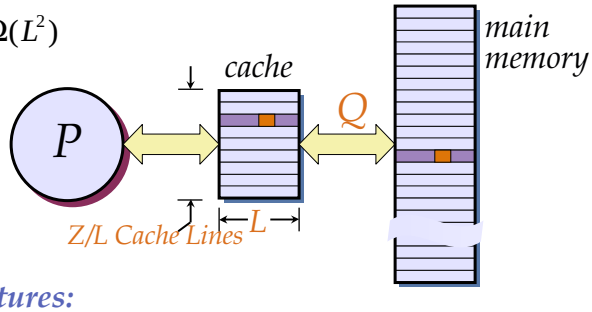
⊗ Two consecutive levels of the hierarchy



(Z,L) Ideal Cache Model



$Z = \Omega(L^2)$



Features:

- Two-level hierarchy.
- Cache of size Z.
- Cache-line length L.
- Fully associative.
- Optimal, omniscient replacement.

Measures:

- *Work W.*
- *Cache misses Q.*

Assumptions?

- Two Levels of Memory
- Tall Cache Assumption
- Optimal Cache Replacement "No Asymptotic loss"
- Fully-associative LRU can be used instead of optimal replacement with no asymptotic loss of performance [ST85].
- Fully-associative LRU caches can be maintained in ordinary memory with constant slowdown in expected performance.

Cache Obliviousness

- Cache-oblivious algorithms naturally tune for
  - varying cache sizes.
  - multiple levels of cache.

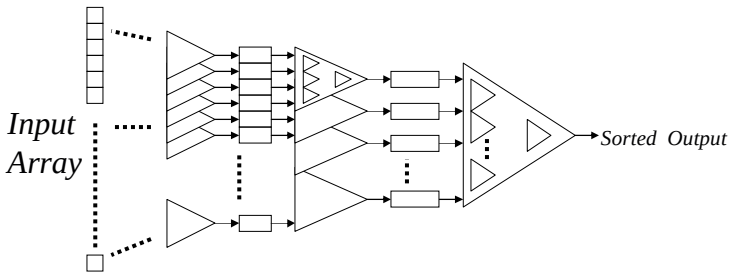
    When a subproblem fits into a given level of cache, no further cache misses are incurred beyond those required to bring the subproblem itself into the cache.
- An optimal cache-oblivious algorithm can be made to run optimally in the HMM [AACN87] and SUMH [VN93] models

CO-Sorting!

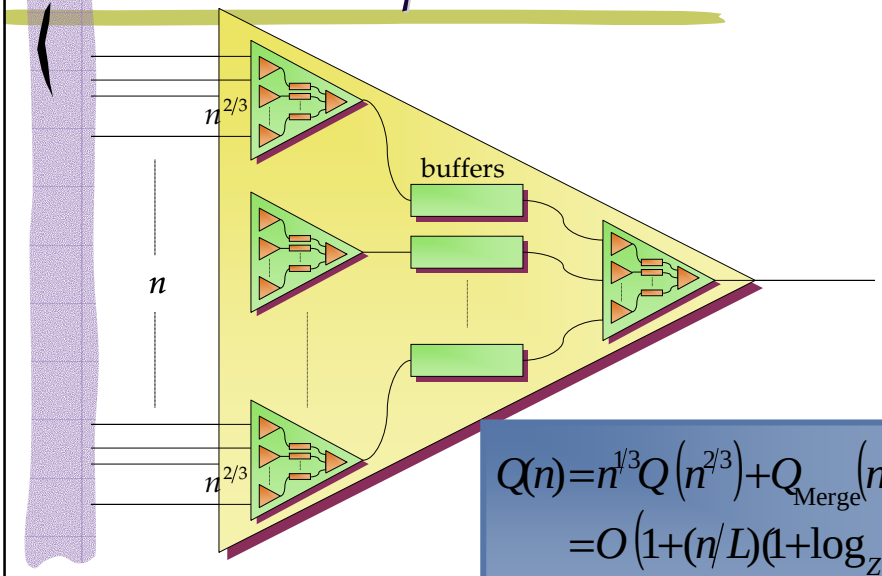
- Only two methods known
  - **Funnel Sort** (Modified Merge Sort)
  - **Distribution Sort** ⑥
    (Modified Sample Sort, We implement a randomized version)
  - **Column Sort**

Funnel Sort

- Partition Input into $N^{1/3}$ pieces of size $N^{2/3}$ each.
- Sort each piece Recursively
- Merge sorted pieces using a $N^{1/3}$ -merger



Harold's slide picture

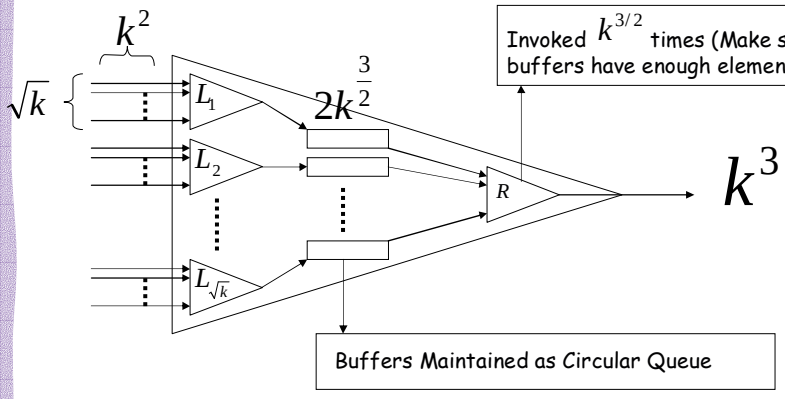


$$Q(n) = n^{1/3} Q\left(n^{2/3}\right) + Q_{\text{Merge}}\left(n^{1/3}\right)$$
$$= O\left(1 + (n/L)(1 + \log_Z n)\right)$$

Funnel Sort: k-mergers

- Takes input k sorted sequences
- Outputs k^3 elements!
- It's a clever scheduling of mergers!
- Keeps work complexity O(nlogn)

Funnel Sort: k-Merger



One invocation of R outputs $k^{3/2}$ elements

Funnel Sort : Optimality

- Work Complexity
  - $O(n \log n)$
- Cache Complexity
  - $O(1+\frac{n}{L}(1+\log_Z n))$

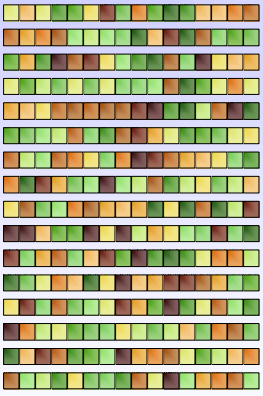
Agarwal and Vitter show that there is an $\Omega(\frac{n}{L}\log_{\frac{Z}{L}}(\frac{n}{L}))$ Bound on the number of cache misses.

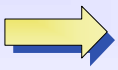
Distribution Sort

- Partition A into $\sqrt{n}$ sub-arrays each of size $\sqrt{n}$; Sort Recursively
- Distribute into buckets
- Sort Buckets Recursively
- Copy Buckets to output

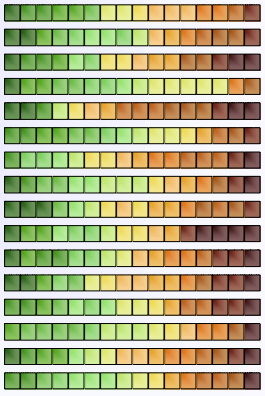
Recursive Sorting of Subarrays

n input elements, partitioned into $\sqrt{n}$ contiguous subarrays of size $\sqrt{n}$:



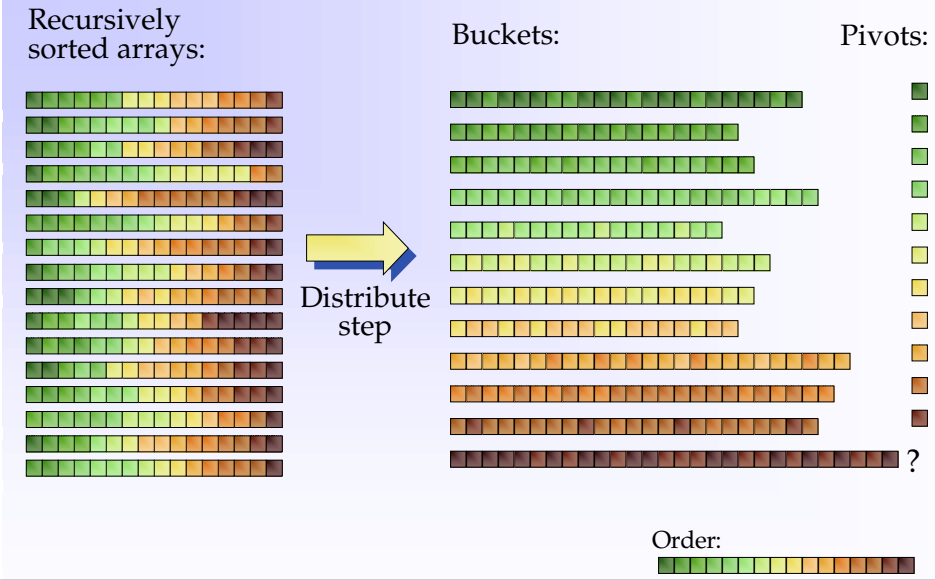


Recursively sorted arrays:



Order:

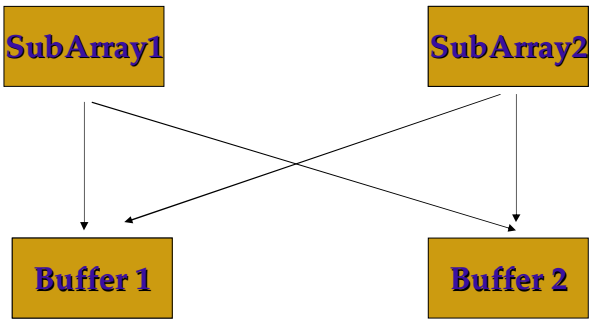
Distribution Step



The Distribution Step

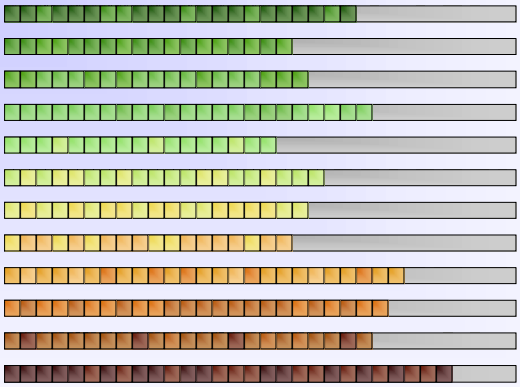
- Has to distribute subarrays into buckets $B_1, B_2, B_3 \ldots B_q$
- Not In-Place
- Similar to recursive Sample-Sort without doing Binary Search on pivots

The Recursive Bucketing used



Recursive Sorting of Buckets

After distribution step:



Recursively sort each bucket.

Some Analysis now

- Matrix Transposition
- Matrix Multiplication(Strassen)
- Column Sort ⑥

Talk outline…

- Motivation (Searching BBT)
- The Model
- CO-Sorting
  - ④ Funnel Sort
  - ④ Distribution Sort
- Some Analysis
- CO-Sorting Experiments

Randomized CO Sorting

Algorithm 9.7.1 procedure `Sort`(A)

Require: An input array A of Size N

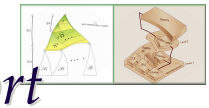
1: Partition A into $\sqrt{N}$ sub arrays of size $\sqrt{N}$. Recursively sort each subarray.
2: $R \leftarrow$ `ComputeSplitters`$(A, \sqrt{N})$
3: Sort(R) recursively. $R = \{r_0, r_1, ..., r_{\sqrt{N}}\}$.
4: Calculate counts c_i such that $c_i = |\{x \mid x \in A \text{ and } r_i \leq x < r_{i+1}\}|$
5: $c_{1+\sqrt{N}} = |A| - \Sigma_i c_i$.
6: Distribute A into buckets $B_0, B_1, ...B_{1+\sqrt{N}}$ where last element of each bucket is r_i except the last bucket. For the last bucket, the last element is maximum element of A. Note that $|B_i| = c_i$.
7: Recursively sort each B_i
8: Copy sorted buckets back to A.

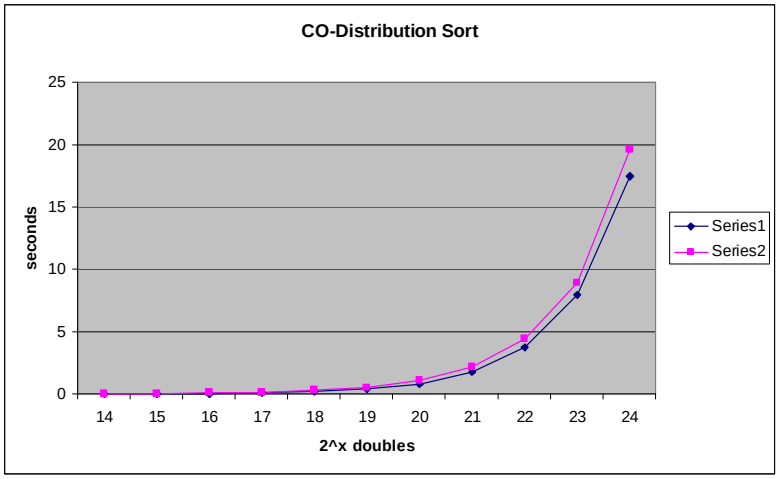
CO-Sorting Experiments

- Results for a 2-Pass Distribution Sort
- Does not use Bucket Splitting
- Uses a CO-Counting Phase
- Uses random sampling for splitters

  (That makes the implementation slightly suboptimal)

In Practice: 2 Pass CO-Dissort





Base = 2^14 = 16384

Does this imply anything?*

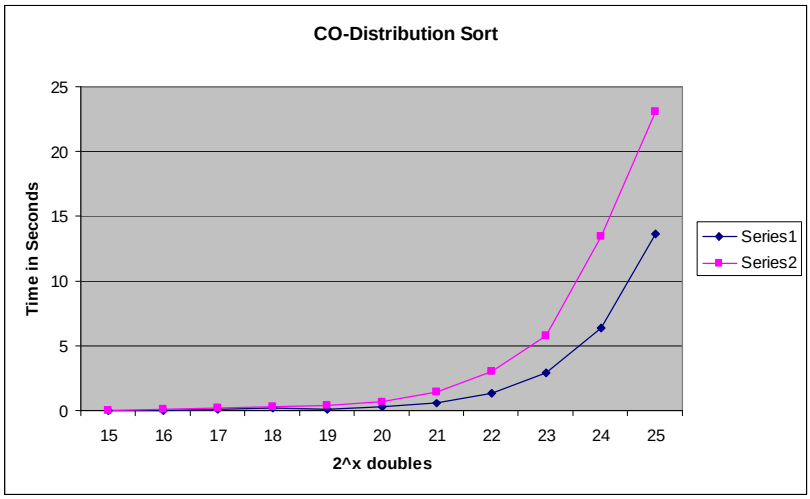
If you have 1Gb of Memory ⌘

You can sort, I/O efficiently,

1000 Gigabytes

just using one level of Recursion.

Two Levels of Recursion

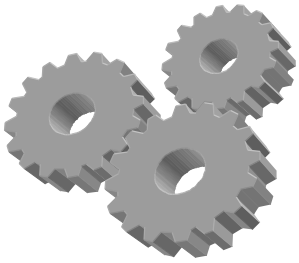


Base = 2^10 = 1024

Talk outline…

- **Motivation**
  - ④ Matrix Multiplication/Transposition
  - ④ Static Searches in Bal. Bin. Trees
- **The Model**
- **CO-Sorting**
- **Some Analysis**
- CO-Sorting **Experiments**
- *Is the model an oversimplification ?*

What went Wrong?



What went Wrong?

Assumptions!

Is the model oversimplified?

In Practice

- Associativity (Not fully associative)
- Complicated Algorithms
  (Asymptotics hides disasters!)
- Unified Caches (Instruction Caches)
- TLB (not tall)
- Concurrency (Coherence misses: Xeon)
- Replacement Policy (4Gb Limit)
- Multiple disks (Can increse I/O speed)
- Write Through Caches
  (Causes misses even if problem fits into cache)

Elaborated Here

What did I learn from it?

DO's
- Scans
- D&C, Rec
- Blocking
- Inline ju
- Local acc
- Experime differen algorithn

Don't's
- Non-Loc
- Believe (Optima
- Ignore platforr
- Belive m
- Believe results.

Sometimes
- Sub-optimal can be better than optimal
- Cache Aware might be the way to go.
- more than one CO optimal algorithm is available and only one is good among them.

Known Optimal Results

- Matrix Multiplication
- Matrix Transpose
- n-point FFT
- LUP Decomposition
- Sorting
- Searching

Results Known

Priority Q (Ins/Del/Del Min)	$O(\frac{1}{B}\log_{\frac{M}{B}}\frac{N}{B})$
List Ranking	$O(sort(V))$
Tree Algos	$O(sort(V))$
Directed BFS/DFS	$O((V+\frac{E}{B})\log_2 V + sort(E))$
Undirected BFS	$O(V + sort(E))$
MSF	$O(sort(E)+\log_2\log_2 V)$

Results Known

Array Reversal	$O(scan(V))$
LU Decomposition	$\Theta(1 + \frac{N^2}{B} + \frac{N^3}{B\sqrt{M}})$
FFT	$O(sort(V))$
B-Trees (Insertions/Del)*	$O(\log_B n)$
Tree Layout	$OPT\ O(\log(B))$
Convex Hulls in 3D	$O(sort(N))$

New Result

Cache oblivious (With Edgar Ramos,UIUC)
Voronoi Diagrams in 2D/3D.

Applications:
- Delaunay Triangulations
- Curve/Surface Reconstruction
- Meshing

Publications

- Book : Algorithms for Memory Hierarchies
  Chapter: Cache Oblivious Algorithms
  (Editors Meyer et.al.)
- *Cache Oblivious Voronoi diagrams*
  (with Edgar Ramos, In Progress)
- *Minimum Enclosing Balls*
  (with Joe Mitchell, Alper Yildirim)
  (in Alenex 03) < Also cache oblivious >

Other Introductions

- Chapter by
  Eric Demaine

Acknowledgements

Michael Bender
Matteo Frigo
Petr Konecny
Joe Mitchell
Harold Prokop
Edgar Ramos
Peter Sanders
Alper Yildrim
Erez Zadok

Special thanks to MPI Saarbruecken where part of this work was done.