



Adaptive Cache-Efficient Algorithms

Presented by
Chung Shin Yee
chungsy@ihpc.a-star.edu.sg



Outline

- Motivation & Objectives
- Literature Review
- Focus & Partial Results
 - Cache-Oblivious All-to-All Operation
 - Cache-Oblivious 2D Jacobi's Method
 - Shortcoming of Cache-Oblivious Model
 - Adaptive Cache-Oblivious Algorithms
- Summary and Future Work

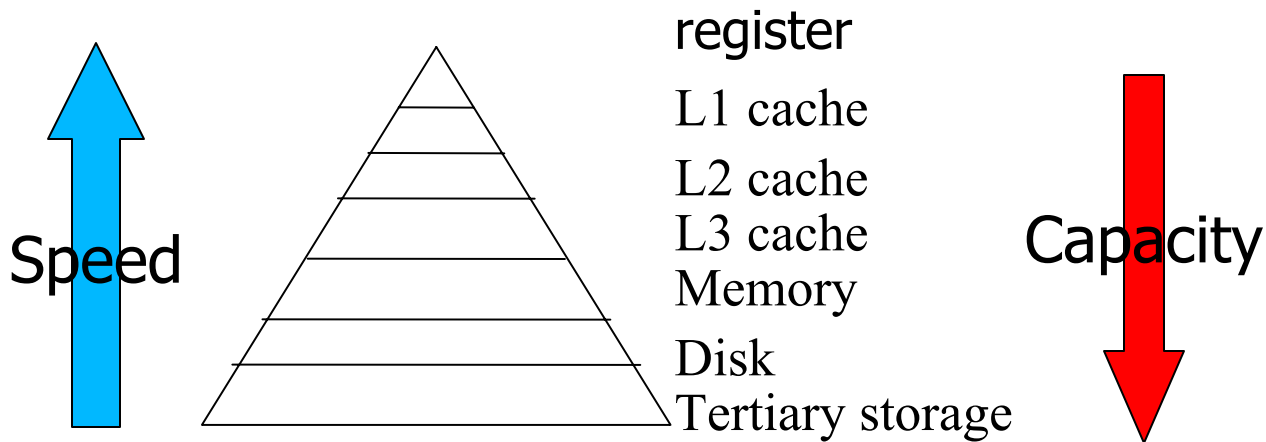


Background & Motivation

- Current Trend
 - Memory is much slower than processor.
 - Multiple platforms executions.
 - Frequent changes in hardware.
- Cache-Aware Optimizations
 - Require knowledge of hardware parameters.
 - Require expensive fine tuning efforts.
 - Not portable.
 - Optimize only for limited cache levels.

Objectives

- Requires no cache parameters.
- Simple & portable algorithms.
- Automatically **adapted** to the memory hierarchy.





Literature Review

- Cache-Oblivious (CO) Algorithms.
- Existing CO Algorithms.
- Analysis of CO Algorithms.
- Naïve & CO Matrix Transposition.



Cache-Oblivious Algorithms

- Cache-Oblivious Algorithms [Frigo 1999]
 - **Ideal-Cache** model.
 - Low cache complexity.
 - Results applicable to **multi-level cache**.
 - No prior knowledge about the cache parameters.
 - **Portable** performance.
 - No hand-tuning required.



Existing CO Algorithms

- Numerical algorithms
 - Matrix Transposition.
 - Matrix Multiplication.
 - Jacobi Multipass Filter.
 - Fast Fourier Transform.
- Searching algorithms
 - Van Emde Boas layout.
 - B-Tree.
 - Suffix Tree.



Existing CO Algorithms

- **Sorting algorithms**
 - Funnelsort & Lazy Funnelsort.
 - Distribution sort.
 - In-place proximity mergesort.
- **Priority queues**

Cache Complexity of CO Algorithms

L: Cache line size

Z: Cache size

m, n, p: Dimensions

Algorithm	Naïve	CO
Matrix Transposition	$O(mn)$	$O(mn/L)$
Matrix Multiplication	$O(mnp)$	$O(mnp/L\sqrt{Z})$
1D Jacobi's Method	$O(n^2/L)$	$O(n^2/LZ)$
Sorting	$O((n/L) \log n)$	$O((n/L) \log_z n)$
B-Tree Search	$O(\log n)$	$O(\log_L n)$

Naive Matrix Transposition

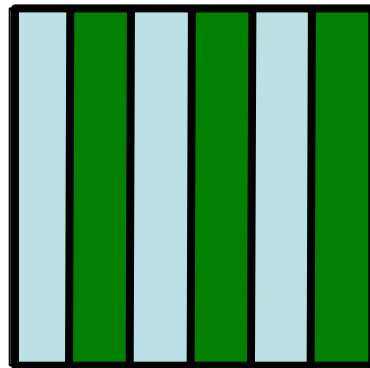
- Transpose $m \times n$ Matrix, $B = A^T$

for $p := 1$ to m

for $q := 1$ to n

$B(q,p) := A(p,q)$

**Column
Access!!!**



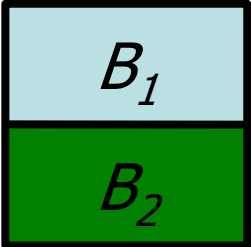
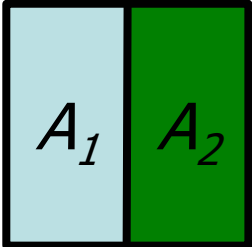
=

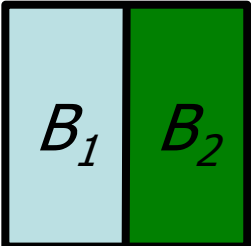
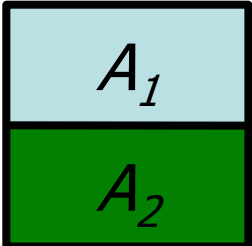


CO Matrix Transposition

- Transpose $m \times n$ matrix, $B = A^T$, [Frigo 1999]

Case 1:  =  if $m = n = 1$

Case 2:  =  if $m \geq n$

Case 3:  =  if $n > m$



Outline

- Motivation & Objectives
- Literature Review
- Focus & Partial Results
 - **Cache-Oblivious All-to-All Operation**
 - Cache-Oblivious 2D Jacobi's Method
 - Shortcoming of Cache-Oblivious Model
 - Adaptive Cache-Oblivious Algorithms
- Summary and Future Work

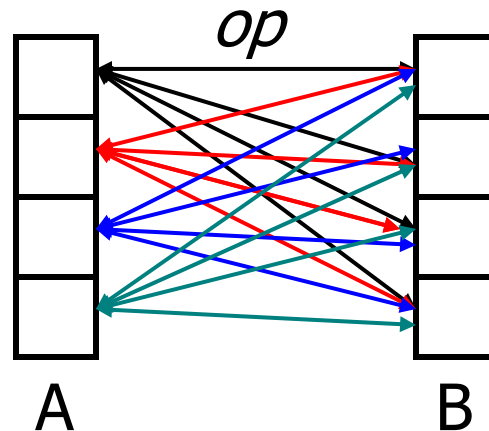
Naïve All-to-All Operation

- Applicable to *N-Body Simulation*.

for p := 1 to n

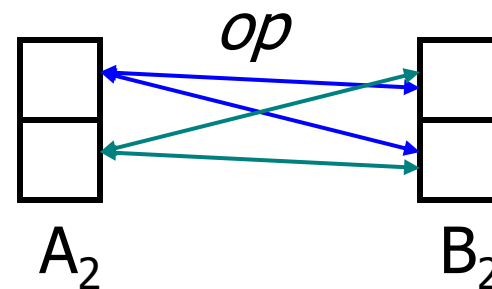
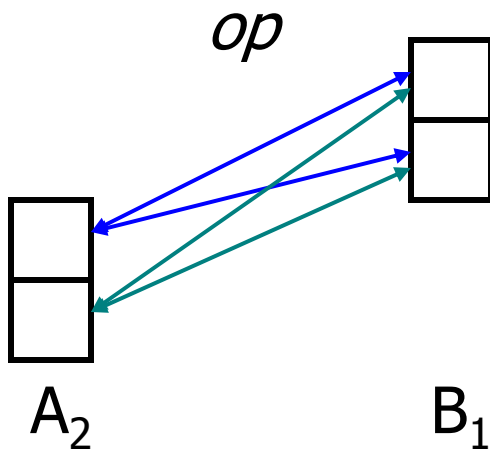
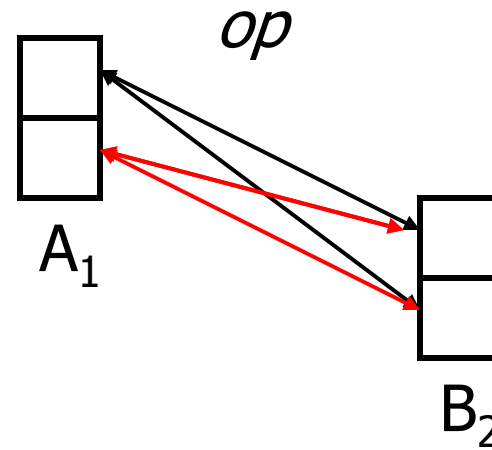
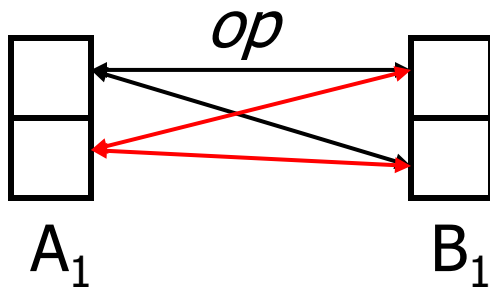
 for q := 1 to n

$(A_p, B_q) := A_p \text{ op } B_q$



CO All-to-All Operation

- Divide set A and set B recursively.





Cache Complexity of All-to-All Operation

- Cache Complexity
 - L = cache line size, Z = cache size, k = size of an element, n = number of elements in each set.
 - Naïve All-to-All Operation
 - $Q(m,n) = O(kn^2/L)$
 - CO All-to-All Operation
 - $Q(m,n) = O(k^2n^2/LZ)$



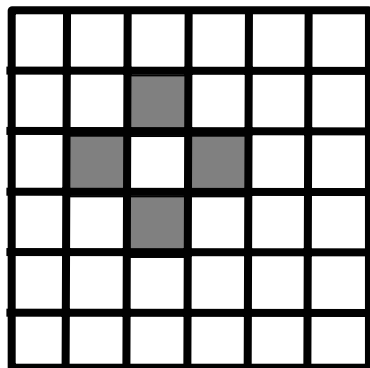
Outline

- Motivation & Objectives
- Literature Review
- Focus & Partial Results
 - Cache-Oblivious All-to-All Operation
 - **Cache-Oblivious 2D Jacobi's Method**
 - Shortcoming of Cache-Oblivious Model
 - Adaptive Cache-Oblivious Algorithms
- Summary and Future Work

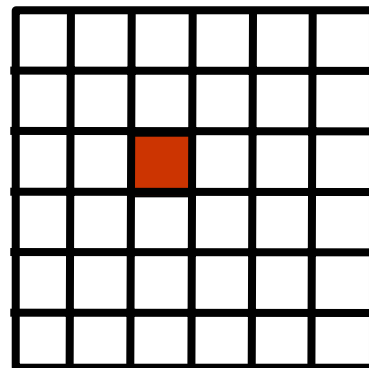
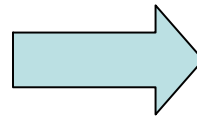
Naïve 2D Jacobi's Method

- Given $m \times n$ mesh M .

$$M(i,j)_{t+1} = \{ M(i+1,j)_t + M(i-1,j)_t \\ + M(i,j+1)_t + M(i,j-1)_t \} / 4$$







$m \times n$ mesh

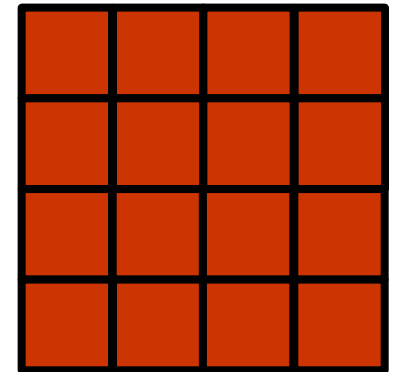
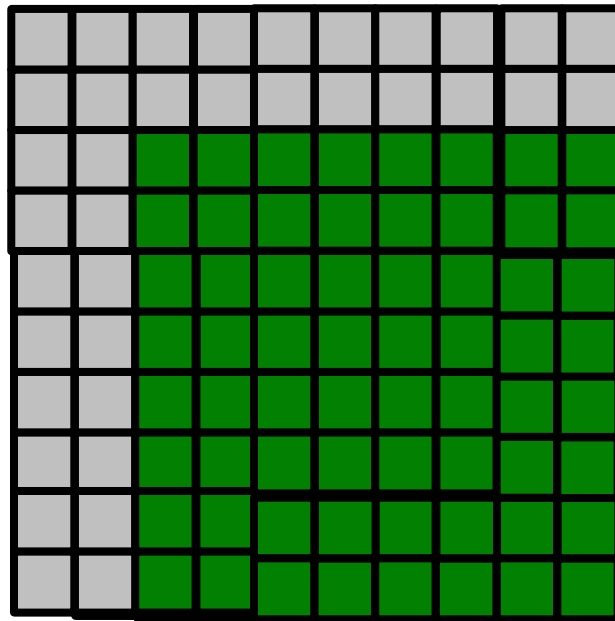


$m \times n$ mesh

CO 2D Jacobi's Method

- Given $m \times n$ mesh M .
 - Apply tiling using $p \times q$ tiles for δ steps.

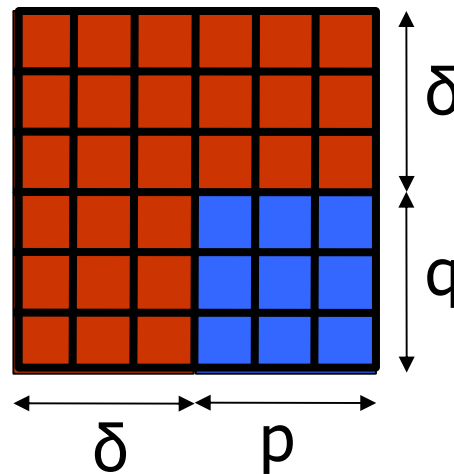
- 0: 
- 1: 
- 2: 
- 3: 



4 x 4 tile
with $\delta = 2$

CO 2D Jacobi's Method

- But what are the values p , q , δ ?



- # Cache misses for a tile computation
 - $O(1 + (p + \delta)(q + \delta)/L)$, L = cache line size



CO 2D Jacobi's Method

- Optimize the average # cache misses.
 - We get $\delta = \sqrt{pq}$.
- Given $n \times n$ mesh, compute n^2 steps.
- Compute n steps at a time.
- Divide & Conquer **recursively**.



CO 2D Jacobi's Method

- Cache Complexity
 - Given $m \times n$ mesh,
 L = cache line size, Z = cache size.
 - Naïve Jacobi's Method
 - $Q(n) = \Theta(n^4/L)$
 - CO Jacobi's Method
 - $Q(n) = O(n^4/L\sqrt{Z})$



Outline

- Motivation & Objectives
- Literature Review
- Focus & Partial Results
 - Cache-Oblivious All-to-All Operation
 - Cache-Oblivious 2D Jacobi's Method
 - **Shortcoming of Cache-Oblivious Model**
 - Adaptive Cache-Oblivious Algorithms
- Summary and Future Work



Shortcoming of CO Model

- Cache-Oblivious algorithms.
 - Good cache performance.
 - Poor execution time!!!
 - Slower than some 'naïve' algorithms.
- Mainly due to function call overhead.
 - # function calls in Matrix Transposition
 - $2mn - 1$ (worst case)
- Needs **Adaptivity**.



Outline

- Motivation & Objectives
- Literature Review
- Focus & Partial Results
 - Cache-Oblivious All-to-All Operation
 - Cache-Oblivious 2D Jacobi's Method
 - Shortcoming of Cache-Oblivious Model
 - **Adaptive Cache-Oblivious Algorithms**
- Summary and Future Work

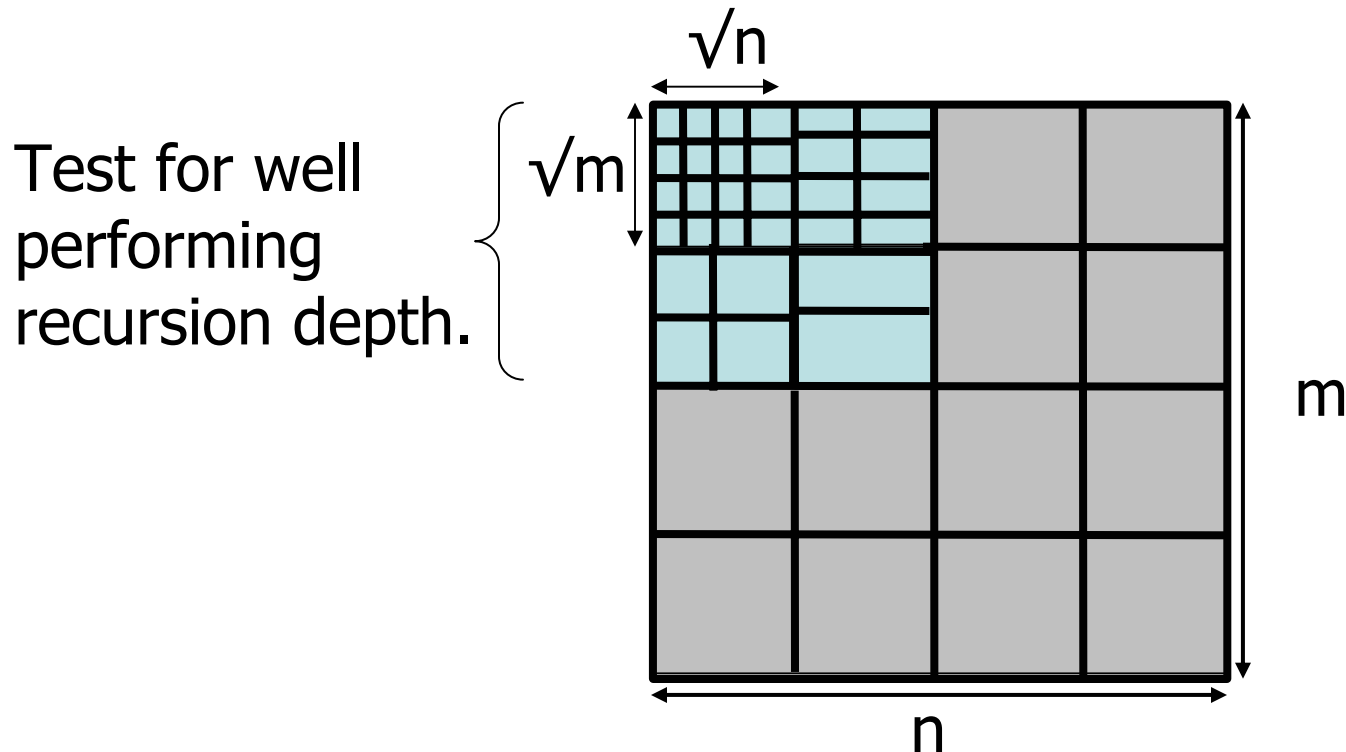


Adaptive CO Algorithms

- Reduce number of function calls.
- Stop recursion early.
- Use low overhead timing function.
- Self-optimized for a good recursion depth.
- Same strategy can be applied to many Cache-Oblivious algorithms.

ACO Matrix Transposition

- $h = \text{Depth of full recursion} = \lg(mn)$.
- Initial recursion up to $h/2 = 0.5\lg(mn)$.





ACO Matrix Transposition

- Depth of full recursion = h .
- # function calls with full recursion
= $2^{\lg(mn)+1} - 1 = 2mn - 1$
- Initial recursion stop at $h/2$.
- # initial function calls
= $2^{0.5\lg(mn)+1} - 1 = 2\sqrt{(mn)} - 1$
- # timing function calls
= $2 \times 0.5\lg(mn) = \lg(mn)$

Experiment Setup

■ Platforms

	AMD Athlon	Pentium 4	Itanium II
Clock Speed	1.2GHz	3GHz	1.4GHz
L1 Line Size	64B	64B	64B
L1 Cache Size	64KB	16KB	16KB
L1 Set-Assoc	2-way	8-way	4-way
L2 Line Size	64B	64B	128B
L2 Cache Size	256KB	1024KB	256KB
L2 Set-Assoc	16-way	8-way	8-way
L3 Line Size	-	-	128B
L3 Cache Size	-	-	1.5MB
L3 Set-Assoc	-	-	4-way per MB
Memory Size	512MB	512MB	4GB



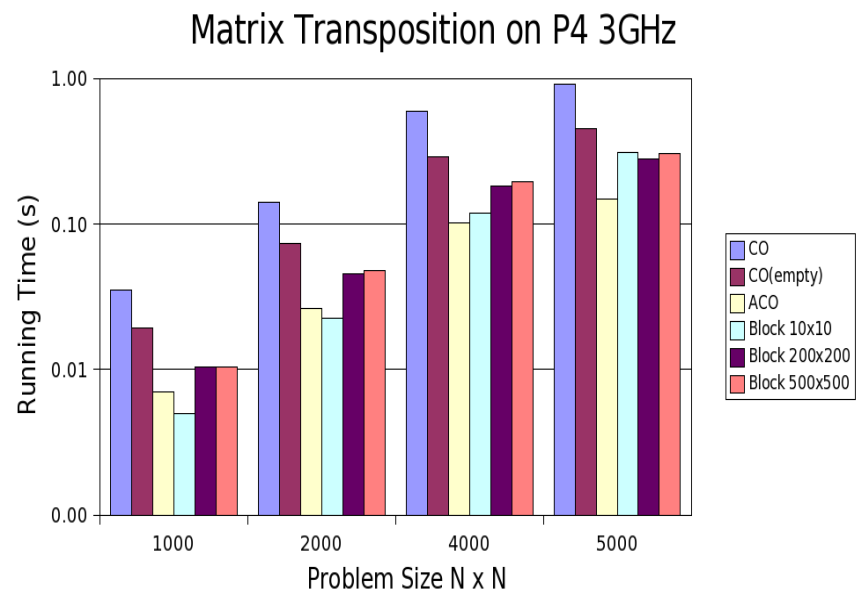
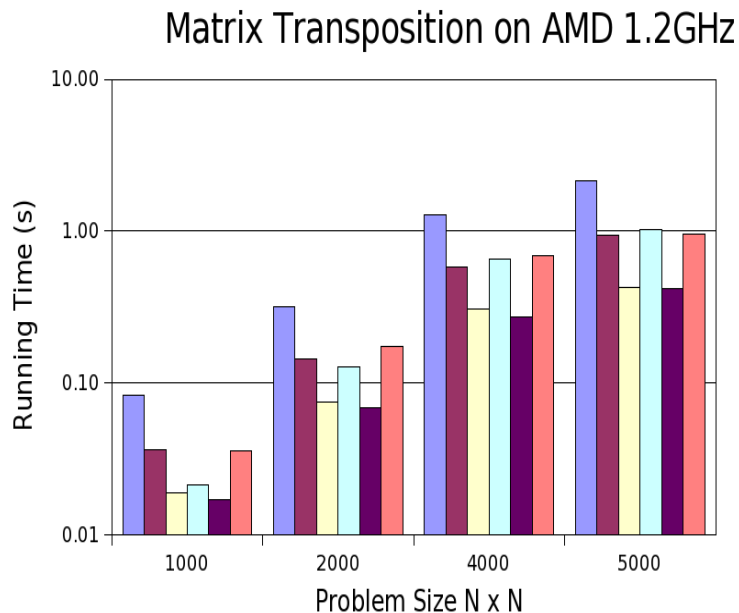
Experiment Setup

- Methodology

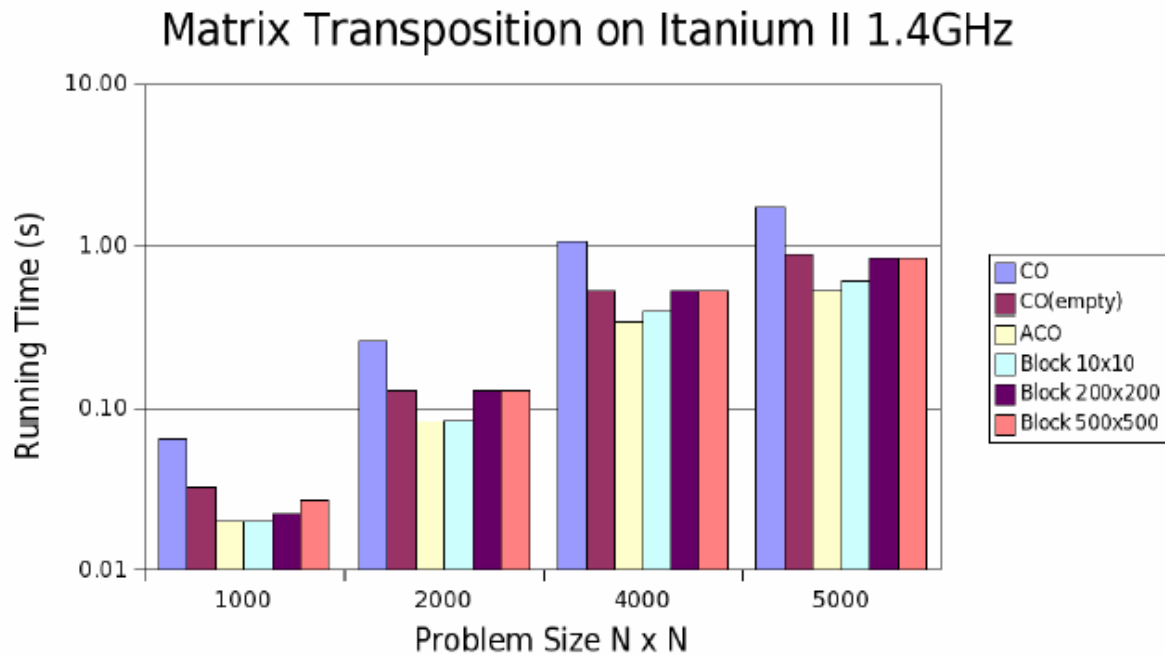
- GCC g++ 3.4.1 with '-O3' optimizations.
- Running times are measured using *clock()*.
- Cache events are measured using Linux Kernel's built-in profiler – *Oprofile*.

Experimental Results

- Recursion overhead is significant.
- ACO offers good performance.

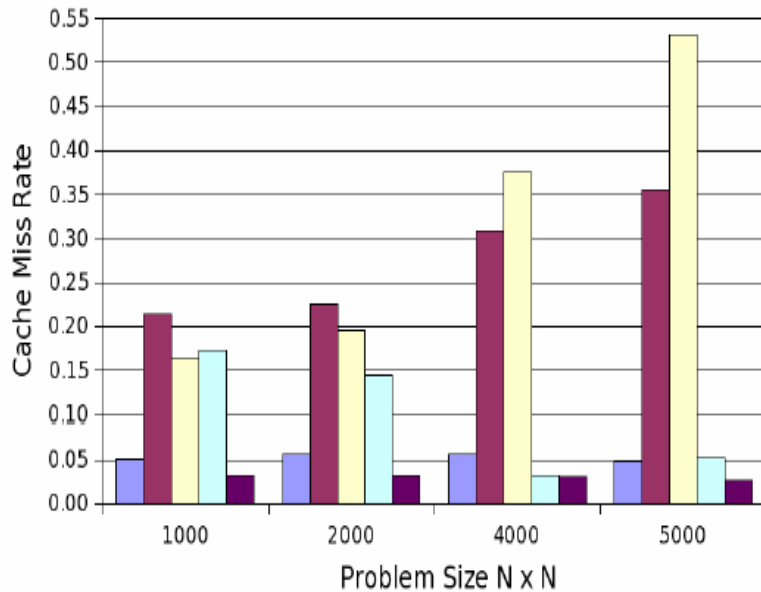


- Similar results in Itanium II platform.

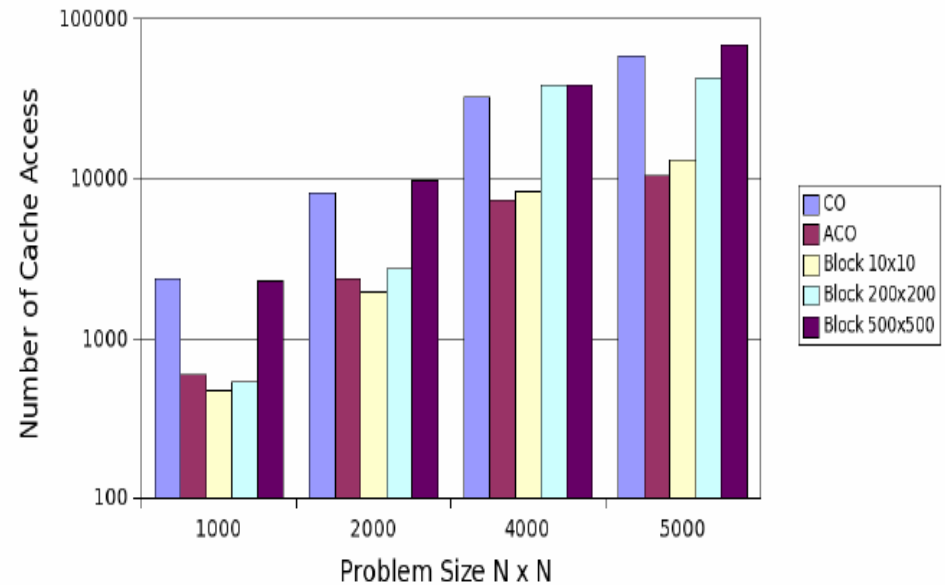


- CO has very low cache miss rate.
- However, CO incurs a lot of cache accesses.

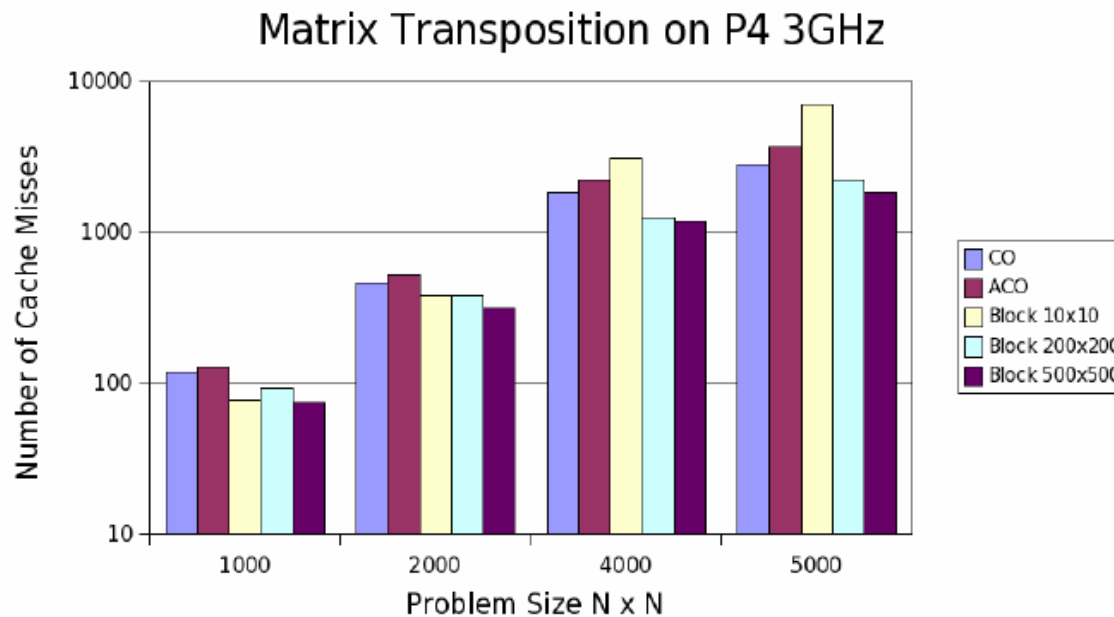
Matrix Transposition on P4 3GHz



Matrix Transposition on P4 3GHz



- CO and ACO have similar cache misses.





Discussions

- Good cache performance.
- Large computations, function calls, and cache accesses overhead.
- Adaptive CO Matrix Transposition performs well in multiple platforms, comparably with Tiling approach.
- Automatically **adapted** to Memory hierarchy.



Outline

- Motivation & Objectives
- Literature Review
- Focus & Partial Results
 - Cache-Oblivious All-to-All Operation
 - Cache-Oblivious 2D Jacobi's Method
 - Shortcoming of Cache-Oblivious Model
 - Adaptive Cache-Oblivious Algorithms
- **Summary and Future Work**



Summary

- Reviewed related work on matrix multiplication, matrix transposition, sorting, searching algorithms.
- Presented new CO All-to-All Operations.
- Presented new CO 2D Jacobi's Method.
- Presented new Adaptive CO Matrix Transposition and its experimental results.



Summary

- Main claims

- Naïve All-to-All Operation incurs $\Theta(1+kn/L+kn^2/L)$ cache misses.
- CO All-to-All Operation incurs $\Theta(1+kn/L+k^2n^2/z^2+k^2n^2/LZ)$ cache misses.
- Naïve 2D Jacobi's Method incurs $\Theta(n^4/L)$ cache misses for n^2 steps on $n \times n$ mesh.
- CO 2D Jacobi's Method incurs $O(n^4/L\sqrt{Z})$ cache misses for n^2 steps on $n \times n$ mesh.



Summary

- CO Matrix Transposition incurs $O(mn)$ recursive function calls.
- When the cache size is large, ACO Matrix Transposition incurs only $O(\sqrt{mn})$ recursive function calls.
- ACO Matrix Transposition incurs $O(\lg mn)$ timing function calls.
- ACO Matrix Transposition performs comparably with Tiled Matrix Transposition on multiple platforms.



Future Work and Plan

- Analyze and design adaptive schemes for CO Matrix Multiplication, Matrix Transposition, Jacobi's Method, N-Body Simulation.
- Research on cache conflicts problems.
- Research on data layout transformation.
- Research on software-controlled prefetching.
- Research on Translation Look-aside Buffer (TLB) misses problems.



Future Work and Plan

- Implement high performance and portable linear algebra library.
- Research on Cache-Oblivious data structures.
- Research on Models for Parallel Cache-Oblivious algorithms.



Thank You

References:

Frigo et al. Cache-Oblivious Algorithms. In *40th Annual Symp. on Foundations of Comp. Sci.* pp. 285. 1999.

M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, pp. 1381-1384, Seattle, WA, May 1998.

S. Y. Chung and W. J. Hsu. Impact of Cache-Oblivious Parallel Matrix Multiplication on a Cluster of Computers. In *Proc. Intl. Conf. on Scientific & Engineering Computation*, 2004.

S.Y. Chung and W. J. Hsu. Adaptive Cache-Oblivious Algorithms. In preparation.

Cache-Aware Optimization Techniques



- Tiling.
- Layout transformation.
- Array copying & padding.
- Cache aligning.
- Compile-time hardware parameter tuning.



Ideal-Cache Model

- Ideal-Cache model
 - Cache line size L and cache size Z .
 - Fully associative.
 - Optimal offline replacement scheme.
 - Tall cache – $Z = \Omega(L^2)$
 - Z : Cache size
 - L : Cache line size
 - Inclusion property.



CO Matrix Transposition

- Cache Complexity

- $L =$ Cache line size and $Z =$ Cache size.

- $Q(m,n) \leq \left\{ \begin{array}{ll} \Theta(1+mn/L) & \text{if } mn \leq cZ, \\ 2Q(m/2,n) + O(1) & \text{if } mn > cZ, \\ 2Q(m,n/2) + O(1) & \text{if } mn > cZ \end{array} \right\}$

- $Q(m,n) = 2^{\lg(mn/pq)} (1+pq/L), \max pq \leq cZ$
 $= \Theta(1+mn/L), \text{ since } pq \approx cZ$



Matrix Transposition

- Cache Complexity (# cache misses)
 - Naïve Matrix Transposition
 - $Q(m,n) = O(mn)$
 - **CO** Matrix Transposition
 - $Q(m,n) = \Theta(1+mn/L)$, $L =$ Cache line size
- Work Complexity (time complexity)
 - $W(m,n) = \Theta(mn)$



CO All-to-All Operation

- Cache Complexity

- L = Cache line size and Z = Cache size,
 k = size of an element,
 n = number of elements in each set.

- $Q(n) \leq \begin{cases} \Theta(1+kn/L) & \text{if } kn \leq cZ, \\ 4Q(n/2) + O(1) & \text{if } kn > cZ, \end{cases}$

$$Q(n) = 4^{\lg(kn/kp)} (1+kp/L), \max kp \leq cZ$$
$$= O(k^2n^2/Z^2+k^2n^2/LZ), \text{ since } kp \approx cZ$$



Jacobi's Method

- Optimize average cache misses per step
 - $L =$ cache line size.
 - $Q(p,q,\delta) = O(1 + (p + \delta)(q + \delta)/L)$
 - Number of tiles = mn/pq
 - # cache misses for δ steps
 - $Q(m,n,\delta) = O(mn/L + m\delta/pL + n\delta/pL + mn\delta^2/pqL)$
 - Average # cache misses per step
 - $Q(m,n) = O(mn/\delta L + m/pL + n/pL + mn\delta/pqL)$



Jacobi's Method

- $Q(m,n) = O(mn/\delta L + m/pL + n/pL + mn\delta/pqL)$
 $dQ(m,n)/d\delta = -mn/\delta^2 L + mn/pqL = 0$
- $\delta^2 = pq$
- $\delta = \sqrt{pq}$



CO Jacobi's Method

- Cache Complexity

- $L =$ cache line size, $Z =$ cache size.
- $Q(n,n) \leq \{O(1+(n+n)^2/L), \text{ if fit in } C,$
 $8Q(n/2,n/2) + O(1), \text{ if ...}$
- $Q(n,n) = 8^{\lg(n/p)}(p+\delta)^2/L$
 $= O((n/p)^3 Z/L), \text{ as } \delta = p$
- Average cache misses per step
 $= O(n^2/L\sqrt{Z})$



ACO Matrix Transposition

- Depth of full recursion = h .
- # function calls with full recursion
 $= 2^{\lg(mn)+1} - 1 = 2mn - 1$
- Initial recursion stop at $h/2$.
- # initial function calls
 $= 2^{0.5\lg(mn)+1} - 1 = 2\sqrt{(mn)} - 1$
- # timing function calls
 $= 2 \times 2^{0.5\lg(mn)} = 2\sqrt{(mn)}$