

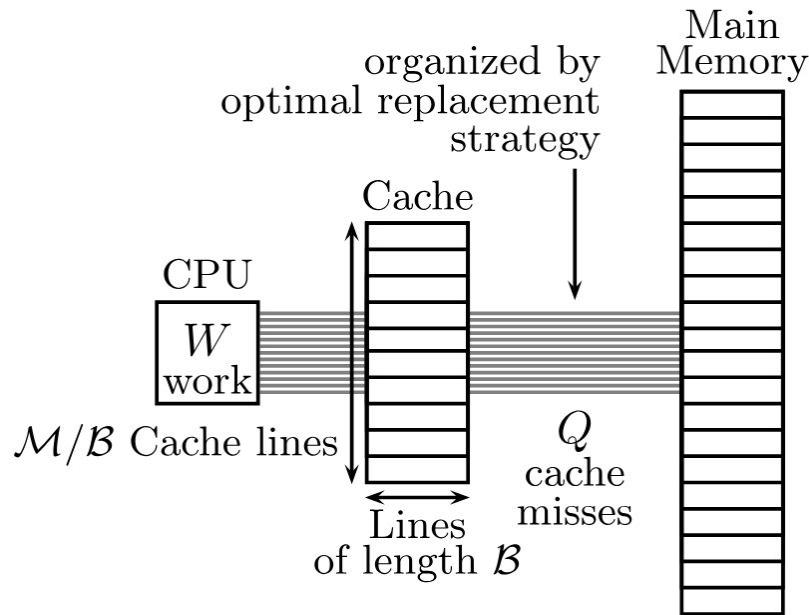
Cache-Oblivious Algorithms

MATTEO FRIGO, CHARLES E. LEISERSON, HARALD PROKOP, and
SRIDHAR RAMACHANDRAN, MIT Laboratory for Computer Science

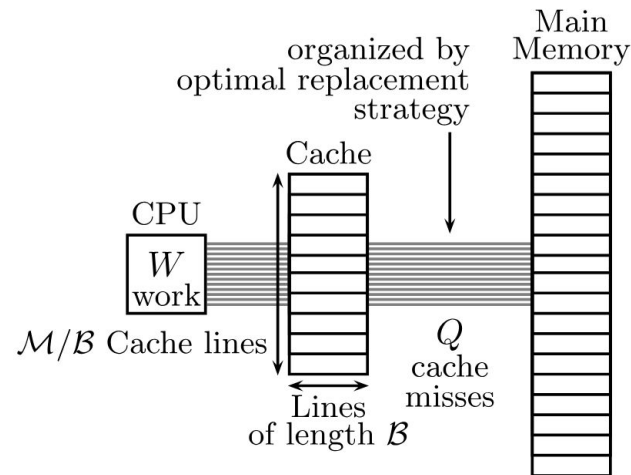
Mero Elmarassy

Ideal cache model

- Computer with an infinite main memory and an M -word ideal cache
- Cache lines are B words long
- The cache is tall: $M = \Omega(B^2)$
- Uses an optimal replacement strategy
- There are two complexity measures: the work complexity $W(n)$ and the cache complexity $Q(n; M, B)$



Ideal cache model assumptions



- Optimal replacement: replaces cache lines whose next access is furthest in the future
- Two memory levels: only cache and main memory
- Automatic replacement: no manual management of when to load a cache line
- Full associativity: any memory block can be stored in any cache line

Cache-aware algorithms: matrix multiplication

- Multiplying two $n \times n$ matrices A and B
- Ordinary $O(n^3)$ algorithm ORD-MULT
- TILED-MULT reduces cache misses by tiling
- For best performance, three $s \times s$ tiles (for A , B , C) fit into cache.
- So s is a tunable parameter and this is a cache aware algorithm.

ALGORITHM: TILED-MULT(A, B, C, n)

```
1  for  $i \leftarrow 1$  to  $n/s$ 
2      do for  $j \leftarrow 1$  to  $n/s$ 
3          do for  $k \leftarrow 1$  to  $n/s$ 
4              do ORD-MULT( $A_{ik}, B_{kj}, C_{ij}, s$ )
```

0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63

┘

Cache-aware algorithms: matrix multiplication analysis

- Choose the largest s so that three $s \times s$ matrices fit into cache
- $s \times s$ matrix needs $\Theta(s + \frac{s^2}{B})$ cache lines, and $s = \Theta(\sqrt{M})$
- ORD-MULT does $\frac{M}{B} = \Theta(s^2/B)$ cache misses and is called $\Theta((\frac{n}{s})^3)$ times
- The algorithm reads $\Theta(n^2)$ elements on $\Theta(1 + \frac{n^2}{B})$ lines
- So the total complexity is $\Theta(1 + \frac{n^2}{B} + \frac{n^3}{s^3} \frac{s^2}{B}) = \boxed{\Theta(1 + \frac{n^2}{B} + \frac{n^3}{B\sqrt{M}})}$

┘

ALGORITHM: TILED-MULT(A, B, C, n)

```
1  for  $i \leftarrow 1$  to  $n/s$ 
2      do for  $j \leftarrow 1$  to  $n/s$ 
3          do for  $k \leftarrow 1$  to  $n/s$ 
4              do ORD-MULT( $A_{ik}, B_{kj}, C_{ij}, s$ )
```

Cache-oblivious algorithm for matrix multiplication

- Want to compute $C_{m \times p} = A_{m \times n} B_{n \times p}$
- REC-MULT is a cache oblivious algorithm that solves this by computing $C \leftarrow C + AB$
- It uses $\Theta(mnp)$ work, which is the same as TILED-MULT
- Recursively divides the largest of the three dimensions m , n , and p
- If a subproblem fits into cache, so do all of its subproblems

┘

The REC-MULT algorithm

- REC-MULT recursively divides the largest dimension
- If $m \geq \max(n, p)$, split A and C to recursively compute $C_1 \leftarrow C_1 + A_1 B$ and $C_2 \leftarrow C_2 + A_2 B$

$$\begin{pmatrix} C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} B$$

- If $n \geq \max(m, p)$, split along shared n dimension and accumulate into C

$$C = \begin{pmatrix} A_1 & A_2 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = A_1 B_1 + A_2 B_2$$

- If $p \geq \max(m, n)$, split B and C

$$\begin{pmatrix} C_1 & C_2 \end{pmatrix} = A \begin{pmatrix} B_1 & B_2 \end{pmatrix}$$

REC-MULT cache complexity

- Choose $\alpha > 0$ so that the subproblem fits in cache when

$$\max(m, n, p) \leq \alpha\sqrt{M}$$

- Analysis depends on how many dimensions are larger than $\alpha\sqrt{M}$
- Case I: all three dimensions are large ($m, n, p > \alpha\sqrt{M}$)
- Case II: exactly two dimensions are large
- Case III: exactly one dimension is large
- Case IV: all three dimensions are small ($m, n, p \leq \alpha\sqrt{M}$)

REC-MULT cache complexity: Case I

- For case I we have $m, n, p > \alpha\sqrt{M}$
- The cache complexity satisfies the recurrence relation

$$Q(m, n, p) \leq \begin{cases} \Theta\left(\frac{mn + np + mp}{B}\right), & m, n, p \leq \alpha\sqrt{M} \\ 2Q(m/2, n, p) + O(1), & m \geq n, m \geq p \\ 2Q(m, n/2, p) + O(1), & n > m, n \geq p \\ 2Q(m, n, p/2) + O(1), & p > m, p > n \end{cases}$$

- Recursion stops when all three dimensions are $\Theta(\sqrt{M})$

- The solution is $\boxed{Q(m, n, p) = \Theta\left(\frac{mnp}{B\sqrt{M}}\right)}$

REC-MULT cache complexity: Case II

- Case II is when one dimension is small. As an example use

$$m \leq \alpha\sqrt{M}, \quad n, p > \alpha\sqrt{M}$$

- The recurrence is of the form

$$Q(m, n, p) = \begin{cases} \Theta\left(1 + n + \frac{np}{B} + m\right), & n, p \leq \alpha\sqrt{M} \\ 2Q(m, n/2, p) + O(1), & n \geq p \\ 2Q(m, n, p/2) + O(1), & p > n \end{cases}$$

- Solving this gives $Q(m, n, p) = \Theta\left(\frac{np}{B}\right) = \Theta\left(\frac{np}{B} + \frac{mnp}{B\sqrt{M}}\right)$

REC-MULT cache complexity: Case III

- Case III is when two dimensions are small. As an example use

$$n, p \leq \alpha\sqrt{M}, \quad m > \alpha\sqrt{M}$$

- The recurrence is of the form

$$Q(m, n, p) = \begin{cases} \Theta(1 + m), & m = \alpha\sqrt{M} \\ 2Q(m/2, n, p) + O(1), & \text{otherwise} \end{cases}$$

- Solving this gives

$$Q(m, n, p) = \Theta(m) = \Theta\left(m + \frac{mnp}{B\sqrt{M}}\right)$$

REC-MULT cache complexity: Case IV

- For case IV all of the dimensions are small and the whole problem fits in cache, so we get $Q(m, n, p) = \Theta \left(1 + \frac{mn+np+mp}{B} \right)$
- Combining all four cases gives the total cache complexity,

$$Q(m, n, p) = \Theta \left(m + n + p + \frac{mn + np + mp}{B} + \frac{mnp}{B\sqrt{M}} \right)$$

- Using $m = p = n$, REC-MULT has the same work and cache complexity as TILED-MULT

A cache-oblivious matrix transpose algorithm

- REC-TRANSPOSE transposes an $m \times n$ matrix A into an $n \times m$ matrix B
- If $n \geq m$, split A vertically and B horizontally:

$$A = (A_1 \ A_2), \quad B = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$$

and recursively transpose (A_1, B_1) and (A_2, B_2)

- If $m > n$, split A horizontally and B vertically:

$$A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}, \quad B = (B_1 \ B_2)$$

and recursively transpose (A_1, B_1) and (A_2, B_2)

- This algorithm uses $O(mn)$ work and has a cache complexity of $O(1 + \frac{mn}{B})$, which is optimal

Model assumptions: optimal replacement

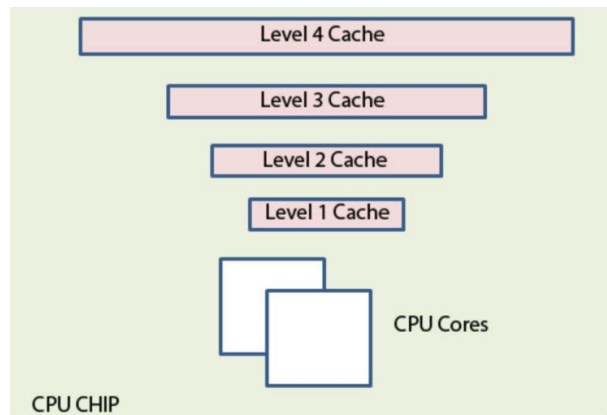
- Ideal cache uses optimal replacement instead of more realistic LRU
- The paper used a result from Sleator and Tarjan [1985] to show that

$$Q_{\text{LRU}}(n; M, B) \leq 2Q^*(n; M/2, B)$$

- Therefore if $Q(n; M, B) = O(Q(n; 2M, B))$, then using LRU does not change the asymptotic complexity
- So under this condition,

$$Q_{\text{LRU}}(n; M, B) = \Theta(Q(n; M, B))$$

Model assumptions: two cache levels



- Assume inclusion property: if data is in cache level i , it is also in level $i + 1$.
- Cache hits in levels lower than i are also hits in i
- If a miss in a lower level also misses in i , LRU would replace a line of i that is not in any lower levels.
- Therefore level i behaves like the first level of cache, and this model generalizes to apply to any number of cache levels with LRU replacement.

Performance of the REC-MULT and REC-TRANPOSE algorithms

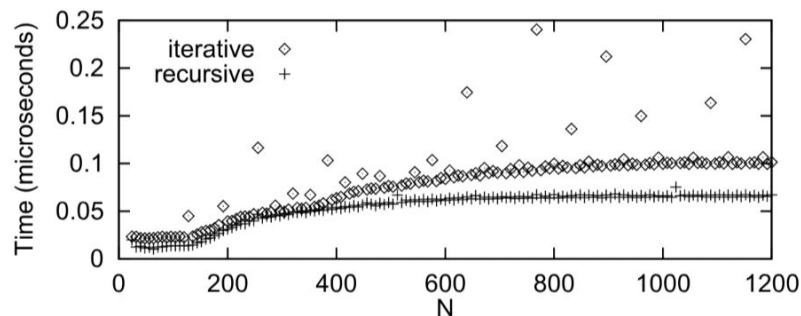


Fig. 4. Average time to transpose an $N \times N$ matrix, divided by N^2 .

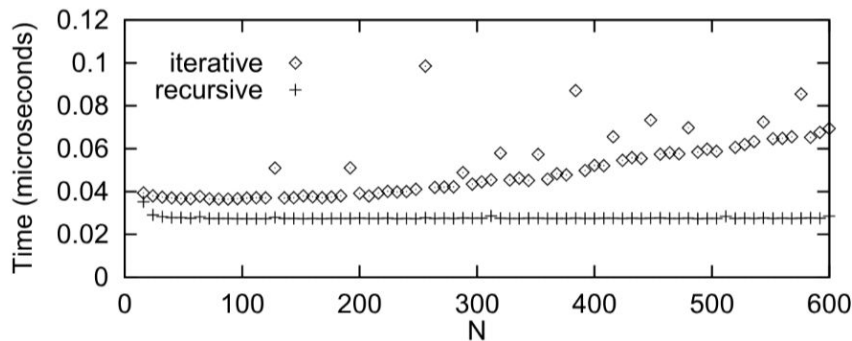


Fig. 5. Average time taken to multiply two $N \times N$ matrices, divided by N^3 .

Strengths and weaknesses of the paper

- The paper provides good justification for why the model assumptions are valid
- Cache oblivious algorithms are more portable to different machines
- They provide several optimal cache oblivious algorithms
- They gave only two performance comparison plots. In their REC-MULT performance comparison, it is not clear whether they compared against TILED-MULT or ORD-MULT.
- Topic for discussion: are the constant factors in these recursive algorithms actually beneficial? Recursive algorithms have large constant factors, might these outweigh the gains of fewer cache misses?