Cache-Oblivious Algorithms

By Matteo Frigo, Charles E. Leiserson, Harald Prokop, Sridhar Ramachandran

Why Cache-Oblivious Algorithms?

- Cache misses can be **expensive.**
- Not easy to optimize for all cache sizes.
- Cache-oblivious algorithms provide optimal cache-complexity regardless of cache properties.

Why Cache-Oblivious Algorithms?

Level	Size	Assoc.	Latency (ns)
Main	128 GB		50
LLC	30 MB	20	6
L2	256 KB	8	4
L1-d	32 KB	8	2
L1-i	32 KB	8	2

Figure 1: memory and cache access costs, from 6.172

Some Terminology

- Cache line: contiguous memory data imported to cache as a unit
- Cache size (Z): # cache words / cache
- Cache line size (L): # cache words / cache line
- Cache word typically 4 bytes, 8 bytes, etc.

Some Terminology

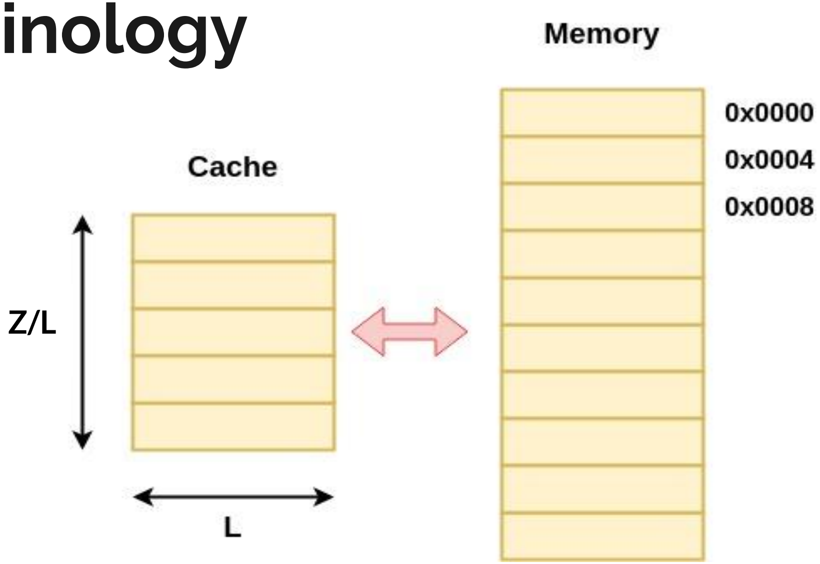

Figure 2: simple cache diagram

Ideal-Cache Model

- 1 limited-size cache, unlimited memory
- Cache fully-associative
- Optimal offline replacement strategy
- **Extra Assumption:** cache is tall:

$$Z = \Omega(L^2)$$

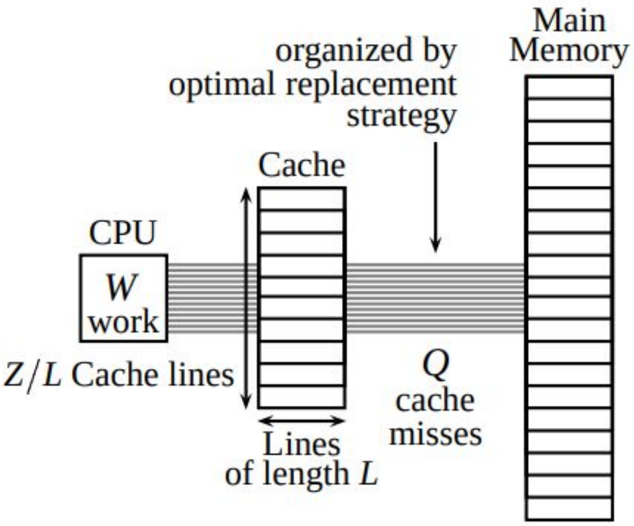

Figure 3: ideal-cache model

Cache-Aware Matrix Multiplication

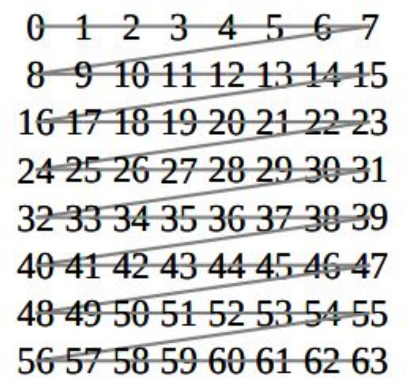

Figure 4: row-major order

1. let $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ be $n\ x\ n$ matrices in row-major order
2. **for** $i = 0$ to $n-1$
3.   **for** $j = 0$ to $n-1$
4.   **for** $k = 0$ to $n-1$
5. $C[i*n+j] = A[i*n+k] * B[k*n+j]$

Figure 4: naive matrix multiplication

Cache-Aware Matrix Multiplication

- Cache miss on each matrix access
- Cache Complexity: $Q(n) = \Theta(n^3)$

  Where $n > c\frac{Z}{L}$ for some c.

- Can do better!

let $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ be $n \ x \ n$ matrices in row-major order
for $i = 0$ to $n - 1$
  for $j = 0$ to $n - 1$
    for $k = 0$ to $n - 1$
      $C[i * n + j] = A[i * n + k] * B[k * n + j]$

Figure 4: naive matrix multiplication

Cache-Aware Matrix Multiplication

- Choose ***s*** s.t. $3 * s^2 \le Z$
- Cache Complexity:

$$Q(n) = (\frac{n}{s})^3 * \Theta(\frac{s^2}{L}) = \Theta(\frac{n^3}{\sqrt{Z} * L})$$

- Optimal cache complexity, but requires knowledge of cache properties.

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

Figure 5: block matrix multiplication

Cache-Aware Matrix Multiplication

- Optimal cache complexity without knowing ***L*** or ***Z*** ?
- Idea: Divide and Conquer!

Cache-Oblivious Matrix Multiplication

Split into $(\frac{n}{2}) \times (\frac{n}{2})$ block matrices and recurse:

$$\begin{pmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{pmatrix}$$

$$= \begin{pmatrix} \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21} & \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22} \\ \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21} & \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22} \end{pmatrix}$$

Figure 6: block matrix multiplication

Analysis

- Work: $W(n) = 8W(\frac{n}{2}) + \Theta(1) \Longrightarrow W(n) = \Theta(n^3)$ Optimal!
- Cache Complexity:

$$Q(n) = \begin{cases} \Theta(\frac{n^2}{L}) & n^2 \leq cZ \\ 8 \times Q(\frac{n}{2}) + \Theta(1) & o/w \end{cases}$$

Which means $Q(n) = \Theta(\frac{n^3}{L \times \sqrt{Z}})$ Optimal!

Cache-Oblivious Matrix Multiplication

Non-square case: Split ***A*** or ***B*** along biggest dimension:

- If ***m > max(n, p):***
- If ***n > max(m, p):***
- If ***p > max(m, n):***

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} B = \begin{pmatrix} A_1 B \\ A_2 B \end{pmatrix},$$

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

$$A \begin{pmatrix} B_1 & B_2 \end{pmatrix} = \begin{pmatrix} AB_1 & AB_2 \end{pmatrix}.$$

Figure 7 : recursion cases for matrix multiplication

Cache-Oblivious Matrix Multiplication

Theorem 1 *The* REC-MULT *algorithm uses* $\Theta(mnp)$ *work and incurs* $\Theta(m + n + p + (mn + np + mp)/L + mnp/L\sqrt{Z})$ *cache misses when multiplying an* $m \times n$ *matrix by an* $n \times p$ *matrix.*

Why Tall-Cache Assumption?

- Cache misses bring full row-major submatrix rows + useless data
- Submatrix might not fit in cache even if $3 \times s^2 \leq Z$

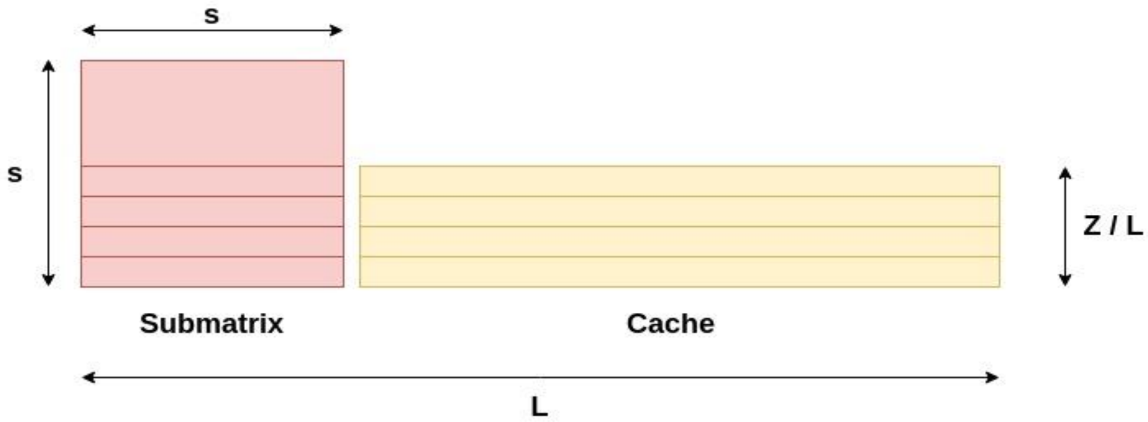

Figure 8: short cache

Cache-Oblivious Matrix Transposition

Cache-Oblivious Matrix Transposition

- Idea: Divide and Conquer
- Transpose each half of matrix ***A*** individually

Cache-Oblivious Matrix Transposition

- Idea: Divide and Conquer
- Transpose each half of matrix ***A*** individually

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}, B = A^T = \begin{bmatrix} A_1^T & A_3^T \\ A_2^T & A_4^T \end{bmatrix}$$

Figure 9: recursive transpose

Analysis

- Work: $W(n) = 4 \times W(\frac{n}{2}) + \Theta(1) \Longrightarrow W(n) = \Theta(n^2)$
- Cache complexity:

$$Q(n) = \begin{cases} \Theta(\frac{n^2}{L}) & n^2 \leq cZ \\ 4 \times Q(\frac{n}{2}) + \Theta(1) & o/w \end{cases} \Longrightarrow Q(n) = \Theta(\frac{n^2}{L})$$

- Cache complexity optimal. Rectangular case similar to multiplication.

Cache-Oblivious FFT

- Want to use cache-oblivious transposition as subroutine.
- Cache complexity: $Q(n) = O(1 + (n/L)(1 + \log_Z n))$

Cache-Oblivious Sorting

Funnelsort

1. Split the input into $n^{1/3}$ contiguous arrays of size $n^{2/3}$, and sort these arrays recursively.
2. Merge the $n^{1/3}$ sorted sequences using a $n^{1/3}$-merger, which is described below.

Work: $\Theta(n \log n)$

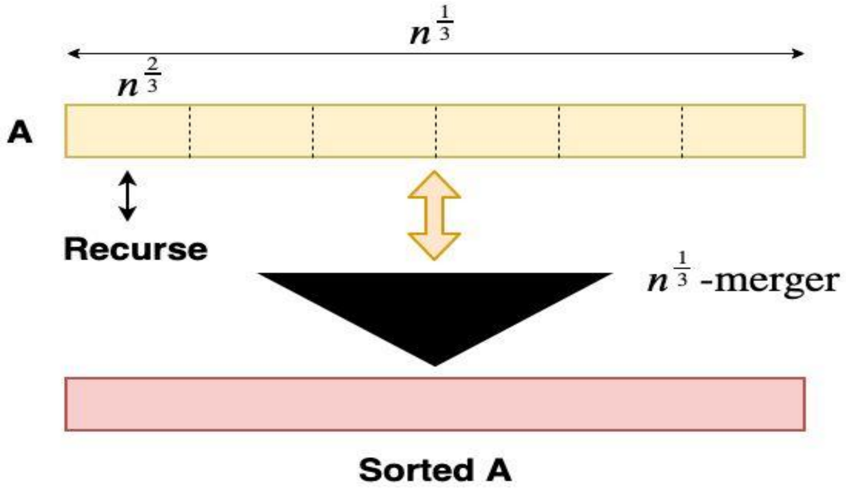

Figure 10: funnel sort

k-Merger

- Suspends merging when output sequence "long enough"
- More details in next presentation

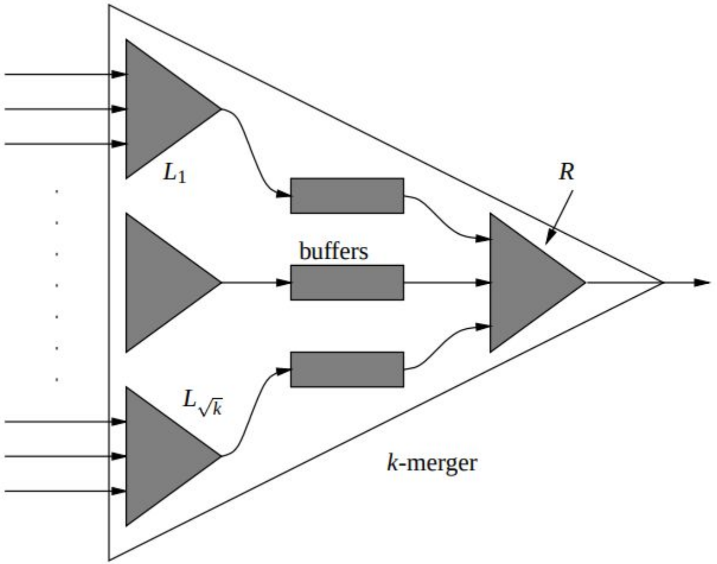

Figure 11: k-merger

Funnelsort Analysis

Lemma 6 *If $Z = \Omega(L^2)$, then a k-merger operates with at most*

$$Q_M(k) = O(1 + k + k^3/L + k^3 \log_Z k/L)$$

cache misses.

$$\implies Q_M(n^{\frac{1}{3}}) = O(n \times \frac{\log_Z n}{L})$$

Funnelsort Analysis

- $$Q(n) = n^{\frac{1}{3}} \times Q(n^{\frac{2}{3}}) + O(n \times \frac{\log_Z n}{L})$$
- Using induction:

$$Q(n) = O(\frac{n}{L} \times \log_Z n)$$

Distribution Sort

- $\Theta(n \log n)$ work.
- $Q(n) = O(\frac{n}{L} \times \log_Z n)$ cache complexity - optimal

Distribution Sort

1. Partition A into $\sqrt{n}$ contiguous subarrays of size $\sqrt{n}$. Recursively sort each subarray.
2. Distribute the sorted subarrays into q buckets $B_1, \ldots, B_q$ of size $n_1, \ldots, n_q$, respectively, such that
   1. $\max\{x \mid x \in B_i\} \le \min\{x \mid x \in B_{i+1}\}$ for $i = 1, 2, \ldots, q-1$.
   2. $n_i \le 2\sqrt{n}$ for $i = 1, 2, \ldots, q$.

   (See below for details.)
3. Recursively sort each bucket.
4. Copy the sorted buckets to array A.

```
DISTRIBUTE(i, j, m)
if m = 1
   then COPYELEMS(i, j)
   else DISTRIBUTE(i, j, m/2)
        DISTRIBUTE(i + m/2, j, m/2)
        DISTRIBUTE(i, j + m/2, m/2)
        DISTRIBUTE(i + m/2, j + m/2, m/2)
```

Theoretical Justifications for the Ideal Cache Model

Lemma 12 *Consider an algorithm that causes* $Q^*(n;Z,L)$ *cache misses on a problem of size* n *using a* (Z,L) *ideal cache. Then, the same algorithm incurs* $Q(n;Z,L) \leq 2Q^*(n;Z/2,L)$ *cache misses on a* (Z,L) *cache that uses LRU replacement.*

LRU competitive with optimal replacement.

Theoretical Justifications for the Ideal Cache Model

Corollary 13 *For any algorithm whose cache-complexity bound $Q(n;Z,L)$ in the ideal-cache model satisfies the regularity condition*

$$Q(n;Z,L) = O(Q(n;2Z,L)) \ , \qquad (14)$$

the number of cache misses with LRU replacement is $\Theta(Q(n;Z,L))$.

Theoretical Justifications for the Ideal Cache Model

- **Inclusion property:** cache level (i+1) contains all cache lines in level (i).
- Same-line elements in level (i) are same-line in level (i+1).
- More cache lines in level (i+1) than level (i).

Theoretical Justifications for the Ideal Cache Model

Lemma 14 *A (Z_i, L_i)-cache at a given level i of a multilevel LRU model always contains the same cache lines as a simple (Z_i, L_i)-cache managed by LRU that serves the same sequence of memory accesses.* □

Lemma 15 *An optimal cache-oblivious algorithm whose cache complexity satisifies the regularity condition (14) incurs an optimal number of cache misses on each level[3] of a multilevel cache with LRU replacement.*

Theoretical Justifications for the Ideal Cache Model

Lemma 16 *A* (Z, L) *LRU-cache can be maintained using* $O(Z)$ *memory locations such that every access to a cache line in memory takes* $O(1)$ *expected time.*

- Eliminates full-associativity and automatic replacement assumptions.
- Proof outline: hashtable - doubly-linked list LRU cache implementation in memory. LRU policy in O(1) expected time.

Preliminary Experimental Analysis

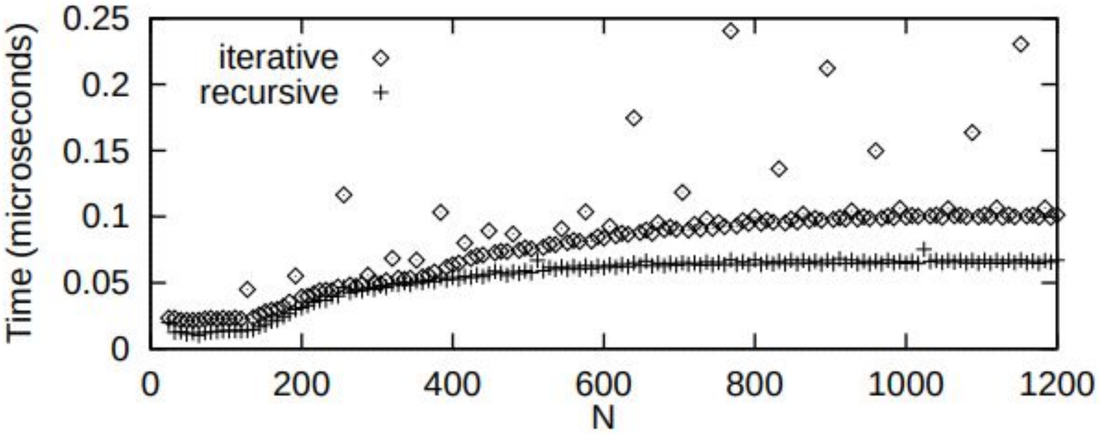

Figure 12: N x N matrix transposition runtime / N^2

Preliminary Experimental Analysis

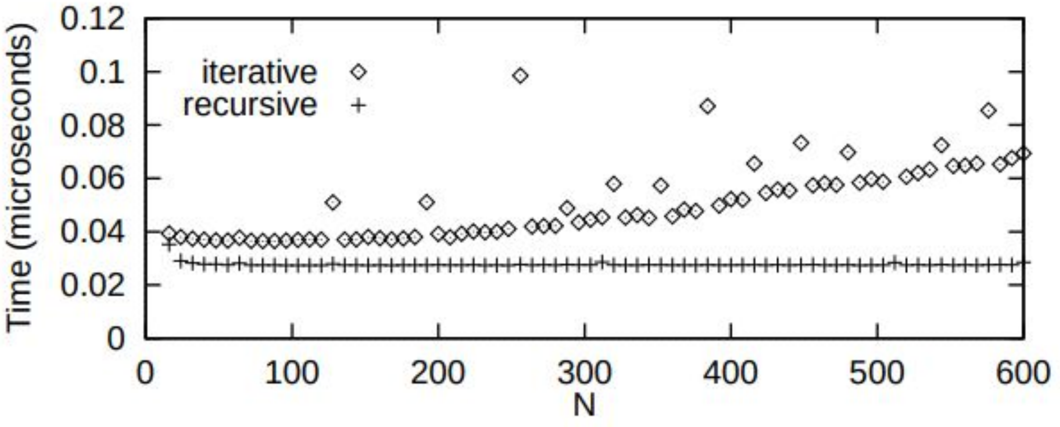

Figure 13: N x N matrix multiplication runtime / N^3

Strengths

- Novel approach to construct cache-efficient algorithms
- Plenty of detailed proofs for cache complexities

Weaknesses

- Hard to understand details of all proofs
- Could have presented experimental analysis of some same-work **cache-oblivious** vs **cache-aware** algorithms

Discussion Questions

- Are cache-oblivious algorithms more or less efficient than cache-aware algorithms?
- Does the recursion overhead overshadow the obtained cache efficiency?