# Cache-Oblivious Algorithms

## Paper Reading Group

Matteo Frigo
Charles E. Leiserson
Harald Prokop
Sridhar Ramachandran
Presents: Maksym Planeta

03.09.2015

# Table of Contents

# Table of Contents

# Matrix multiplication

ORD-MULT$(A, B, C)$

**for** $i \leftarrow 1$ **to** $m$
    **for** $j \leftarrow 1$ **to** $p$
        **for** $k \leftarrow 1$ **to** $n$
            $C_{ij} \leftarrow C_{ij} + A_{ik} \times B_{kj}$

# Matrix layout

Like in C . . .

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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 |

Figure: Row major order

# Matrix layout

Like in C . . .

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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 |

Figure: Row major order

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

Figure: Column major order

Or like in Fortran

# Cache friendly algorithm

BLOCK-MULT$(A, B, C, n)$

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

# BLOCK-MULT issues

Being cache aware is hard:

- Cumbersome structure
- Complicated choice of $s$
- Expensive mispicking of $s$
- Problematic if $n \mod s \neq 0$

# Motivation

- Keeping algorithm simple is nice.
- But cache effectiveness is the **must**.

# Table of Contents

# System model

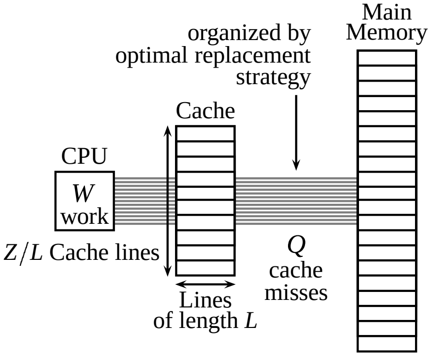


**Figure 1:** The ideal-cache model

- Two level memory
- Fully associative
- Strictly *optimal* replacement
- Automatic replacement
- Tall cache:

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

where:

$Z$ – number of words in the cache

$L$ – number of words in a cache line

# Matrix multiplication

Given: $A[m \times n] \times B[n \times p] \rightarrow C[m \times p]$

$$
\begin{aligned}
\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} B &= \begin{pmatrix} A_1 B \\ A_2 B \end{pmatrix}, & m \geq \max(n, p) \quad (1) \\
\begin{pmatrix} A_1 & A_2 \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} &= A_1 B_1 + A_2 B_2, & n \geq \max(m, p) \quad (2) \\
A \begin{pmatrix} B_1 & B_2 \end{pmatrix} &= \begin{pmatrix} AB_1 & AB_2 \end{pmatrix}, & p \geq \max(n, m) \quad (3) \\
C_{ij} &:= C_{ij} + A_{ik} \cdot B_{kj}, & m = n = p = 1 \quad (4)
\end{aligned}
$$

# Bounds

REC-MULT

Work: $\Theta(n^3)$ Cache misses: $\Theta(n + n^2/L + n^3/L\sqrt{Z})$

vs BLOCK-MULT

Work: $\Theta(n^3)$ Cache misses: $\Theta(1 + n^2/L + n^3/L\sqrt{Z})$

vs Strassen's [2] (cache oblivious)

Work: $\Theta(n^{\log_2 7})$ Cache misses: $\Theta(1 + n^2/L + n^{\log_2 7}/L\sqrt{Z})$

# Matrix transposition

Given: $A[m \times n] \rightarrow B[n \times m]$

$$A = \begin{pmatrix} A_1 & A_2 \end{pmatrix}, B = \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} \tag{5}$$

# Bounds

REC-TRANSPOSE

Work: $\Theta(n \cdot m)$

Cache misses: $\Theta(1 + mn/L)$

*Asymptotically optimal*

Naïve

Work: $\Theta(n \cdot m)$

Cache misses: $\Theta(n \cdot m)$

# Discrete Fourier Transform (DFT)

Compute:

$$Y[i] = \sum_{j=0}^{n-1} X[j]\omega_n^{-ij},$$

where $\omega_n = e^{2\pi\sqrt{-1}/n}$
Assume $n = 2^k \mid k \in \mathbb{N}$
Choose $n_1 = 2^{\lceil log_2 n/2 \rceil}$, $n_2 = 2^{\lfloor log_2 n/2 \rfloor}$
Factorized $Y$ (Cooley-Turkey algorithm):

$$Y[i_1 + i_2 n_1] = \sum_{j_2=0}^{n_2-1} \left[ \left( \sum_{j_1=0}^{n_1-1} X[j_1 n_2 + j_2]\omega_n^{-j_1 j_2} \right) \omega_{n_2}^{-j_1 j_2} \right]$$

# Sorting

Mergesort is not optimal with respect to cache misses.

1. Funnelsort
2. Distribution sort

- Recursive
- Asymptotically cache-optimal
- Not every recursive sort is cache optimal

# Funnelsort

1. Split input into $n^{\frac{1}{3}}$ of size $n^{\frac{2}{3}}$, and sort these arrays recursively
2. Merge $n^{\frac{1}{3}}$ sorted sequences using $n^{\frac{1}{3}}$-merger

# k-merger

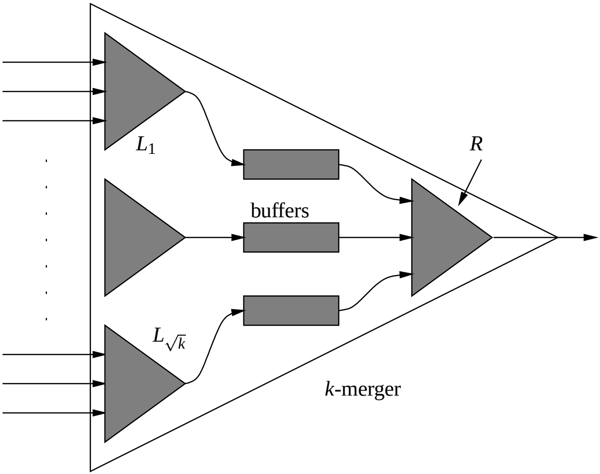


**Figure 3:** Illustration of a $k$-merger. A $k$-merger is built recursively out of $\sqrt{k}$ "left" $\sqrt{k}$-mergers $L_1, L_2, \ldots, L_{\sqrt{k}}$, a series of buffers, and one "right" $\sqrt{k}$-merger $R$.

# Bounds

Work: $O(n \cdot log_2 n)$
Optimal cache misses: $O(1 + (n/L)(1 + log_Z n))$

# Relieved system model

- LRU
  - $\Theta(Q(n; Z; L))$
- Multilevel cache
  - inclusive cache

# Table of Contents

# Micro-benchmarks

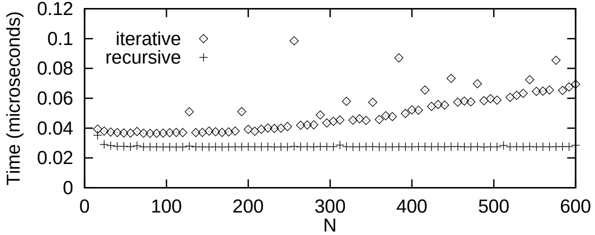


**Figure 5:** Average time taken to multiply two $N \times N$ matrices, divided by $N^3$.

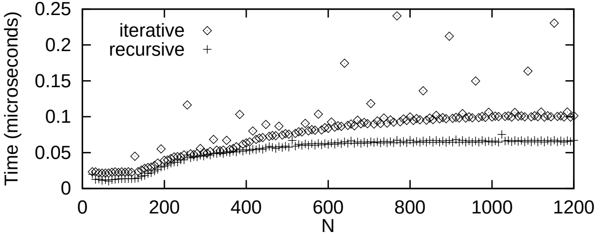


**Figure 4:** Average time to transpose an $N \times N$ matrix, divided by $N^2$.

# Real benchmarks [1]

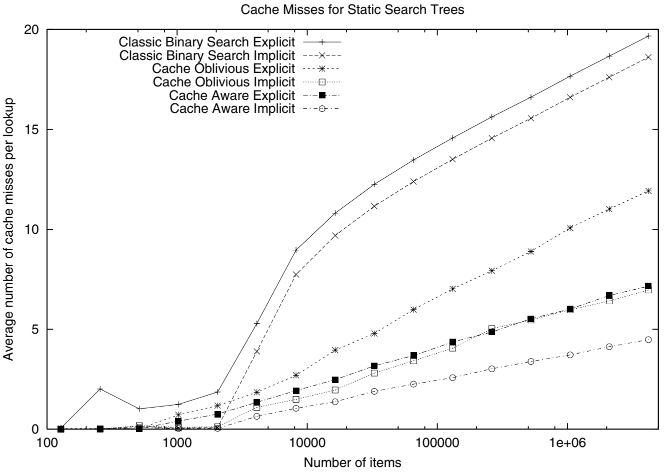


**Fig. 4.8.** Cache misses per lookup for static search algorithms

# Real benchmarks [1]

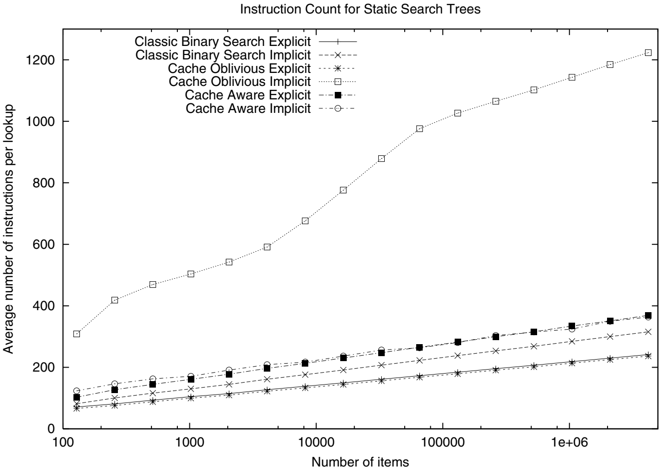


**Fig. 4.9.** Instruction count per lookup for static search algorithms

# Real benchmarks [1]

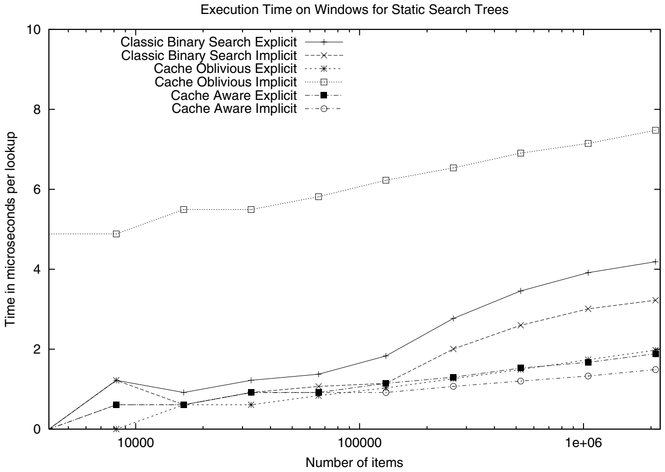


**Fig. 4.10.** Execution time on Windows for static search algorithms

# Table of Contents

# FFMK tribute slide

*. . . FFTW library, which uses a recursive strategy to exploit caches in Fourier transform calculations. FFTW's code generator produces straight-line "codelets", which are coarsened base cases for the FFT algorithm. Because these codelets are cache oblivious, a C compiler can perform its register allocation efficiently, and yet the codelets can be generated without knowing the number of registers on the target architecture.*

# Open questions

- Is there a gap in asymptotic complexity?
- Is there a limit as to how much better a cache-aware algorithm can be?

# Conclusion

- Seem to be slower
- Provide cache optimality without knowing cache size
- Based on recursion

Richard E Ladner, Ray Fortna, and Bao-Hoang Nguyen.
A comparison of cache aware and cache oblivious static search trees using program instrumentation.
In *Experimental Algorithmics*, pages 78–92. Springer, 2002.

Volker Strassen.
Gaussian elimination is not optimal.
*Numerische Mathematik*, 13(4):354–356, 1969.