Cache Friendly Sparse Matrix Vector Multilication

Sardar Anisual Haque[1], Shahadat Hossain[2], Marc Moreno Maza[1]

[1]University of Western Ontario, London, Ontario (Canada)

[2]Department of Computer Science, University of Lethbridge, Lethbridge, Alberta (Canada)

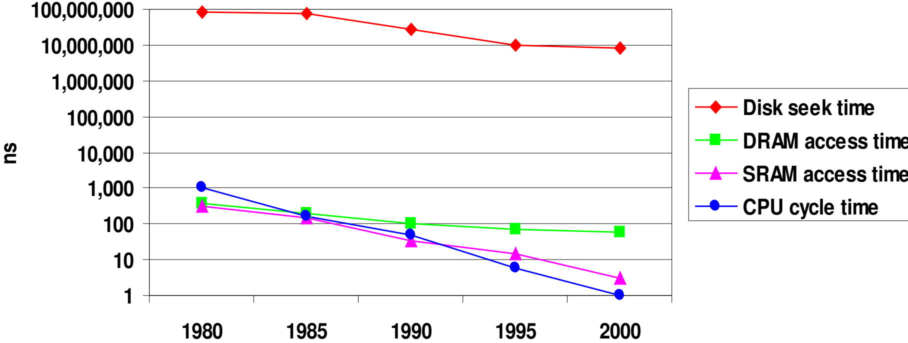
PASCO 2010

Plan

Once upon a time everything was slow in a computer

The CPU-Memory Gap

The increasing gap between DRAM, disk, and CPU speeds.



But I/O complexity was already there

STOC(Milwaukee 1981),326-333.

I/O COMPLEXITY: THE RED-BLUE PEBBLE GAME

Hong, Jia-Wei and H. T. Kung

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

In this paper, the *red-blue pebble game* is proposed to model the input-output complexity of algorithms. Using the pebble game formulation, a number of lower bound results for the I/O requirement are proven. For example, it is shown that to perform the n-point FFT or the ordinary nxn matrix multiplication algorithm with O(S) memory, at least $\Omega(n \log n/\log S)$ or $\Omega(n^3/\sqrt{S})$, respectively, time is needed for the I/O. Similar results are obtained for algorithms for several other problems. All of the lower bounds presented are the best possible in the sense that they are achievable by certain decomposition schemes.

decomposition schemes. The paper is organized according to the techniques used to derive these lower bounds.

In Section 2 we formally define the pebble game and point out its relation to the I/O problem. In Section 3 we show that lower bounds for I/O in the pebble game can be established by studying the so-called *S-partitioning problem*. This is the key result of the paper in the sense that it provides the basis for the derivation of all the lower bounds. In Section 4 we prove a lower bound for the FFT algorithm. Lower bounds in Section 5 are based on the *information speed function*, which

The foundation paper of cache complexity

Cache-Oblivious Algorithms

EXTENDED ABSTRACT

Matteo Frigo Charles E. Leiserson Harald Prokop Sridhar Ramachandran
MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139
{athena,cel,prokop,sridhar}@supertech.lcs.mit.edu

Abstract This paper presents asymptotically optimal algorithms for rectangular matrix transpose, FFT, and sorting on computers with multiple levels of caching. Unlike previous optimal algorithms, these algorithms are ***cache oblivious***: no variables dependent on hardware parameters, such as cache size and cache-line length, need to be tuned to achieve optimality. Nevertheless, these algorithms use an optimal amount of work and move data optimally among multiple levels of cache. For a cache with size Z and cache-line length L where $Z = \Omega(L^2)$ the number of cache misses for an $m \times n$ matrix transpose is $\Theta(1 + mn/L)$. The number of cache misses for either an n-point FFT or the sorting of n numbers is $\Theta(1 + (n/L)(1 + \log_Z n))$. We also give an $\Theta(mnp)$-work algorithm to multiply an $m \times n$ matrix by an $n \times p$ matrix that

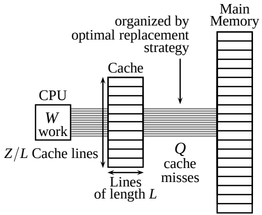


Figure 1: The ideal-cache model

The (Z, L) ideal cache model

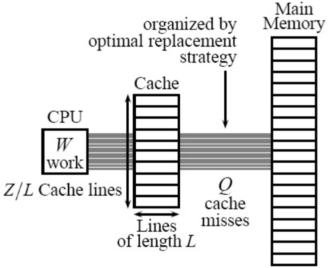


Figure 1: The ideal-cache model

- The ideal cache is **fully associative**: cache lines can be stored anywhere in the cache.
- The ideal cache uses the **optimal off-line strategy of replacing** the cache line whose next access is furthest in the future, and thus it exploits temporal locality perfectly.

Plan

Plan

An illustrative example (1/2)

Input matrix and vector

$$\overset{A}{\begin{pmatrix} a_{0,0} & 0 & 0 & 0 & a_{0,4} & 0 \\ 0 & 0 & a_{1,2} & 0 & 0 & a_{1,5} \\ 0 & a_{2,1} & 0 & a_{2,3} & 0 & 0 \end{pmatrix}} \times \overset{x}{\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}}$$

Cache misses due to x

Assume that the cache has 2 lines each of 2 words. Assume also that the cache is dedicated to store the entries from x:

$$\begin{bmatrix} \emptyset & \emptyset \\ \emptyset & \emptyset \end{bmatrix} \quad \begin{bmatrix} x_0 & x_1 \\ \emptyset & \emptyset \end{bmatrix} \quad \begin{bmatrix} x_0 & x_1 \\ x_4 & x_5 \end{bmatrix} \quad \begin{bmatrix} x_0 & x_1 \\ x_2 & x_3 \end{bmatrix} \quad \begin{bmatrix} x_4 & x_5 \\ x_2 & x_3 \end{bmatrix} \quad \begin{bmatrix} x_4 & x_5 \\ x_0 & x_1 \end{bmatrix}$$

An illustrative example (2/2)

After reordering the columns of A

$$\overset{A'}{\begin{pmatrix} a_{0,0} & a_{0,4} & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{1,2} & a_{1,5} & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{2,1} & a_{2,3} \end{pmatrix}} \times \overset{x'}{\begin{pmatrix} x_0 \\ x_4 \\ x_2 \\ x_5 \\ x_1 \\ x_3 \end{pmatrix}}$$

Cache misses due to x'

Assume that the cache has 2 lines each of 2 words. Assume also that the cache is dedicated to store the entries from x:

$$\begin{bmatrix} \emptyset & \emptyset \\ \emptyset & \emptyset \end{bmatrix} \quad \begin{bmatrix} x_0 & x_4 \\ \emptyset & \emptyset \end{bmatrix} \quad \begin{bmatrix} x_0 & x_4 \\ \emptyset & \emptyset \end{bmatrix} \quad \begin{bmatrix} x_0 & x_4 \\ x_2 & x_5 \end{bmatrix} \quad \begin{bmatrix} x_0 & x_4 \\ x_2 & x_5 \end{bmatrix} \quad \begin{bmatrix} x_0 & x_4 \\ x_1 & x_3 \end{bmatrix}$$

Plan

Gray Codes

Definition

- For $N = 2^n$, an n-bit code $C_n = (u_1, u_2, \ldots, u_N)$, where $N = 2^n$, is a Gray code if u_i and u_{i+1} differ in exactly one bit, for all i.
- This corresponds to a Hamiltonian path (cycle) in the n-dimensional hypercube.

Binary Reflected Gray Codes

The reflected Gray code Γ^n is defined recursively by

$$\Gamma^1 = (0, 1) \ \text{ and } \ \Gamma^{n+1} = \mathbf{0}\,\Gamma^n, \ \mathbf{1}\,\Gamma^{n\mathrm{R}}$$

Introduced by Frank Gray 1953 for shaft encoders

$$\Gamma^3 = 000, 001, 011, 010, 110, 111, 101, 100.$$

Gray Codes

Definition

- For $N = 2^n$, an n-bit code $C_n = (u_1, u_2, \ldots, u_N)$, where $N = 2^n$, is a Gray code if u_i and u_{i+1} differ in exactly one bit, for all i.
- This corresponds to a Hamiltonian path (cycle) in the n-dimensional hypercube.

Binary Reflected Gray Codes

The reflected Gray code Γ^n is defined recursively by

$$\Gamma^1 = (0, 1) \ \text{ and } \ \Gamma^{n+1} = \mathbf{0}\,\Gamma^n, \ \mathbf{1}\,\Gamma^{n\mathrm{R}}$$

Introduced by Frank Gray 1953 for shaft encoders

$$\Gamma^3 = 000, 001, 011, 010, 110, 111, 101, 100.$$

Gray Codes

Binary reflected Gray code for arithmetic operations

- Integers of dimension m can be represented by a data structure that uses $m + \log m + O(\log\log m)$ bits so that increment and decrement operations require at most $\log m + O(\log\log m)$ bit inspections and 6 bit changes per operation. (M.Z. Rahman and J.I. Munro, 2007).
- They have also good results for addition and subtraction.

Binary reflected Gray code for sorting big integers

- A set of n binary strings of dimension m is sparse if any 2 strings are very unlikely to have two consecutive 1's at the same positions.
- BRGC-Sorting a sparse set of n binary strings of dimension m requires
  - $O(\tau)$ index comparisons and $O(\tau + n)$ data-structure updates,
  - where τ is the total number of 1's.

  (Sardar Anisul Haque and M.M.M., 2010).
- Thus, BRGC-sorting a sparse set of n big integers fits within $O(\tau + n)$.

Gray Codes

Binary reflected Gray code for arithmetic operations

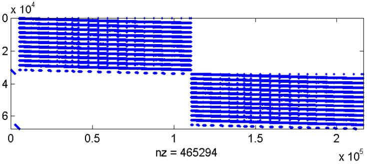
- Integers of dimension m can be represented by a data structure that uses $m + \log m + O(\log\log m)$ bits so that increment and decrement operations require at most $\log m + O(\log\log m)$ bit inspections and 6 bit changes per operation. (M.Z. Rahman and J.I. Munro, 2007).
- They have also good results for addition and subtraction.

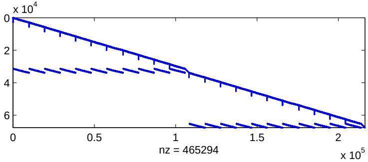
Binary reflected Gray code for sorting big integers

- A set of n binary strings of dimension m is sparse if any 2 strings are very unlikely to have two consecutive 1's at the same positions.
- BRGC-Sorting a sparse set of n binary strings of dimension m requires
  - $O(\tau)$ index comparisons and $O(\tau + n)$ data-structure updates,
  - where τ is the total number of 1's.

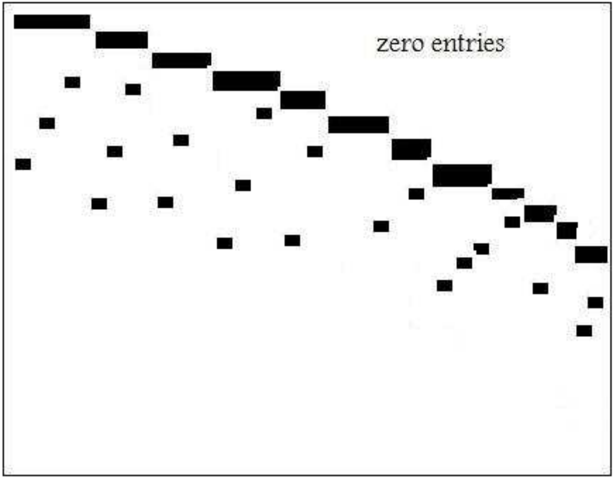
  (Sardar Anisul Haque and M.M.M., 2010).
- Thus, BRGC-sorting a sparse set of n big integers fits within $O(\tau + n)$.

From the University of Florida sparse matrix collection





Non-zero streams



Plan

Cache complexity estimates

Expected cache misses in accessing x when A is **random**

Recall that A is very sparse. Assume that n is large enough such that the vector x does not fit into the cache, typically $n \in O(Z^2)$

$$\overline{Q_1} = Z/L + (\tau - Z/L)\frac{n - Z/L}{n}.$$

Indeed, no spatial locality should be expected in accessing x.

Expected cache misses in accessing x after **BRGC-reordering** A

$$\overline{Q_2} = n/L + Z/L + (n - Z/L)\frac{n/\rho - Z/L}{n/\rho} + (\tau - 2n)\frac{cn/\rho - Z/L}{cn/\rho},$$

where c is the average number of nonzero streams under one step of first level nonzero stream and $1 \leq c \leq \rho$.

Cache complexity estimates

Expected cache misses in accessing x when A is **random**

Recall that A is very sparse. Assume that n is large enough such that the vector x does not fit into the cache, typically $n \in O(Z^2)$

$$\overline{Q_1} = Z/L + (\tau - Z/L)\frac{n - Z/L}{n}.$$

Indeed, no spatial locality should be expected in accessing x.

Expected cache misses in accessing x after **BRGC-reordering** A

$$\overline{Q_2} = n/L + Z/L + (n - Z/L)\frac{n/\rho - Z/L}{n/\rho} + (\tau - 2n)\frac{cn/\rho - Z/L}{cn/\rho},$$

where c is the average number of nonzero streams under one step of first level nonzero stream and $1 \le c \le \rho$.

How much do we save?

For our large test matrices and today's L2 cache sizes, the following conditions hold:

- $n \in O(Z^2)$ and
- $Z > 2^{10}$.

Using MAPLE, we could prove the following relation:

$$\overline{\mathbf{Q_1}} - \overline{\mathbf{Q_2}} \approx \mathbf{n}.$$

Plan

Experimentation

Matrix name	m	n	τ	SPMxV with BRGC ordering	SPMxV without any ordering
fome21	67748	216350	465294	3.6	3.9
lp_ken_18	105127	154699	358171	2.7	3.1
barrier2-10	115625	115625	3897557	19.0	19.1
rajat23	110355	110355	556938	3.0	3.0
hcircuit	105676	105676	513072	2.6	2.5
GL7d24	21074	105054	593892	3.0	3.2
matrix_9	103430	103430	2121550	8.4	8.0
GL7d17	1548650	955128	25978098	484.6	625.0
GL7d19	1911130	1955309	37322725	784.6	799.0
wiki-20051105	1634989	1634989	19753078	258.9	321.0
wiki-20070206	3566907	3566907	45030389	731.5	859.0

Summary

1. Reordering columns or rows in SPMxV can improve locality significantly.
2. Optimal reordering is computationally hard.
3. Preprocessing cost needs to be amortized against the SPMXVs in the conjugate gradient type algorithms.
4. BRGC ordering improves locality.
5. BRGC ordering technique can be implemented in linear time w.r.t. τ.
6. The cost of BRGC ordering can be amortized before $\sqrt{n}$ SPMXVs in conjugate Gradient type algorithms.
7. The matrices used in our experimentation are very sparse but already have nice structures, so they are far from the (ideal) random case.