Brief Announcement: The Cache-Oblivious Gaussian Elimination Paradigm — Theoretical Framework and Experimental Evaluation*

Rezaul Alam Chowdhury & Vijaya Ramachandran
Department of Computer Sciences, UT Austin, Austin, TX 78712
shaikat@cs.utexas.edu, vlr@cs.utexas.edu

Categories & Subject Descriptors: B.3.2 [Memory Structures]: Design Styles – *Cache memories*; F.2.1 [Analysis of Algorithms and Problem Complexity]: Numerical Algorithms and Problems – *Computations on matrices*

General Terms: Algorithms, Theory, Experimentation.

Cache-efficient algorithms improve execution time by exploiting data parallelism inherent in the transfer of blocks of useful data between adjacent memory levels. By increasing locality in their memory access patterns, these algorithms try to keep the number of block transfers small. The cache-oblivious model [1] is a further refinement that enables the development of system-independent cache-efficient algorithms that simultaneously adapt to all levels of a multi-level memory hierarchy. This leads to fuller use of data parallelism and also produces portable code. The cache-oblivious model represents the memory hierarchy with two memory levels — an ideal cache of size M and an unlimited main memory partitioned into blocks of size B. The *cache complexity* of an algorithm is the number of *I/Os* (i.e., block transfers) performed between these two levels.

In [2] we introduced the *Gaussian Elimination Paradigm* (*GEP*) — a cache-oblivious framework for several important problems solvable using a construct similar to the computation in Gaussian elimination without pivoting (see Fig. 1).

The traditional GEP code in Fig. 1 runs in $\mathcal{O}\left(n^3\right)$ time and incurs $\mathcal{O}\left(\frac{n^3}{B}\right)$ I/Os. In [2] we presented a framework (which we call here I-GEP) for an in-place $\mathcal{O}\left(n^3\right)$ time and $\mathcal{O}\left(\frac{n^3}{B\sqrt{M}}\right)$ I/O cache-oblivious execution of Gaussian elimination without pivoting, all-pairs shortest paths, and other important special cases of GEP. However, there exist instances of f and Σ_G for which I-GEP does not solve GEP correctly (e.g., see full version of this write-up [3]).

In this work we establish several important properties of I-GEP and build on these results to derive C-GEP, which is a provably correct and optimal cache-oblivious implementation of GEP for all f and Σ_G. C-GEP has the same time and cache complexity as I-GEP, and uses $n^2 + n$ extra space (where the size of the input matrix c is n^2).

Both I-GEP and C-GEP have potential application in optimizing compilers as cache-oblivious 'tiling' techniques.

G(c, n, f, Σ_G)

(Input $c[1 \ldots n, 1 \ldots n]$ is an $n \times n$ matrix, $f(\cdot,\cdot,\cdot,\cdot)$ is an arbitrary problem-specific function, and Σ_G is a problem-specific set of triplets such that $c[i,j] \leftarrow f(c[i,j], c[i,k], c[k,j], c[k,k])$ is executed in line 5 if $\langle i,j,k\rangle \in \Sigma_G$.)

1. ***for*** $k \leftarrow 1$ ***to*** n ***do***
2.   ***for*** $i \leftarrow 1$ ***to*** n ***do***
3.   ***for*** $j \leftarrow 1$ ***to*** n ***do***
4.   ***if*** $\langle i,j,k\rangle \in \Sigma_G$ ***then***
5. $c[i,j] \leftarrow f(c[i,j], c[i,k], c[k,j], c[k,k])$

Figure 1: Traditional GEP code.

We present an empirical comparison of I-GEP and C-GEP with the traditional GEP in Fig. 1 for both *in-core* and *out-of-core* computations. We use $f(x,u,v,w) = \min(x, u+v)$ and $\Sigma_G = \{\ \langle i,j,k\rangle \mid i,j,k \in [1,n]\ \}$ as in Floyd-Warshall's all-pairs shortest paths algorithm.

We ran our in-core experiments on Intel Xeon and SUN UltraSPARC-III+. On both architectures the relative performance of I-GEP and C-GEP with respect to GEP improved as n increased, and both of them ran faster than GEP when $n \geq 2^{11}$. On the SUN machine both I-GEP and C-GEP *ran 1.7 times faster* and *incurred a factor of 100 times fewer L2 cache misses* than GEP when $n = 2^{13}$.

We ran our out-of-core experiments on an Intel Xeon machine equipped with a fast (10K RPM and 4.5 ms avg seek time) hard disk. For disk accesses we used STXXL — an implementation of the C++ standard template library STL for external memory computations. STXXL maintains a fully associative cache in RAM with pages from the disk. In our experiments, the I/O wait time for I-GEP/C-GEP was *less than that for GEP by at least a factor of 180* when executed on an input matrix only half of which fit in internal memory.

All experimental data along with a more detailed description of experiments are given in [3], together with detailed results for properties of I-GEP and C-GEP.

REFERENCES

[1] M. Frigo, C. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. *Proc. FOCS*, 1999.
[2] R. Chowdhury and V. Ramachandran. Cache-oblivious dynamic programming. *Proc. SODA*, 2006.
[3] R. Chowdhury and V. Ramachandran. The cache-oblivious Gaussian elimination paradigm: theoretical framework and experimental evaluation. TR-06-04, CS Dept., UT Austin, 2006.

*This work was supported in part by NSF Grant CCF-0514876 and NSF CISE Research Infrastructure Grant EIA-0303609.

SPAA'06, July 30–August 2, 2006, Cambridge, Massachusetts, USA.
ACM 1-59593-262-3/06/0007.