

A MATLAB Interface to the GPU

Second Winter School
Geilo, Norway

André Rigland Brodtkorb
<Andre.Brodtkorb@sintef.no>

SINTEF ICT
Department of Applied Mathematics

2007-01-24

Outline

- 1 Motivation and previous work.
- 2 MATLAB interface.
 - Gauss-Jordan.
 - PLU factorization.
 - Tridiagonal Gaussian elimination.
- 3 Summary.

Why use the GPU from MATLAB?

- High-level, with mathematical syntax: $[U S V] = \text{svd}(A)$.
- A standard tool for scientists and engineers for prototyping.
- Extendible with user-defined MEX files.
- **Using the GPU can speed up computations.**

Previous work - GPU

- Matrix multiplication:
 - Fixed function (Larsen & McAllister, 2001),
 - Packing (Moravánszky, 2003; Hall et al., 2003),
 - Analysis (Fatahalian et al., 2004),
 - Automatic tuning (Jiang & Snir, 2005),
 - Analysis (Govindaraju et al., 2006).
- G-J/PLU factorization: Single component (Galoppo et al., 2005).
- Conjugate gradients: Sparse matrix-vector product (Bolz et al., 2003; Krüger & Westermann, 2003).
- CUBLAS.
- RapidMind, ~~PeakStream~~.

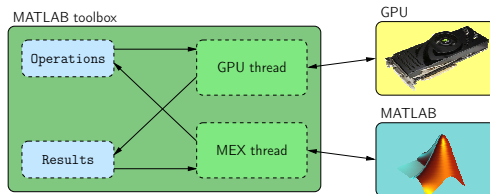
Previous work - MATLAB

- NVIDIA PhD fellow: CUFFT + MATLAB.
- NASA: GPU backend for MATLAB:
 - “**Virtually all software** requiring extensive numerical processing **could benefit** from the solvers developed in this project”,
 - “Most such software is already created using BLAS or similar libraries, thus **requiring very little modification** to be used with these accelerated solvers.”
- Parallel MATLAB survey: 8 message passing, 7 embarrassingly parallel, 6 backends, 7 compilers

What do we want?

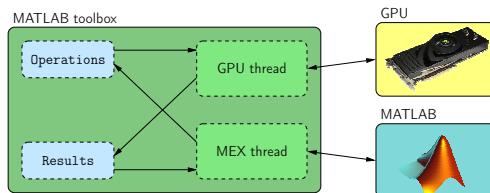
- Use the GPU as a coprocessor.
- An easy-to-use interface (tight integration with existing MATLAB syntax).
- BUT: (Almost) no thread-safe GPU APIs, and no thread-safe MEX API (MATLAB).

Two threads of execution



- A queue of operations, and a map of results.
- Similar to RapidMind and PeakStream ideas.
- Not a backend (as NASA's project), but a new class.

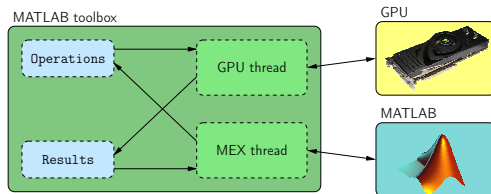
Two threads of execution



- 1 MATLAB user creates a gpuMatrix object.
- 2 Operation (+, -, rref, plu, etc) on the object.
- 3 MEX file is called using operator overloading.
- 4 The MATLAB thread enqueues the operation.

...

Two threads of execution



...

- 5** The GPU thread computes the results, and moves to result map.
- 6** User requests result (e.g., `single(var)`).
- 7** Data is transparently moved from the GPU to MATLAB memory.

Syntax

Standard MATLAB

```
a = rand(n, n);  
b = a*a;  
[l u p] = lu(b);
```

GPU toolbox

```
a = gpuMatrix(rand(n, n));  
b = a*a;  
[l u p] = lu(b);
```

Background processing

```
a = gpuMatrix(rand(n, n));  
b = a*a;  
c = lu(b);  
read(c);
```

```
%CPU computations here
```

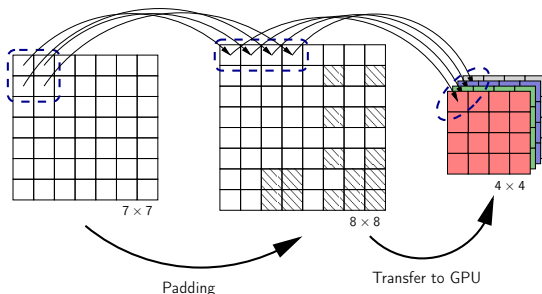
```
[l u p] = single(c);
```

Summary interface

- Simple interface with familiar MATLAB syntax.
- Background processing by splitting into two threads.
- Uses MATLAB objects, not a backend -> easier to reuse data?
- MATLAB has internal memory management.

Algorithms

- Four algorithms: (matrix multiplication), Gauss-Jordan, PLU factorization, tridiagonal Gaussian elimination (GE).
- All implemented using operator overloading to call the MEX file.
- All but tridiagonal GE are background (i.e. non-blocking) operators.
- Use of SIMD-vectorization (RGBA -> two-by-two packing)



Gauss-Jordan factorization (i)

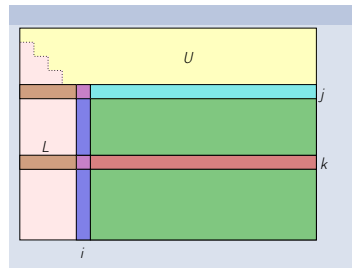
- Direct solver
- Slower than Gaussian elimination, but fewer passes needed.
- Need to employ a pivoting strategy for numerical stability.
 - Full - Overkill for most problems and not applicable for the chosen implementation (Doolittle)
 - Rook - Not applicable for the chosen implementation (Doolittle)
 - Partial - Works well for most cases
- Need to pivot two-by-two sub-matrices

PLU factorization (i)

- Direct solver.
- Can use a modification to the Doolittle algorithm.
- Can use same pivoting as for Gauss-Jordan elimination.
- Suitable for many right hand sides (Factorization $\mathcal{O}(n^3)$, while substitution is $\mathcal{O}(n^2)$).

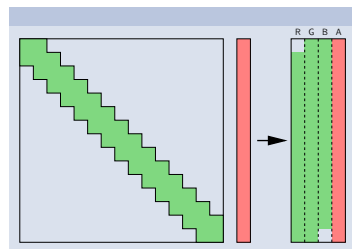
PLU factorization (ii)

- 1 Locate pivoting element
- 2 Exchange two-by-two rows, and calculate multipliers
- 3 Reduce below pivot element



Tridiagonal Gaussian elimination

- Tridiagonal storage - RGBA
- Can solve many systems in parallel
- Poor performance when solving only one system
- Stack many systems in columns.



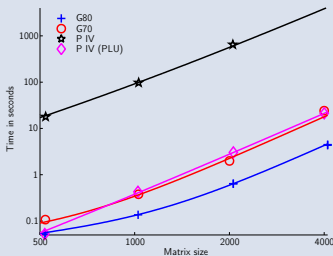
Summary operations

- Inherently parallel algorithms, but a lot of memory access per FLOP.
- Multi-pass algorithms are potentially expensive (reduction).
- Results will differ from CPU code (single precision, denormals).

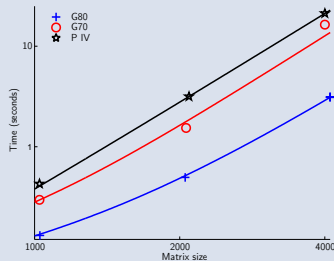
Accuracy

- Accurate storage and computation yields accurate results (e.g. most integral matrices).
- “A function of the condition number” (often matrix size).
- Experienced error is comparable to CPU single precision.

Runtime of different algorithms (loglog)



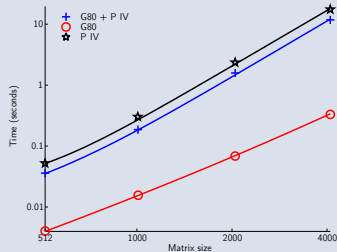
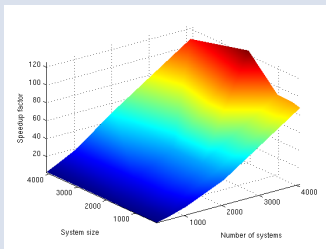
(a) Gauss-Jordan elimination (640×/4.3×)



(b) PLU factorization (7.55×)

- Fit $a + bx + cx^2 + dx^3$ to the runtimes.
- Compare the coefficient d , giving a speedupfactor valid for large n .
- Gauss-Jordan benchmarked against both rref (640×), and PLU (4.3×).

Runtime of different algorithms (loglog)



(c) Tridiagonal Gaussian elimination (a lot (d) Background computation PLU (1.7 \times))
(120 \times))

- Tridiagonal GE does not scale well wrt system size. Scales very well wrt number of systems solved.
- Background computation is free when enough CPU computation is computed simultaneously.

Summary

- Programmed using OpenGL -> difficult. CUDA remedies this.
- The GPU can be utilized as an efficient coprocessor, but MATLAB memory management “stinks”.
- As NASA says: A high-level mathematical interface to efficient algorithms is useful for prototyping

Contributions

- A high-level mathematical interface to the GPU.
- A new pivoting strategy for vectorized operations.
- The use of packing for Gauss-Jordan and PLU factorization.

- Paper accepted, soon to come:)
- Masters thesis available: <http://babrodtk.at.ifi.uio.no>

-fin-

Bolz, J., Farmer, I., Grinspun, E., & Schröder, P. 2003.

Sparse matrix solvers on the GPU: conjugate gradients and multigrid.

[ACM Trans. Graph.](#), 22(3), 917–924.

Fatahalian, K., Sugerman, J., & Hanrahan, P. 2004.

Understanding the efficiency of GPU algorithms for matrix-matrix multiplication.

Pages 133–137 of: [HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware.](#)

New York, NY, USA: ACM Press.

Galoppo, N., Govindaraju, N. K., Henson, M., & Manocha, D. 2005.

LU-GPU: Efficient Algorithms for Solving Dense Linear Systems on Graphics Hardware.

Page 3 of: [SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing.](#)
Washington, DC, USA: IEEE Computer Society.

Govindaraju, N. K., Larsen, S., Gray, J., & Manocha, D. 2006.

A memory model for scientific algorithms on graphics processors.

Page 89 of: [SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing.](#)
New York, NY, USA: ACM Press.

Hall, J. D., Carr, N. A., & Hart, J. C. 2003.

[Cache and bandwidth aware matrix multiplication on the GPU.](#)

Jiang, C., & Snir, M. 2005.

Automatic Tuning Matrix Multiplication Performance on Graphics Hardware.

Pages 185–196 of: [PACT '05: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques.](#)

Washington, DC, USA: IEEE Computer Society.

Krüger, J., & Westermann, R. 2003.

Linear algebra operators for GPU implementation of numerical algorithms.
ACM Trans. Graph., 22(3), 908–916.

Larsen, E. S., & McAllister, D. 2001.

Fast matrix multiplies using graphics hardware.

Pages 55–55 of: Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM).

New York, NY, USA: ACM Press.

Moravánszky, A. 2003.

Dense Matrix Algebra on the GPU.

Online; <http://www.shaderx2.com/shaderx.pdf>.

[accessed 2006-05-11].