A MATLAB Interface to the GPU

André Rigland Brodtkorb

Department of Informatics
Faculty of Mathematics and Natural Sciences
University of Oslo

June 2007
What is the GPU?

- GPU - Graphics Processing Unit.
- Transforms input data in form of geometry into pixels on screen.
- Highly efficient processor for computing with homogeneous 3D coordinates \([x y z w]\) and the RGBA color model.
- Modeled as a pipeline, where all output elements have traversed all stages.
The Graphics Pipeline

What is the GPU?

- **Vertex Stage**
- **Primitive Assembly**
- **Rasterization**
- **Fragment Stage**
- **Buffer Operations**

---

A MATLAB Interface to the GPU

André Rigland Brodtkorb
### Why use the GPU?

#### Comparison of features

<table>
<thead>
<tr>
<th></th>
<th>CPU 1</th>
<th>GPU 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theoretical GFLOPS</td>
<td>90</td>
<td>570</td>
</tr>
<tr>
<td>Theoretical bandwidth (GB/s)rti</td>
<td>6.4</td>
<td>100</td>
</tr>
<tr>
<td>Folding@home GFLOPS</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>Price per GFLOPS (NOK)</td>
<td>90</td>
<td>10</td>
</tr>
<tr>
<td>Watts per GFLOPS</td>
<td>1.4</td>
<td>0.3</td>
</tr>
</tbody>
</table>

1 Intel Core 2 Extreme QX6700  
NVIDIA GeForce 8800 Ultra  

A MATLAB Interface to the GPU  
André Rigland Brodtkorb
Why use MATLAB?

- High-level, with mathematical syntax: \([U S V] = \text{svd}(A)\).
- A standard tool for scientists and engineers used all over the world (over one million users of MATLAB and SIMULINK worldwide).
- Extendible with user-defined MEX files.
## Matrix multiplication timeline

<table>
<thead>
<tr>
<th>Year</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>Fixed function implementation (Larsen &amp; McAllister, 2001).</td>
</tr>
<tr>
<td>2003</td>
<td>Packing (Moravánszky, 2003; Hall et al., 2003) and blocking (Hall et al., 2003) introduced.</td>
</tr>
<tr>
<td>2004</td>
<td>Analysis of previous algorithms. New algorithm faster than ATLAS (Fatahalian et al., 2004).</td>
</tr>
<tr>
<td>2005</td>
<td>Automatically tuning to underlying hardware (Jiang &amp; Snir, 2005).</td>
</tr>
<tr>
<td>2006</td>
<td>Analysis of bandwidth and blocking techniques (Govindaraju et al., 2006).</td>
</tr>
</tbody>
</table>
PLU Factorization

- Single-component textures
- Intricate algorithm for pivoting requiring many passes
- 35% speedup over CPU claimed for partial pivoting.
- GPU claimed to be an order of magnitude faster than the CPU for full pivoting.
- Highly synthetic benchmarks, where all texture reads were restricted to three locations in memory.
Background processing

Overview

- Want to utilize the GPU as a *coprocessor*.
- Need an easy-to-use interface (tight integration with existing MATLAB syntax).
- Want to execute efficiently on the GPU.
Background processing

- Want to utilize the GPU as an *extra* resource.
- Blocking and non-blocking calls – Can utilize threads.
- Neither OpenGL, nor MATLAB are thread-safe.
- Need to split logic into two parts – MEX and OpenGL.
Splitting of logic into two threads

- A queue of operations, and a map of results.
- Similar to RapidMind and PeakStream ideas.
Simultaneous computation

Using non-blocking calls, we can utilize both the CPU and the GPU simultaneously:

1. Enqueue GPU operations.
2. Compute on the CPU while the GPU operates in the background.
3. Retrieve results from GPU.

This makes computation on the GPU virtually free.
A MATLAB Interface to the GPU

Background processing

Syntax

Standard MATLAB

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

GPU toolbox in MATLAB

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

Background processing

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

%CPU computations here

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

- Four-way vectorized arithmetic.
- Packing influences computational and memory intensity.
- Want to reuse data without having to repack data.
- Two-by-two packing is a good compromise.
- Possible to extend the toolbox to support other packing algorithms.
A MATLAB Interface to the GPU

Packing

Transfer to GPU

Padding

7×7

8×8

4×4
Matrix-matrix multiplication

Definition

\[(AB)_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}, \quad A \in \mathbb{R}^{m,n}, \quad B \in \mathbb{R}^{n,o}. \quad (1)\]

- Can be viewed as vector-vector inner products.
- Can be viewed as the sum of individual multiplications.
Vector-vector inner product

1. Each fragment contains the texture-coordinate to the corresponding row in $A$, and column in $B$

2. Each fragment is computed by gathering one two-by-two matrix from $A$, and one from $B$, computing their inner product, and summing over all elements.
Sum of individual multiplications

1. Each fragment contains the texture-coordinate to the corresponding element from \( A \) and \( B \).

2. Each fragment is computed by gathering the two two-by-two matrices, computing their inner product, and adding to one accumulation buffer.
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
Gauss-Jordan factorization (ii)

Algorithm

1. Find the pivoting element by reducing the pivot area to the largest element. Use quasi-harmonic norm as a measure of suitedness for each element.
2. Exchange two-by-two rows
3. Eliminate two-by-two column above and below pivot element
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 $O(n^3)$, while substitution is $O(n^2)$).
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
Accuracy and stability

Test problems

Matrices

- Matrix multiplication - uniform random in $[0, 10]$ (condition number $10^2$ to $5 \times 10^6$)
- Gauss-Jordan, PLU - uniform random in $[0, 5]$ with a random integer in $[0, 100]$ added to the diagonal (condition number $10^2$ to $4 \times 10^4$).
- Tridiagonal Gaussian - uniform random in $[0, 5]$ with a random integer in $[0, 100]$ added to the diagonal (condition number $2 \times 10^1$ to $5 \times 10^3$)
Accuracy and stability

Measured error

(a) Matrix multiplication

(b) Gauss-Jordan elimination
Accuracy and stability

Measured error

(c) PLU factorization

(d) Tridiagonal Gaussian elimination
Accuracy and stability

Accuracy

- Accurate storage and computation yields accurate results (e.g. most integral matrices)
- Absolute error depends on \( n \) and the size of input/output elements.
- Relative error depends only on \( n \).
Runtime of different algorithms

(a) Matrix multiplication

(b) Gauss-Jordan elimination
Runtime of different algorithms

(c) PLU factorization

(d) Tridiagonal Gaussian elimination
Background processing

The diagram shows the speed comparison between GPU and CPU for different algorithms. The x-axis represents the input size (x), while the y-axis represents the processing time. The lines and markers indicate the performance of:
- GPU + CPU
- CPU
- GPU
- Lsqr GPU + CPU
- Lsqr CPU
- Lsqr GPU

The data suggests that the GPU + CPU combination is the most efficient, followed by the CPU and then the GPU alone. The Lsqr algorithms show a linear increase with input size, with the GPU + CPU method being the fastest.
Conclusions

- The GPU is hard to program, but there are great rewards (speedups of 2-7 shown here).
- The GPU can be utilized as an efficient coprocessor.
- A high-level mathematical interface to efficient algorithms is useful.
Conclusions, and further research

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.
Conclusions, and further research

Understanding the efficiency of GPU algorithms for matrix-matrix multiplication.
New York, NY, USA: ACM Press.

A memory model for scientific algorithms on graphics processors.
New York, NY, USA: ACM Press.

Cache and bandwidth aware matrix multiplication on the GPU.

Automatic Tuning Matrix Multiplication Performance on Graphics Hardware.
Washington, DC, USA: IEEE Computer Society.

Fast matrix multiplies using graphics hardware.
New York, NY, USA: ACM Press.