High-level GPU computing with Jacket for MATLAB and C/C++

Gallagher Pryor, Brett Lucey, Sandeep Maddipatla, Chris McClanahan, John Melonakos, Vishwanath Venugopalakrishnan, Krunal Patel, Pavan Yalamanchili, James Malcolm

AccelerEyes, 75 5th Street NW STE 204, Atlanta, GA 30308

ABSTRACT

We describe a software platform for the rapid development of general purpose GPU (GPGPU) computing applications within the `MATLAB` computing environment, C, and C++: `Jacket`. `Jacket` provides thousands of GPU-tuned function syntaxes within `MATLAB`, C, and C++, including linear algebra, convolutions, reductions, and FFTs as well as signal, image, statistics, and graphics libraries. Additionally, `Jacket` includes a compiler that translates MATLAB and C++ code to CUDA PTX assembly and OpenGL shaders on demand at runtime. A facility is also included to compile a domain specific version of the `MATLAB` language to CUDA assembly at build time. `Jacket` includes the first parallel GPU FOR-loop construction and the first profiler for comparative analysis of CPU and GPU execution times. `Jacket` provides full GPU compute capability on CUDA hardware and limited, image processing focused compute on OpenGL/ES (2.0 and up) devices for mobile and embedded applications.

Keywords: CUDA, GPU, GPGPU, MATLAB, C, C++, Jacket

1. INTRODUCTION

`Jacket` is a software platform developed at AccelerEyes,[1] that allows users and programmers to rapidly develop data-parallel programs in MATLAB (by Mathworks[2]), C, and C++, on `CUDA-capable` GPUs for HPC applications and OpenGL/ES (2.0 and up) capable GPUs for mobile and embedded applications. `Jacket` provides a simple, high-level matrix abstraction over low-level GPU APIs such as CUDA and OpenGL along with thousands of GPU-tuned functions to allow users in science, engineering, and finance to take full advantage of GPU hardware.[3] The combination of a simple matrix language, automatic memory management, on-the-fly compilation, parallel GPU FOR-loop construction, hybrid CPU/GPU profiler, interactive hardware-accelerated graphics library, and `MATLAB` language compiler make `Jacket` well suited to rapid prototyping of data-parallel algorithms and building end-to-end applications.

Over the last decade GPUs have proliferated both consumer and developer computers. Despite the growing list of success stories, GPU software development adoption has had a slow rise. The slowness of the rise is attributable to the difficulty in programming GPUs.

`Cg`, `GLSL`, `HLSL`, and `Brook` marked the beginning of stream programming, a precursor to general purpose GPU programming, where computation is mapped onto the graphics pipeline and consequently subject to various constraints. Following on the heels of these technologies, `CUDA` and `OpenCL` introduced a more generally programmable software architecture, easier than stream programming but harder to program than standard single-threaded C/C++. However, even with these advances there remains a steep learning curve for the average programmer to reach success with CUDA, OpenCL, etc.

`Jacket` is attempts to bridge this gap by mapping high-level languages (i.e. easier to program than standard C/C++) onto the underlying hardware. The goal is to deliver progammability without sacrificing performance.

Several previous companies also made attempts to reach this goal. One of the first such companies, PeakStream (now at Google) built a C/C++ runtime and library of functions providing a rich toolset to GPU development. RapidMind (now at Intel) built a flexible middle-layer supporting various frontend languages and backend hardware targets. Both of these sought to bridge the gap between the hardware and developers. One of the newest platforms for GPU development, `Jacket`, allows programmers to use the high-level M-language and to abstract away the low-level details of GPU programming. The result is increased productivity and performance.

Modeling and Simulation for Defense Systems and Applications VI, edited by Eric J. Kelmelis, Proc. of SPIE Vol. 8060, 806005 · © 2011 SPIE · CCC code: 0277-786X/11/$18 · doi: 10.1117/12.884899

2. JACKET

2.1 API: MATLAB and C++

`Jacket` revolves around a single matrix class, the `garray`, and its typed subclasses,

`C++`	`MATLAB`	Description
`f64`	`gdouble`	Single-precision (float) array on GPU
`f32`	`gsingle`	Double-precision (float) array on GPU
`c64`	`gdouble`	Complex double-precision array (cuComplex) on GPU
`c32`	`gsingle`	Complex single-precision array (cuComplex) on GPU
`b8`	`gboolean`	Boolean array (8-bit bool) on GPU

Jacket `garrays` are multidimensional, can be generated via simple matrix creation functions such as (`ones`, `rand`, etc), and can be manipulated with arithmetic and functions much like standard scalar CPU-side counterparts.

Additionally, Jacket provides a parallel FOR-loop implementation, `gfor`, which executes arbitrarily many instances of *independent* routines in a data-parallel fashion.

2.2 A Simple Example: Stereo Disparity Computation

```
f32 left = get_img(), right = get_img(), sad;
gfor (i = 0; i < nshifts; ++i) {
  f32 shifted = circshift(right, 0, i);
  f32 diff = abs(left  shifted);
  sad(span,span,i) = conv2(diff, f32::ones(5,5));
}
f32 disparity = min(sad, 3);
```

Figure 1. Using Jacket to compute the horizontal disparity between two images using plane sweeping and a 5-by-5 correlation window. All disparity computations and the final minimum are executed in parallel while the source code is kept clean and simple without diving into low-level GPU APIs.

We demonstrate that the `Jacket` API (both `MATLAB` and `C++`) in the simple example depicted in Figure 1 showing computation of stereo disparity between two images. This code runs on all GPUs that Jacket supports and remains concise without the need for low-level GPU programming APIs such as CUDA or OpenCL.

We describe the program line by line. The declaration of `left`, `right`, and `sad` indicate that we will be working with 4-byte floating point arrays and that `left` and `right` are imagery that we have acquired. We assume that all arrays are the same size.

The second line indicates via the `gfor` keyword that we will be computing the following block `nshifts + 1` times from `0` to `nshifts` in parallel. All subsequent commands are still executed only once, but instead of describing one computation, each command invokes `nshifts + 1` computations. Thus, the next command shifts the right image `0` pixels, `1` pixel, `2` pixels, etc. and stores the results (if compiled and executed immediately) as a handle in the variable `shifted`. The remaining lines compute the sum of the absolute difference between the two images (multiple times as is the case with the first line) in a sliding 5-by-5 window. Finally, the minimum over all computed sums of absolute differences (SADs) is taken. Note that during execution `Jacket` may be compiling commands together lazily and all the previously discussed operations may be merged into one batch of execution of the GPU and dispatched.

2.3 Compiling M-Code: gcompile

Oftentimes, a more fine-grained description than vectorized code is required to properly describe a procedure. Therefore, Jacket includes a facility, `gcompile`, which allows the run and build-time compilation of a domain specific version of the MATLAB language to GPU code. `gcompile` allows the use of control flow directives (`if`, `while`, `for`, `break`, etc) in order to express more fine grained algorithms than vectorized code allows.

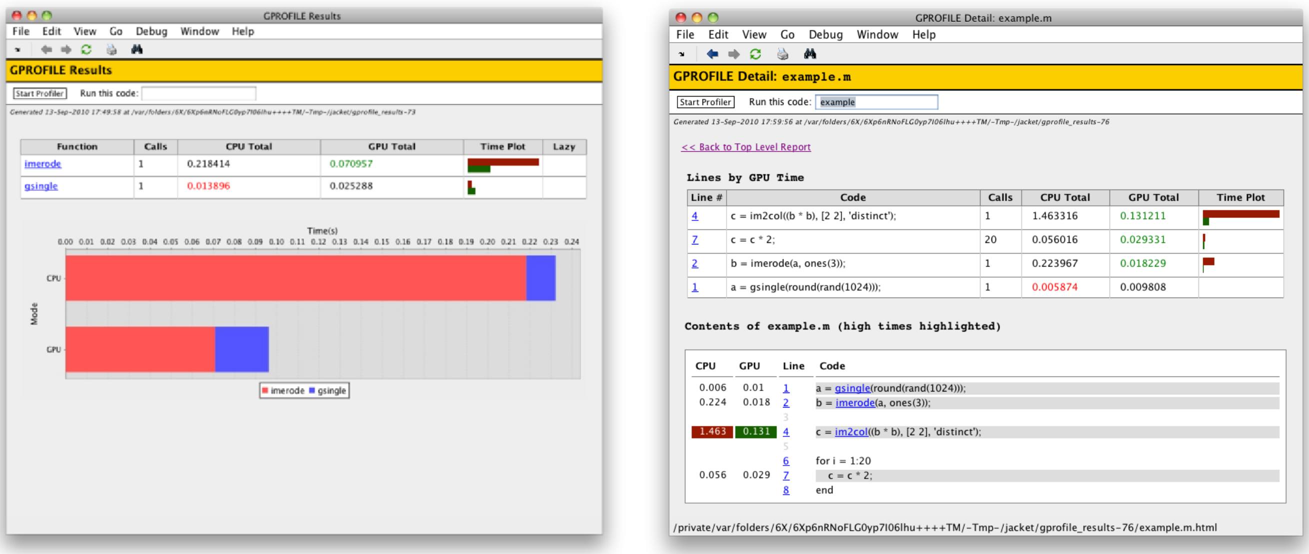
We demonstrate the ubiquitous map/reduce operation utilizing `gcompile` in Figure 2. The code in the left hand side pane begins by defining a string containing the code to be compiled via the `verbatim` command ??. The following code block implements a capped norm of the vectors `in` at a supplied `threshold`. On the first line, `gcompile` compiles the code block and returns a handle that may be utilized as a normal function to run the code described in the string. Finally, the last line utilizes the compiled function to perform the reduction (capped norm) on the given vectors.

```
reduce = gcompile(verbatim);
%{
function single_result = map(in)
  err = norm(in);
  if (err < threshold) result = err;
  else                 result = 0;
end
%}

all_results = reduce(map, vectors);
```

Figure 2. `gcompile` allows the specification of routines that require fine grained control flow that vectorized programming alone may not be able to express. `gcompile` allows the run and build time compilation of a domain specific version of the MATLAB language to GPU code. `gcompile` allows the use of control flow directives (`if`, `while`, `for`, `break`, etc)

2.4 Optimizing Hybrid GPU/CPU Programs: gprofile



Generated 13-Sep-2010 17:49:58 at /var/folders/6X/6Xp6nRNoFLG0yp7I06lhu++++TM/-Tmp-/jacket/gprofile_results-73

Function	Calls	CPU Total	GPU Total	Time Plot	Lazy
imerode	1	0.218414	0.070957		
gsingle	1	0.013896	0.025288		

Generated 13-Sep-2010 17:59:56 at /var/folders/6X/6Xp6nRNoFLG0yp7I06lhu++++TM/-Tmp-/jacket/gprofile_results-76

Lines by GPU Time

Line #	Code	Calls	CPU Total	GPU Total	Time Plot
4	c = im2col((b * b), [2 2], 'distinct');	1	1.463316	0.131211	
7	c = c * 2;	20	0.056016	0.029331	
2	b = imerode(a, ones(3));	1	0.223967	0.018229	
1	a = gsingle(round(rand(1024)));	1	0.005874	0.009808	

Contents of example.m (high times highlighted)

```
CPU     GPU     Line  Code
0.006   0.01    1     a = gsingle(round(rand(1024)));
0.224   0.018   2     b = imerode(a, ones(3));
                3
1.463   0.131   4     c = im2col((b * b), [2 2], 'distinct');
                5
                6     for i = 1:20
0.056   0.029   7       c = c * 2;
                8     end
```

/private/var/folders/6X/6Xp6nRNoFLG0yp7I06lhu++++TM/-Tmp-/jacket/gprofile_results-76/example.m.html

Figure 3. Example report from the `gprofview` facility in `Jacket`. `gprofview` allows execution and timing of code on both the CPU and GPU and the exploration of the timing data overlaid on a source code browser. The left hand side depicts the overview profiling page for a piece of source code and the right hand illustrates a detailed breakdown view showing CPU/GPU timings on a line by line basis.

Focussing solely on either GPU or CPU performance in a program means that only a portion of the available resources in a computer are being utilized to their fullest extent. It is well known that whereas GPUs are particularly well suited for data-parallel tasks, CPUs are better suited for task-parallel operations with complex memory access patterns and control flow.

To achieve maximum performance from a program, `Jacket` provides a hybrid CPU/GPU profiling tool that reports on CPU and GPU timings for all function calls within a program and provides a report detailing a comparison of the two (See Figure 3). As shown, the code run in the figure performs more than 2 times better than the CPU overall (note Amdahl's law) whereas key components such as imcol and imerode are over two times faster. The benchmarks were completed by comparing a recent Intel i5 against a modest GeForce 9600 GPU utilizing `Jacket 1.5`.

By changing input data sizes and timing CPU/GPU commands close to one another on independent data sets, one may utilize `gprofile` to tune for hybrid compute performance leveraging an entire system rather than just one piece.

3. BENCHMARKS

First and foremost, `Jacket` provides a development environment that is not only easy to use, but fast. In what follows, we present benchmarking data of `Jacket` against the well known performance libraries IPP, MKL, and Eigen, with a final emphasis on 1D, 2D, and 3D convolution.

Figure 4 illustrates timings between a GeForce 480 GPU and a Intel i7 950 CPU with 4 cores (hyperthreading enabled) for an assortment of various heavyweight operations. As shown, Jacket is capable of accelerating these commands to a maximum of 12x against IPP (diff command). Figures 5, 6, and 7 illustrate benchmarks of Jacket versus IPP on several different GPUs and CPUs (see legends) for convolution operations with varying kernel and signal sizes. The maximum speedups obtained with an Intel i7 950 against a GeForce 480 were 50x (largest case of conv), 150x (largest case of conv2), and 200x (largest case of convn). All operations were performed in double precision.

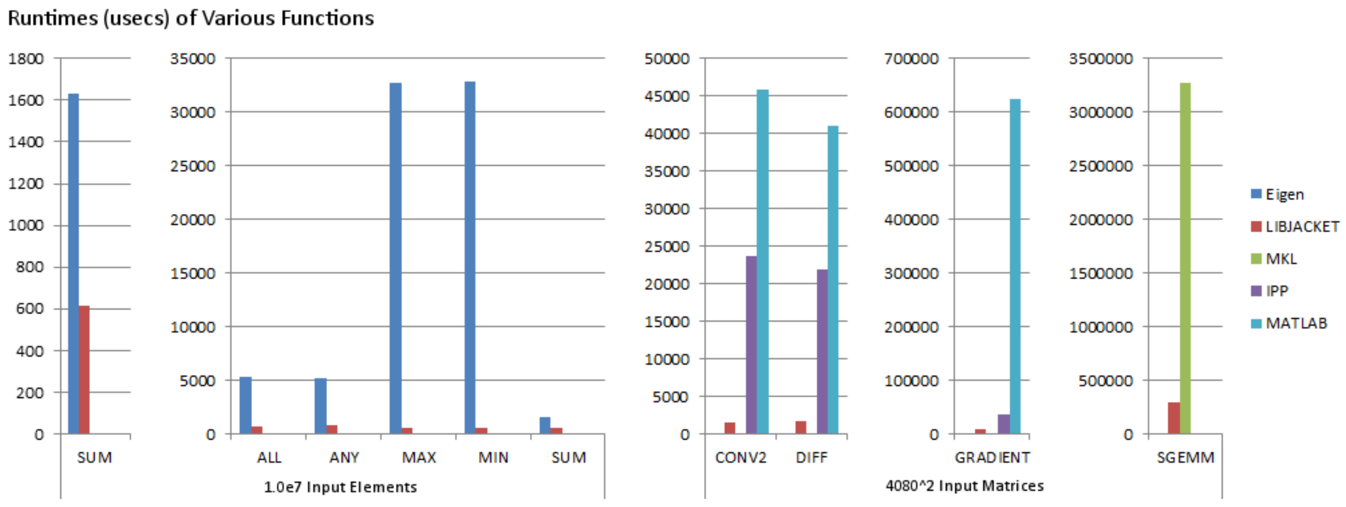


Figure 4. Various benchmarks.

4. CONCLUSION

In this paper, we have presented `Jacket`, a high-level platform for the rapid development of fast algorithms and applications on the GPU. We presented `Jacket's` simple API, its compiler, and profiler. We then closed with results showing that Jacket provides a fast approach to CPU computing.

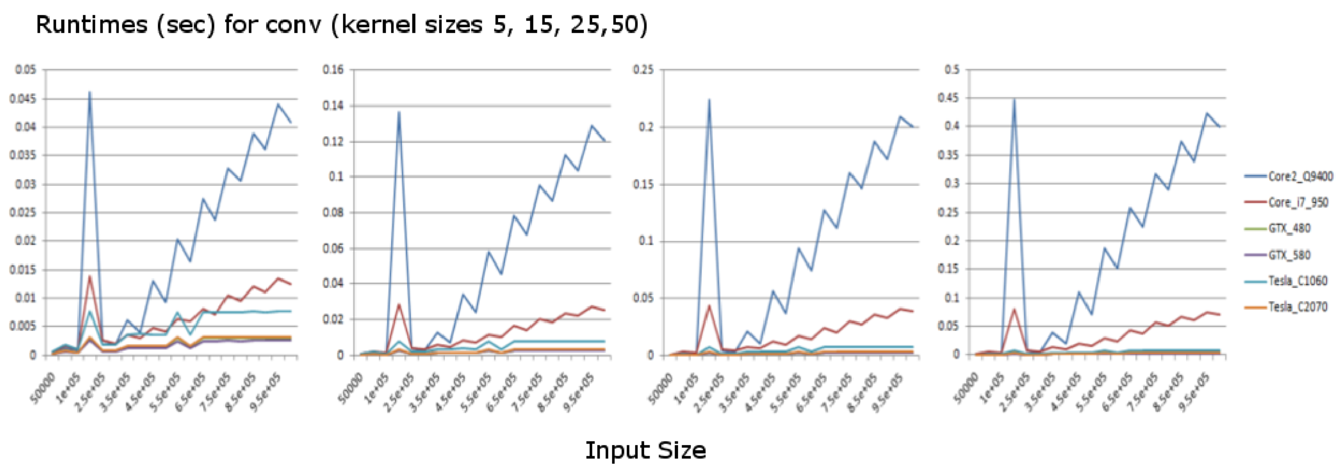


Figure 5. Various benchmarks.

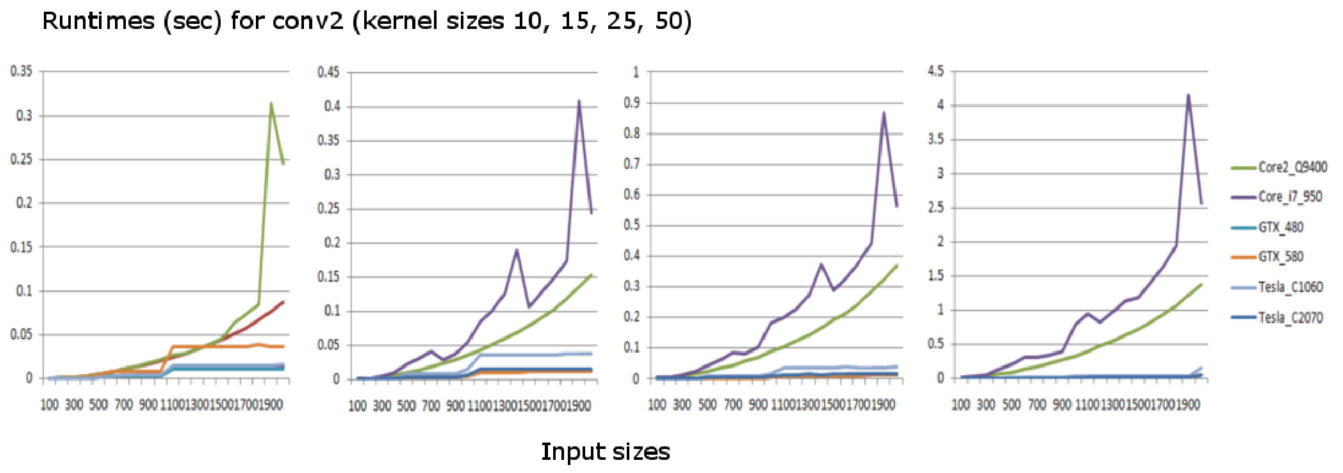


Figure 6. Various benchmarks.

REFERENCES

[1] AccelerEyes. Addr.: 800 W Peachtree St NW, Atlanta, GA 30308, USA. URL: `http://www.accelereyes.com`.

[2] The MathWorks, Inc. *MATLAB*. Addr.: The MathWorks, Inc., 3 Apple Hill Drive, Natick, MA 01760-2098, USA. URL: `http://www.mathworks.com`.

[3] Andy Webb: "MATLABs Racing Jacket". Automated Trader, vol. 16, no. 1, pp 54–61. 2010.

[4] Gaurav Sharma and Jos Martin, "MATLAB: A Language for Parallel Computing". *Springer International Journal of Parallel Programming*, vol. 37, no.1, pp. 3–36. 2008.

[5] AccelerEyes: "Jacket v1.7.1: Getting Started Guide". URL: `http://www.accelereyes.com/services/documentation`.

[6] Mark Harris, "Optimizing CUDA". *SuperComputing 2007 Tutorial*, Reno, NV, USA. November 2007.

[7] V. Volkov and J. W. Demmel,"Benchmarking GPUs to Tune Dense Linear Algebra". *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pp. 1-11, Austin, Texas, USA, 2008.

[8] Torben's Corner. Available at the Wiki of AccelerEyes at: URL: `http://www.accelereyes.com/wiki/index.php?title=Torben's_Corner`.

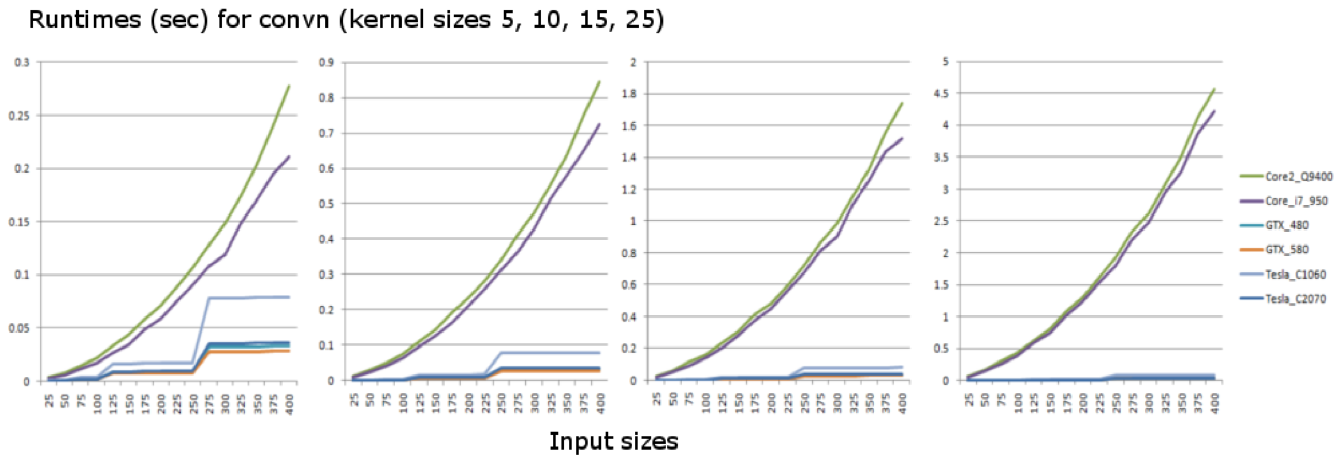


Figure 7. Various benchmarks.

[9] Goering, R., "MATLAB edges closer to electronic design automation world," EE Times, Oct. 2004.

[10] The Mathworks, "The Origins of MATLAB" Available at http://www.mathworks.com/company/newsletters/news_notes/clevescorner/dec04.html

[11] nVidia, "CUDA programming guide 1.1," Available at http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf

[12] nVidia, "CUDA CUBLAS Library 1.1," Available at http://developer.download.nvidia.com/compute/cuda/1_1/CUBLAS_Library_1.1.pdf

[13] nVidia, "CUDA CUFFT Library 1.1," Available at http://developer.download.nvidia.com/compute/cuda/1_1/CUFFT_Library_1.1.pdf

[14] nVidia, "Accelerating MathWorks MATLAB with CUDA," Available at http://developer.download.nvidia.com/compute/cuda/1_0/AcceleratingMATLABwithCUDA.pdf

[15] nVidia, "PTX: Parallel Thread Execution ISA Version 1.1", Available at http://www.nvidia.com/object/cuda_develop.html

[16] Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P., "Brook for GPUs: Stream computing on graphics hardware." *Transactions on Graphics and Visualization* vol. 23, no. 3, Aug. 2004.

[17] Tarditi, D., Puri, S., and Oglesby, J.. Accelerator: Using data parallelism to program GPUs for General-Purpose uses. In *International Conference on Architectural Support for Programming Languages and Operating Systems* (2006)

[18] McCool, M. and Toit, S.D., *Metaprogramming GPUs with Sh.* A K Peters, 2004.

[19] Mark, W.R., Glanville, R.S., Akeley, K., and Kilgard, M.J. "Cg: A system for programming graphics in a c-like language." *Transactions on Graphics and Visualization* vol. 22, no. 3, pp. 896-907, 2003

[20] Choy, R. and Edelman, A. "Parallel MATLAB: Doing it Right." *Proceedings of the IEEE* vol. 93, no. 2, pp. 331-341, 2005

[21] Matt Pharr, ed., *GPU Gems 2*, Addison-Wesley, 2005.

[22] Hubert Nguyen, ed., *GPU Gems 3*, Addison-Wesley, 2007.

[23] D. Göddeke, *GPGPU Basic Math Tutorial*, tech. report 300, Fachbereic Mathematik, Universitt Dortmund, 2005