

Accelerating MATLAB Using GPU Technology

Students: Katrina Barnett and Luke Parkinson

Association: Victorian Partnership for Advanced Computing

Supervisor: Mike Kuiper (Research Scientist)

Discipline: Engineering

Date: December 2008 – February 2009

Note: Please read the 2008/09 Summer Internship Project summary document "Evaluating the use of GPGPU's in Life Science and Super Computing applications" first, for background information on this case study

Objective

This component of the 2008/09 VPAC Summer Internship Program¹ was to evaluate and compare the available methods of accelerating MATLAB using GPU Technology.

Introduction

MATLAB, produced by MathWorks², is a complex and widely used program that has applications ranging from basic mathematics to advanced simulation and data processing. Much of the functionality of MATLAB is done using the built in or toolbox functions which are often optimised algorithmically to perform the desired calculations as quick (and as accurate) as possible. Despite this, some calculations in MATLAB can take a long time to complete, tying up a large amount of computer resources in the process. Many of the functions that are regularly used would benefit from GPU acceleration, and, due to the modular style of MATLAB this speed up seemed quite easy to obtain. From here we set out to evaluate some of the methods that are in development to speed up MATLAB calculations using the GPU.

Implementation

In order to evaluate the Jacket software we used the NVIDIA³ GPU machines (as currently Jacket makes use of CUDA⁴ which is only implemented for the NVIDIA cards). We used MATLAB version 7.6.0.324 (R2008a) and obtained Jacket version 0.6 (build 328) from the Accelereyes website⁵. This Jacket version gave a temporary free license to use and evaluate the software which expired on the 31st of January 2008 when it was replaced by the release of version 1.0 and associated commercial licence.

For the BLAS libraries we used the 0.7 beta version of the GPU accelerated ACML (AMD⁶ Core Maths Libraries) on the ATI GPU machines, again using the same 2008a version of MATLAB. The ACML binaries are freely available, and the GPU version can be

obtained from their ftp server⁷ or by request⁸, so do not require a license.

Software

ACML BLAS

ACML implements the standard BLAS (Basic Linear Algebra Subroutines), LAPACK (Linear Algebra Package) functions, as well as libraries for FFTs (Fast Fourier Transforms), random number generators and fast maths and fast vector mathematics⁹. The ACML-GPU libraries are still in development, but they are designed to take advantage of the stream processors on the GPU to accelerate some of these standard functions. The version we tested had only implemented DGEMM and SGEMM (Double and Single precision general dense matrix multiplication) on the GPU, with more planned for future releases. The premise being that other routines within LAPACK call these two functions, so are indirectly sped up.

To use the BLAS functions in ACML-GPU, we had to set the Linux environment variable `BLAS_VERSION` to the name of the main library (`libacml.so`). We then placed a symbolic link to the library in the standard MATLAB libraries directory.ⁱ Since the GPU version depends on the Compute Abstraction Layer (CAL), we needed to include these libraries somehow. MATLAB only supports one library in the `BLAS_VERSION`ⁱⁱ variable, so we got around this by including the extra library dependencies in the `LD_PRELOAD` environment variable¹⁰.

CUBLAS

We were hoping to perform the same style of implementation of the CUBLAS library as we did for ACML BLAS, using the NVIDIA graphics cards. CUBLAS is a CUDA implementation of the BLAS library and can be obtained with the CUDA SDK. The problem with using it in MATLAB was that CUBLAS does not use the standard interface that is used in BLAS (i.e. the

i `<install_dir>/matlab2008a/bin/glnxa64/`

ii To the best of our knowledge at the time.

accelerated functions have different names). As MATLAB calls the standard functions the advantages of using CUDA were not available. There was the possibility of creating a wrapper for CUBLAS which would allow MATLAB to use the functions using the standard interface but we did not do this as the time to implement this would outweigh the performance benefits, and again, very few functions have been accelerated.

Jacket

The major piece of software we looked at was Accelerereyes' Jacket. Jacket is a toolbox for MATLAB which uses code written in CUDA to accelerate calculations on the NVIDIA GPUs. This toolbox is similar to the NVIDIA CUDA plug-in¹¹ which allows the writing of MATLAB MEX files using CUDA to implement an acceleration. We did not choose to evaluate the CUDA plug-in directly as the features included in were also incorporated into Jacket and would be more user friendly under Jacket. The NVIDIA CUDA plug-in however is free and so could be an alternative option.

The advantage of Jacket is that the functions available can be used like any other MATLAB function. To cause a function to run on the GPU the variables and/or data in question needs to be cast as a *gsingle*. Then, when the *gsingle* variables are used the calculation is done on the GPU. Currently, there is no *gdouble* so all calculations done on the GPU must be single precision only. There is currently a limited set of functions that have been fully ported to the GPU and most of these use the exact same function call as the regular CPU variant. The functions available and their use are detailed in the Jacket User Guide which can be obtained from the Accelerereyes web page¹².

Jacket also includes a graphics toolbox¹³ which allows for better manipulation of graphs and images produced by MATLAB. This means that the GPU can be used for the calculations behind the graphics as well as the rendering. Unfortunately this does not currently work over remote systems so could not be tested with the set-up used here.

Jacket uses several other useful concepts such as a lazy execution style. This means that the computation on the GPU is only done when the answer is required. For example, if a multiplication is requested but the output suppressed, the calculation is only sent and completed when the output is used or displayed. The computation will also be completed when the queue of computation exceeds what can be done on the GPU. This gives the advantage of filling up the GPU buffer with computations making better use of the space and processors available, giving much more efficient performance for the amount of computation.

Benchmarking & Comparison

Bench() Benchmarking

We began to measure the acceleration by using the benchmarking function that is built into MATLAB, *bench()*. This function is designed to compare MATLAB running on the current hardware with other hardware configurations (same version of MATLAB), the data of which can be updated via the internet (Mathworks website¹⁴). *Bench* works by performing six tests to evaluate the running speed of some basic operations. The sequence of tests are run a designated number of times to enable an average to be produced (thus allowing for random occurrences or disruptions).

The tests done were:

- LAPACK which tests floating point operations and regular memory access.
- Fast Fourier Transform which tests floating point and irregular memory access.
- Ordinary Differential Equations which test data structures and M-file use
- Sparse System which tests mixed integer and floating point operations.
- 2D plot which tests the 2D line drawing graphics
- 3D plot which tests animated OpenGL graphics

The MATLAB *bench* function is not significantly sped-up when utilising the ACML-GPU subroutines. There is a significant gain (half the time) in the 3D plot, but that is due to the presence of a GPU, not the libraries themselves. There are slight gains in FFT and ODE times, which are probably due to calls to the GPU accelerated matrix multiplication functions, but the nature of GPGPU computation with slow memory transfers means that to achieve significant gains, the problem size would have to increase dramatically (greater than 1GB of data). This could be interpreted as a 'practical' test, in that problems of a size large enough to be accelerated well are uncommon, and realistically there is no benefit. However, we assumed that if GPU acceleration was necessary, then the problems would be of a significant size. Hence we decided that the *bench()* function was inadequate for testing the ACML-GPU capabilities.

Unfortunately, the *bench* function was not very useful for testing the Jacket implementation either as the numerical functions were designed to use double precision (which is the MATLAB default) and the implementation in Jacket is only single precision (it would not be 'fair' comparing single precision runs with double precision), also, as has been said above, only a limited number of functions are accelerated. This meant that the tests performed using the *bench* function did not produce any valid or meaningful results.

Matrix Order	Single Precision			Double Precision	
	CPU	ACML BLAS	Jacket	CPU	ACML BLAS
0	0.000005	0.000006	0.000454	0.031200	0.031063
1	0.000026	0.000026	0.000245	0.000042	0.000030
2	0.000005	0.000006	0.000144	0.000008	0.000007
3	0.000016	0.000007	0.000134	0.000007	0.000006
4	0.000035	0.000009	0.000133	0.000011	0.000010
5	0.000014	0.000019	0.000378	0.000031	0.000022
6	0.000125	0.000085	0.000377	0.000106	0.000119
7	0.000188	0.000483	0.000606	0.000539	0.000372
8	0.000956	0.002257	0.001775	0.001999	0.002283
9	0.005395	0.004076	0.006398	0.014830	0.015292
10	0.037211	0.014184	0.034770	0.081895	0.079059
11	0.287365	0.085949	0.152721	0.599314	0.578715
12	2.202176	0.604072	0.777990	4.644414	4.483960
13	17.332200	4.656838	4.586449	36.252179	34.954182

Table 1: Matrix multiplication benchmarking results. Note that the matrix order is n in $2^n \times 2^n$.

Matrix Multiplication Benchmarking

As both the ACML BLAS libraries and Jacket both implemented an accelerated version of matrix multiplication it was a good candidate for some comparative, if limited, benchmarking. To do this two dense matrices were generated (using the pseudo random number generator built into MATLAB – no explicit seed) and multiplied together. The version used with Jacket used the *gforce* command here to ensure that the multiplication was timed (and not left up to lazy execution). This was done for matrices of sizes 1×1 to $2^{13} \times 2^{13}$ (note that this equates to a 8192×8192 matrix at the upper end). The maximum was chosen as this appeared to be where the limit of the GPU memory was reached, whether this will be altered or better arrangements (as far as memory management goes) are made in the future remains to be seen.

As can be seen by the results above, the execution time (as recorded by MATLAB using the *tic* and *toc* paired commands) is decreased when either of the acceleration methods are used, especially in the single precision case. The differences are only really noticeable as the size of the matrix increases. Two graphs of the results are shown below which better illustrates the speed up, note that Figure 2 is a subset of Figure 1, allowing the behaviour for smaller matrices be seen better on the scale.

From these results it can be seen that the ACML BLAS library acceleration gives a small advantage over the non-accelerated version when used with double precision calculations whereas a large advantage, approximately $\times 3.7$ speed up, is seen with both the Jacket and the ACML BLAS acceleration when used in single precision. While the time taken is largely identical for the smaller to midrange matrices, the very small advantages would become advantageous when the same size calculations are done multiple times and when the simple matrix multiplication is used as part of a larger function. The small improvement may also be because MATLAB already includes optimised algorithms

for matrix multiplication which would give such fast results when only using the CPU.

Another discovery that was made was that the graphics cards performed better (i.e. faster computation times) once some computations had already been performed on them – once they had in effect been 'warmed up'. This manifested in some initial ('cold') runs of the matrix multiplication code giving longer timings for matrices around the $2^4 \times 2^4$ to $2^7 \times 2^7$ size range. Once the computations were repeated (with different, pseudo randomly generated data to ensure there was nothing saved in cache or memory) the gradual scale up shown in the data of Table 1 above was observed and maintained through successive runs.

Future Possibilities

The BLAS and LAPACK libraries are used in many academic programs and will continue to be used as they are the standard for mathematical computation. If they can be accelerated on the GPU, many existing software applications could benefit from a huge performance boost. Earlier, we explained our results which showed an example of this speed gain, albeit in a limited area. However, AMD have stated that they will continue to expand the GPU accelerated functions in their ACML-GPU implementation of these routinesⁱⁱⁱ. If NVIDIA can make their interface conform to the standard with CUBLAS, and they do same as AMD-ATI by offloading more and more functions onto the GPU, this could become a powerful platform-independent way to accelerate existing software relatively painlessly (only requiring to link a different library which could be automated in MATLAB and other software).

As said above, Jacket is still very new in development and very few functions have been successfully implemented on the GPU at this point in time. Despite this there seems to be an active development

ⁱⁱⁱ This statement is in the official documentation released with the libraries themselves.

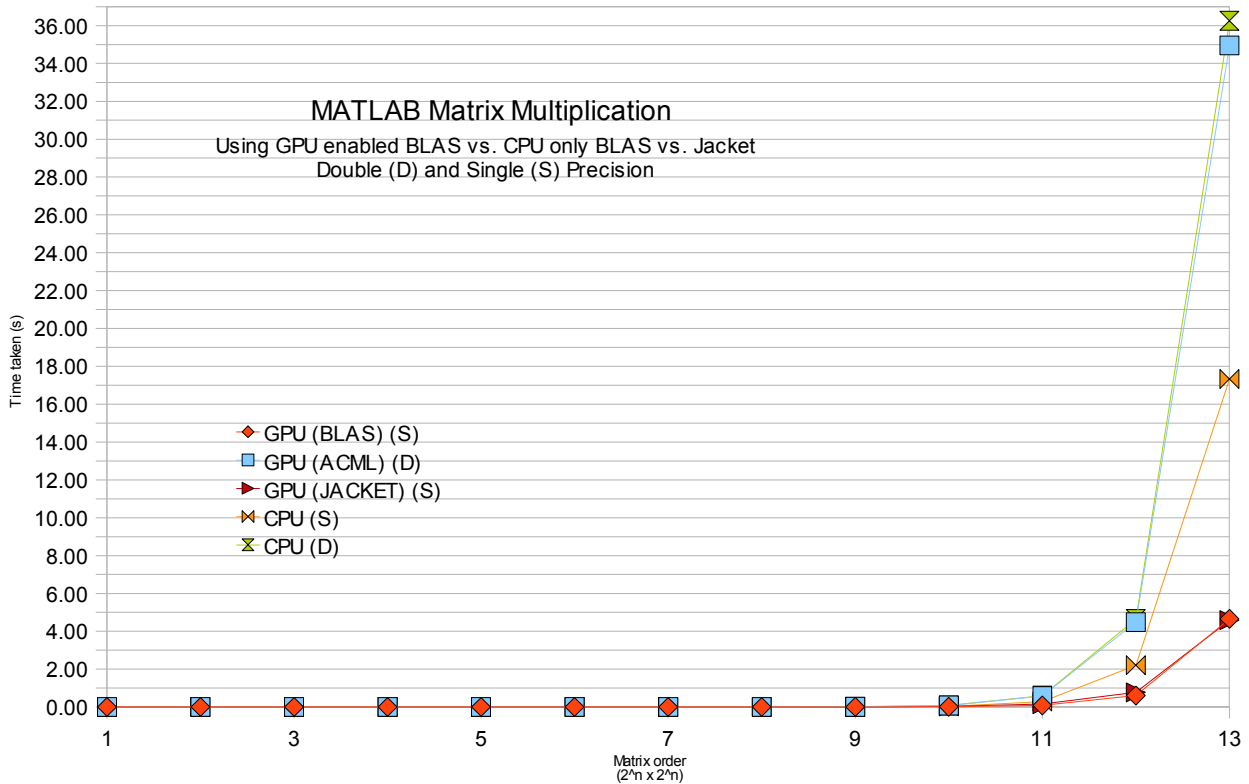


Figure 1: A graph of the time taken to perform matrix multiplication using single and double arithmetic. Using GPU acceleration with Jacket and the ACML BLAS libraries.

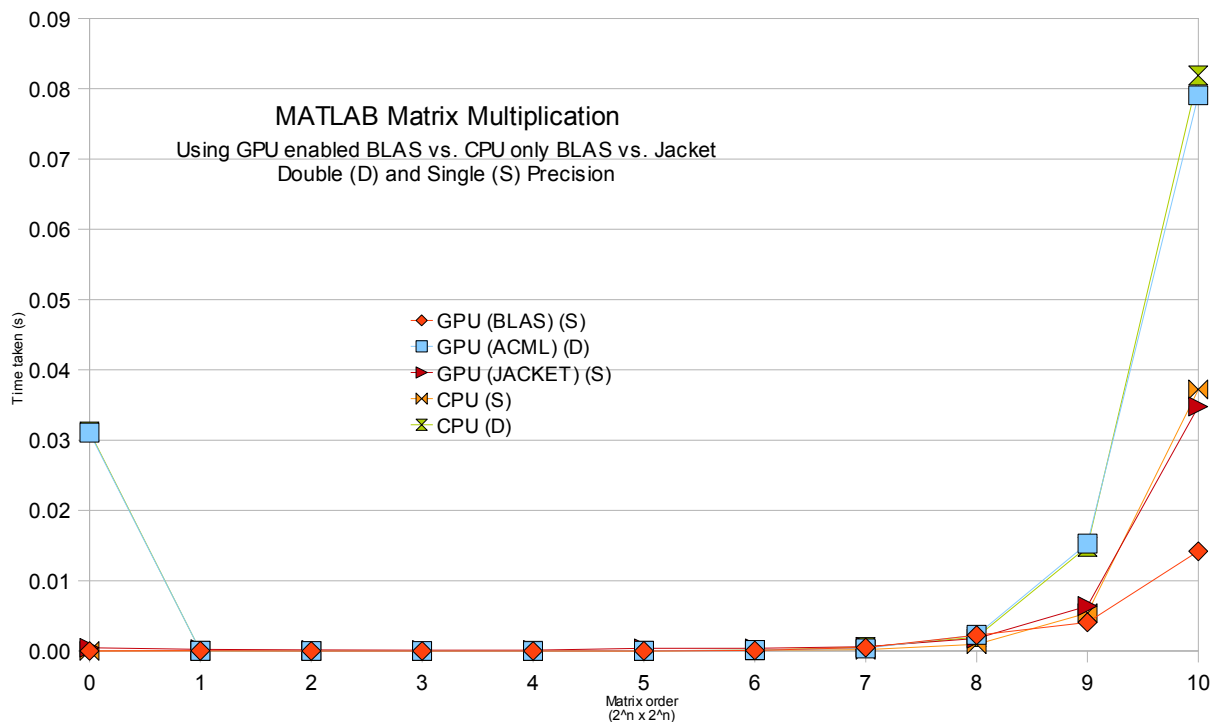


Figure 2: Graph of a subset of the data to show the behaviour at smaller matrix orders. Note the initial small 'matrix' (really a scalar) penalty for double precision.

community with forums¹⁵ that are regularly monitored for user help. As this is commercial software it does require a license fee to be paid which may discourage some users from adopting it at this early stage but as time moves on more functionality should be implemented and hopefully the speed ups observed will become more worthwhile.

Another welcome addition will be double precision calculation support as many scientific tasks require such a high accuracy. An added benefit of the way that both MATLAB and Jacket are built is that the more complex functions are build on top of the simpler, more basic functions which means that as these basic functions are accelerated the functions higher up in the hierarchy will also receive a performance increase. This will help in accelerating some of the other, third-party, toolboxes. As these are based on the simpler functions the same benefit will be observed.

Conclusion

We have seen some significant speed up of execution for the simple functions, namely dense matrix multiply, that have been accelerated to use GPU technology. Both the ACML BLAS libraries and the toolbox Jacket have shown this improvement but with varying implementation styles. While this is a great result there is much more work required to be done in the future before real advantages will be seen from using a GPU to speed up general MATLAB calculations.

References

- 1 VPAC Summer Internship Program: <http://www.vpac.org/training/internships>, Feb 2009.
- 2 MathWorks website: <http://www.mathworks.com/>, Feb 2009.
- 3 NVIDIA website: <http://www.nvidia.com>, Feb 2009.
- 4 CUDA website: http://www.nvidia.com/object/cuda_home.html#, Feb 2009.
- 5 Accelereyes/Jacket website: <http://www.accelereyes.com/>, Feb 2009.
- 6 AMD website: <http://www.amd.com>, Feb 2009.
- 7 <http://forums.amd.com/devforum/messageview.cfm?catid=217&threadid=98925&enterthread=y>, Feb 2009.
- 8 <http://ati.amd.com/technology/streamcomputing/sdkdwnld.html>, Feb 2009.
- 9 For more information, please see the documentation included with these libraries.
- 10 For more information on Linux environment variables, please see the online documentation.
- 11 MATLAB CUDA plug-in: http://developer.nvidia.com/object/matlab_cuda.html, Feb 2009.
- 12 Jacket Documentation: <http://www.accelereyes.com/documentation.php>, Feb 2009.
- 13 Graphics Toolbox: <http://www.accelereyes.com/graphics-toolbox.php>, Feb 2009.
- 14 Bench update files: <http://www.mathworks.com/matlabcentral/fileexchange/1836> (Accessed 16th February 2009)
- 15 Jacket Forums: <http://www.accelereyes.com/forums/>, Feb 2009.