Accelerating Algorithms on GPUs in SCIRun:

the Conjugate Gradient Case Study

Devon Yablonski
Northeastern University
Boston, Massachusetts
yablonski.d@husky.neu.edu

Miriam Leeser
Northeastern University
Boston, Massachusetts
mel@coe.neu.edu

Dana Brooks
Northeastern University
Boston, Massachusetts
brooks@ece.neu.edu

Abstract

The goal of this research is to integrate graphics processing units (GPUs) into SCIRun, a biomedical problem solving environment, in a way that is transparent to the scientist. We have developed a portable mechanism that allows seamless co-existence of CPU and accelerated GPU computations to provide the best performance while also providing ease of use. Features include integration into the existing graphical user interface (GUI) as well as easy extensibility of GPU processing to future algorithm development. As a case study, we focus on the linear solving algorithms of SCIRun for sparse data, including the conjugate gradient (CG) with Jacobi preconditioner. Acceleration of nearly 6x was achieved using NVIDIA's CUDA with sparse matrices, demonstrating the performance of our approach. The linear solving algorithms were chosen for their suitability for acceleration on the parallel processing architecture that NVIDIA's GPUs exhibit and illustrate a mechanism that can be extended to other existing and future algorithms in SCIRun.

Keywords: GPU; Acceleration; Sparse Matrix; CUDA

I. SCIRun & CUDA

SCIRun, developed at the University of Utah, is a problem solving environment aimed at the biomedical research community [1]. It allows the user to easily implement a sequential network of mathematical functions to process data and simulate results. The user creates this network by selecting computation modules in the GUI and connecting them according to the desired data flow. Each module has mathematical algorithms or data I/O functionality, written in C++, allowing the user to create the simulation without advanced programming and mathematical knowledge. We have created a method for integrating GPU acceleration into user applications without breaking this user friendly paradigm.

GPUs are of great interest in the high performance computing community due to the high level of parallelism in their single instruction multiple data [2] (SIMD) architecture. Specifically, NVIDIA is a leader in making GPUs accessible to programmers for use in scientific research. NVIDIAs CUDA [3] is an extension of the C language that allows the many processing cores of the GPU to be accessible to the programmer. There are an increasing number of tools and open source GPU projects available in the academic and scientific research communities, making programming with this hardware efficient in both effort and performance. We use NVIDIA's CUDA C to accelerate the algorithms in SCIRun.

II. Design & Performance

We have designed a reproducible and adaptable code structure to allow GPU acceleration in SCIRun that is maintainable and replaceable on a modular basis. We used the SCIRun "SolveLinearSystem" module as a case study for formulating a design that integrates well into the SCIRun environment. The algorithms in this module, including the conjugate gradient with Jacobi preconditioner, benefit from the massively parallel GPU hardware compared to the more sequential nature of CPUs for which SCIRun was originally designed.

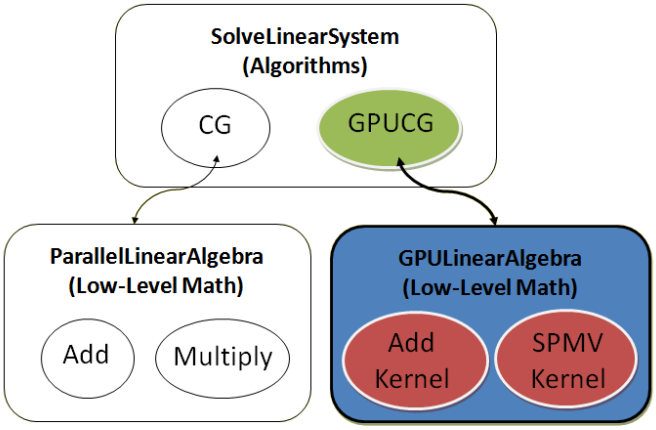

Figure 1: Layout of SolveLinearSystem with our GPU implementation
Note: SPMV = Sparse Matrix Vector Multiplication

Most data in these simulations are in the form of sparse matrices that are very large in size (107k x107k for the largest example in table 1). SCIRun implements a form of compressed row storage [4] (CRS) that allows more efficient memory usage as only non-zero entries are stored. As a trade-off for the memory savings, the computation using the sparse format has more overhead as column and row information for each data point is stored in multiple arrays that must be referenced to compute any matrix computation, such as scaling or sparse matrix-vector multiplication (SPMV). CRS is

a necessary strategy due to the relatively low amounts of memory available on most GPUs but makes it more difficult to achieve speedup.

SCIRun abstracts complex calculations in its code by using a "ParallelLinearAlgebra" class that contains low level math functions like matrix vector multiplication, vector-vector addition and others. This design allows algorithms (like CG) to be written at a higher, more readable level in the module code. We have extended this abstraction by creating a duplicate class, "GPULinearAlgebra" that contains identical lower level functions to perform the calculations on the GPU. We have created sparse matrix vector multiplication, scaling, addition, subtraction and other custom CUDA kernels in this class. This provides a mechanism for any programmer to write their higher level mathematical algorithm and run it on the GPU with little or no GPU programming experience or knowledge. Instead of creating CPU matrices and vectors, GPU matrices and vectors are created (in a nearly identical way) and the functions are named and used the same way as the CPU functions.

Using this structure, the algorithms in the SolveLinearSystem module have been implemented using the GPU. The CPU version of the conjugate gradient algorithm was duplicated and a simple replacement of all CPU algebra object types with the GPU counterpart created a GPU version. The algorithm ran on an NVIDIA 280GTX GPU providing speedup of over 5x in double precision (Table 1) over the single threaded CPU version running on an Intel Core 2 CPU at 1.86GHz. This speedup is an end-to-end speedup including all data transfer to and from the GPU. The input data is from a sample network of heart ischemia available at the SCIRun website [5]. The correctness of the results has been verified by comparing the outputs of the GPU and original CPU versions of the code.

The SolveLinearSystem module contains other algorithms including: biconjugate gradient, MinRes, and Jacobi[6]. We illustrated the usability of the GPU implementation by accelerating these algorithms as well, using the same approach we used for CG. Similar performance was achieved, indicative of the extensibility of our approach.

The performance is on par with other GPU studies due to the fact that we are computing with sparse matrices and in double precision. Double precision computation is only recently available; the NVIDIA 2xx GTX series was the first to provide this capability. Double precision is necessary to achieve comparable results to the CPU in many scientific problems, including CG. Furthermore, the linear solvers are iterative computations with inter and intra iteration dependencies, so only the individual matrix and vector computations can be parallelized. Finally, the speedup was achieved with no modifications to the algorithm design from the original SCIRun code, nor was the CUDA implementation optimized for any specific problem size or specific dataset. Optimizations of the CUDA code for the solver implementations are part of our future work.

As previously mentioned, there are many open source GPU projects that could provide instant acceleration for applications, including many algorithms in SCIRun. Our design allows us to easily employ third-party GPU accelerated algorithms in our GPULinearAlgebra class for specific GPU tasks in order to achieve better performance. To demonstrate this, we integrated a sparse matrix conjugate gradient implementation created by the Gocad Research Group at Nancy Université in France[7]. Using this CG solver, slightly improved performance of 5.9x was obtained compared to our GPU module which is based on translations of individual operations such as matrix vector multiply. This is expected since this third-party version has optimized both the algorithm and the kernel for CG and the input data size. The tradeoff is that more programming knowledge is required to include third party source code in the GPULinearAlgebra class and to convert SCIRun's data to the format expected. The resulting algorithm code is less readable since it is abstracted below the normal algorithm layer of SCIRun. Our design provides close to the same performance while maintaining ease of use. When desirable, advanced techniques including third party code or a kernel implementing an entire algorithm can be integrated using our high level interface.

Data	Matrix Size	Time(s)		Speedup
		CPU	GPU	
Sample 1	$62K^2$	50.05	15.27	**3.3x**
Sample 2	$83K^2$	19.82	6.14	**3.2x**
Heart Ischemia	$107K^2$	7.42	1.46	**5.1x**
Heart Ischemia*	$107K^2$	164.57	27.98	**5.9x**
* This run is without preconditioner, illustrating long runs provide increased performance.				

Table 1: Performance of CG GPU vs CPU in SCIRun

III. Conclusions & Future Work

This research is a case study of GPU integration into SCIRun. Our GPU linear algebra design achieves our original goal by demonstrating a structure for conveniently applying GPU hardware to problems in SCIRun and remaining transparent to the scientist using the application. We achieved acceleration of nearly 6x in both the third-party software and our own, easy to use interface. This approach to GPU programming is extendable to other modules in SCIRun, and is easily adaptable to GPU hardware changes in the future.

Future work will focus on automated architecture choice between CPU and GPU algorithms at run-time to provide the most efficient completion of the linear solving algorithms on available hardware. Also, more optimization to the sparse

matrix computations on the GPU will be explored to maximize performance.

Acknowledgment

This work was supported in part by the National Science Foundation Engineering Research Centers Innovations Program (Award Number EEC-0946463) and CIBC, the Center for Integrative Biomedical Computing, an NIH / NCRR funded P41 center, grant P41-RR12553-10, which develops and maintains SCIRun and supports the third author. We would like to also thank Jeroen Stinstra and Darrell Swenson from the University of Utah for their guidance and for providing data (matrices) to test.

References

[1] S.G. Parker, D.M. Weinstein, & C.R. Johnson. *The SCIRun computational steering software system.* Modern Software Tools in Scientific Computing, E. Arge, A.M. Bruaset, & H.P.Langtangen, eds. Birkhauser Press, 1997

[2] Single Instruction Multiple Data, SIMD. Wikipedia http://en.wikipedia.org/wiki/SIMD

[3] NVIDIA's CUDA C http://www.nvidia.com/object/what_is_cuda_new.html

[4] F.S. Smailbegovic, G. N. Gaydadjiev, S. Vassiliadis. *Sparse matrix storage format*. Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing. ProRisc 2005, pp. 445-448, Veldhoven, the Netherlands, November 2005

[5] SCIRun Data Examples: http://www.sci.utah.edu/software.html

[6] Shewchuk, J.R. *An introduction to the conjugate gradient method without the agonizing pain*. Technical report, CMU School of Computer Science, 1994

[7] L.Buatois, G.Caumon & B.Lévy. *Concurrent number cruncher: an efficient sparse linear solver on the GPU.* High Performance Computing and Communications, Third International Conference, HPCC 2007