Hierarchical Parallelization of a Radio Frequency Tomography Application via Multiple GPUs

Dana Schaa[1], Mark Barnell[2], Roope Astala[3], Steve Reinhardt[3], and Viral Shah[3]
[1]Northeastern University [2]Air Force Research Lab, Rome, NY [3]Interactive Supercomputing
dschaa@ece.neu.edu mark.barnell@rl.af.mil {rastala, sreinhardt, vshah}@interactivesupercomputing.com

Abstract
Faced with the excruciating pressure of the drastic performance needs of embedded applications on one side, the difficulty of programming heterogeneous multicore cluster architectures on another, and rapid-development requirements on a third, embedded application developers are reconsidering the conventional wisdom that productivity languages such as M and Python are only for prototyping and not for deployment. Prior work has shown the scalability of M programs to 128-512 cores via the Star-P parallel platform and the feasibility of acceleration of M programs by hardware accelerators. In this work on a radio frequency tomography application, we show that O(32) GPUs can be used effectively on a single problem, and that the absolute performance of the CPU-GPU linkage is sufficiently fast not only for weak scaling, but also for strong scaling within limits, delivering near-real-time performance.

Background
Numerous attempts at M language parallelism have been made [Choy05], of which the most widely used current implementations include pMATLAB [Mullen], the Parallel Computing Toolbox™ [Sharma], and Star-P™ [ISC]. Each of these tools seeks to enable use of parallelism with modest changes to the serial code.

The Radar Signal Processing Technology Branch of the Air Force Research Lab (AFRL) develops radar frequency (RF) tomography algorithms [Wicks]. They are typically written in the M language of MATLAB[†] for rapid algorithmic development, and also have demanding performance requirements that cannot be met without parallelism [Elton08]. Exploiting parallelism while preserving algorithmic flexibility has high value to the larger project.

Prior work [Elton09] has explored the parallelization of these algorithms with Star-P on general-purpose (*i.e.*, x86-64) processors, including the practical issues involved in running at large shared HPC centers, such as security and scheduling, and [Murphy] the use of hardware accelerators with M programs.

Aims
This work explores another dimension of performance for these algorithms, namely the use of general-purpose graphic processing units (GPGPUs) for the bulk of the computational work, instead of general-purpose CPUs. Given the data size and computational intensity of typical RF tomography data, many GPUs must be used to meet the absolute performance demands, leading to the need to balance the parallelism within a chip with the parallelism across many chips. Multiple GPUs can accelerate the performance of a fixed data size (*i.e., strong* scaling) or can cope with larger data sizes by scaling the number of GPUs proportionally with the data size (*i.e., weak* scaling).

Methods
The algorithm exhibits parallelism among different pulses of radio waveforms and among pixels in each image. The algorithm is viewed as an M program, but the kernel that consumes more than 99% of the time was already coded in C for higher speed. The first task was to extend the C code to the CUDA interface [Lindholm], simultaneously refactoring the processing between the C/CUDA code and M code so that the C/CUDA code would work correctly on a subset of the pulses in the data. CUDA 2.0 (Bale) and 2.2 (Lincoln) and Star-P version 2.8 (both) were used for the tests, with the Star-P client installed collocated with the server. The Mex data-exchange interface was used to pass data between the M and C/CUDA functions executed via the Star-P Task Parallel Engine. The *ppeval* task-parallel construct in Star-P was used to call multiple instances of the C/CUDA kernel (each working on its own data) simultaneously. Numerical differences did exist between double-precision execution in CPUs and single-precision execution in GPUs, but the differences were not visible in the resulting images. The initial data size was 180 pulses, which we used for our strong scaling work and as the per-node size for weak scaling.

The first system used was the *Bale* cluster at the Ohio Supercomputer Center, the GPU-accelerated portion of which consisted of 8 nodes, each with two AMD Opteron DualCore sockets (2.6Ghz) and two NVIDIA Quadro FX 5600 GPUs, each with 128 cores running at 600MHz . The second cluster used was the *Lincoln* cluster at the National Center for Supercomputing Applications, which consisted of 192 computer nodes, each a Dell PowerEdge 1950 dual-socket node with quad-core Intel Xeon E5410 (“Harpertown”) sockets (2.33GHz). Pairs of these nodes shared access to a NVIDIA Tesla S1070 accelerator, each containing 4 Tesla GPUs, each with 240 cores running at 1.44GHz. All the GPUs used were capable only of single-precision floating-point arithmetic.

Performance Results

The original serial algorithm processes 180 pulses in 45.2 seconds on a single (x86-64) core of Bale.[‡] With the CUDA kernel, it executes in 1.07s on 2 GPUs on a single node of Bale or 1.00s on 1 GPU of Lincoln, 42.2 and 45.2 times faster, respectively, than a single x86-64 core. We quickly discovered that, on Lincoln, using a second GPU on a node ran almost 2X slower, which we attributed to bandwidth-sharing on the PCI-Express 8x link (contrasted with PCI-Express 16x links on Bale); thus our Lincoln results used only a single GPU per node, while the Bale results used 2 GPUs per node.

The first dimension we explored was weak scaling, using the basic unit of 180 pulses per node. As expected (see Figure 2), on Bale each added node is as efficient as the

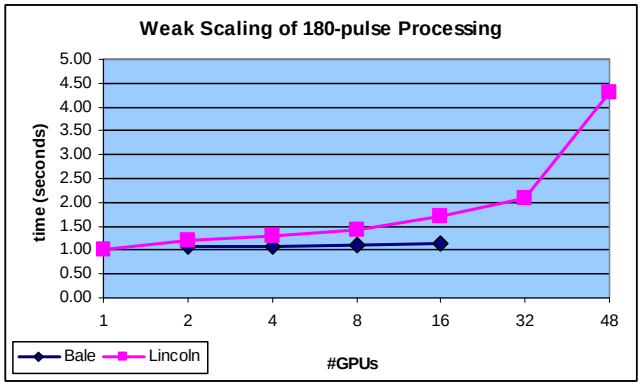

Figure 1. Weak Scaling on Bale and Lincoln

first, enabling solution of an 8X larger problem in close to the same time. On Lincoln, the results degraded somewhat up to 32 GPUs and then markedly at 48 GPUs, for reasons we do not fully understand. With the 32-GPU case, this application effectively uses about 8,000 (32*240) cores.

With success at demonstrating weak scaling across a

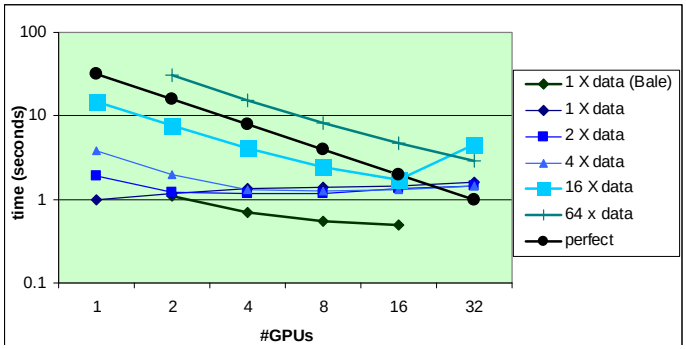

Figure 2. Strong Scaling on Bale and Lincoln

modest number of GPUs, we turned our attention to a more demanding problem, that of reducing the run-time of a given problem size as far as practical by adding more GPUs, also known as *strong* scaling. While this is clearly desirable in the interests of absolute performance, there are minimum amounts of work per chip or per core that we approach, given that each GPU has hundreds of cores in it, and the basic data size we used (180 pulses) runs for only 45 seconds on a single CPU core. Figure 2 shows a family of speed-up curves for different data sizes, with the slope of the Perfect line for comparison[§]. On Lincoln, scaling is good until each GPU is executing only 180 pulses, which we did not try to subdivide further among GPUs. On Bale, we subdivided even 180 pulses among multiple GPUs, showing that a factor of two reduction in absolute time is possible even if efficiency (on 8 GPUs, 4 times more GPUs to get 2 times more performance) is not ideal.

Summary

This work extends the use of the M productivity language deeply into parallelism, with one level of parallelism exploited within a GPU (via CUDA) and a second level of parallelism exploited between GPUs (via Star-P). Weak scaling performs very well within the tested range (modulo the anomalies on Lincoln). Strong scaling performs well down to the granularity of the basic data size, and bears more investigation of subdividing further to achieve absolute times below one second.

Acknowledgments

The authors acknowledge the gracious help of OSC and NCSA personnel in their use of the systems described, and Niraj Srivastava's help with the systems at AFRL/Rome.

References

[Choy04] R. Choy, A. Edelman, J.R. Gilbert, V. Shah, and D. Cheng, *Star-P: High Productivity Parallel Computing*, High Performance Embedded Computing Workshop, 2004.

[Elton08] B.H. Elton and K.M. Magde, *A Scalability Study on Multicore Cluster Systems of an AFRL Radar Frequency Tomography Imaging Code Written in MATLAB® for Parallel Execution Using Star-P®*, Proceedings of the DoD HPC Modernization User Group Conference, IEEE 2008.

[Elton09] B.H. Elton, S. Samsi, H.B. Smith, L. Humphrey, B. Guilfoos, S. Ahalt, A. Chalker, K.M. Magde, N.K. Srivastava, A.H. Abdullah, and P. Boyle, *Practical High Performance Computing: A Case Study*, SC09, 2009.

[ISC] Interactive Supercomputing, Inc., *Star-P® Programming Guide for Use with MATLAB® (Release 2.8)*, 2009.

[Lindholm] Lindholm, E., Nickolls, J., Oberman, S., and Montrym, J., *NVIDIA Tesla: A Unified Graphics and Computing Architecture*, IEEE Micro 28(2) pp. 39-55, 2008.

[Mullen] J. Mullen, N. Bliss, R. Bond, J. Kepner, H. Kim, and A. Reuther, *High-Productivity Software Development with pMatlab*, Computing in Science and Engineering 11(1), 2009.

[Murphy] M. Murphy, M. Raymond, and S. Reinhardt, *Automatic Mapping of MATLAB Code to Parallel FPGAs on the SGI Altix*, HPEC Workshop, 2005.

[Sharma] G. Sharma and J. Martin, *MATLAB®: A Language for Parallel Computing*, International Journal of Parallel Programming 37(1), 2009.

[Wicks] M.C. Wicks, B. Himed, J.L.E. Bracken, H. Bascom, and J. Clancey, *Ultra narrow band adaptive tomographic radar*, 2005 1st IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing. IEEE, 2005.

[‡] Or 22 seconds on one core of an Intel® Xeon® 5450 socket(3.0GHz).

[§] Note that the time to store the resulting image was not counted in the strong scaling case, as the systems were not configured for high I/O bandwidth, unlike systems actually deployed for this application.