Power Consumption of Desktop and Mobile GPU's for IRSTAP Applications

Michael Roeder, roederm@saic.com
Jeremy Furtek, Jeremy.d.furtek@saic.com
Nolan Davis, nolan.r.davis@saic.com
Cezario Tebcherani, cezario.e.tebcherani@saic.com
Masatoshi Tanida, masatoshi.tanida@saic.com
Dennis Braunreiter, dennis.c.braunreiter@saic.com

Scientific Applications International Corporation

Abstract

Emerging capabilities in stream and multi-core computation along with high speed memory bandwidths in commercial graphics processor (GPU) architectures are enabling breakthrough low cost and low power teraflop computing solutions to DoD embedded computing challenges.

Under the DARPA MTO STAP-BOY program, SAIC has developed mappings of complex signal processing algorithms to GPU architectures. The work discussed in this presentation is from STAP-BOY phase two which has focused on using CUDA from NVIDIA to develop prototypes for fieldable applications running on NVIDA GPUs in low power high performance embedded environments.

For phase two of the DARPA STAP-BOY program, we have implemented a highly adaptive IR STAP algorithm using CUDA tools from NVIDIA. Unlike many previous algorithms developed [1] this algorithm performs all STAP computation from end to end entirely on the GPU without intermediate GPU to CPU transfers. This end to end implementation allows for considerable performance improvement and allows for considerable power savings.

In our presentation we will discuss the performance and performance to power ratios for the kernels in our end to end STAP algorithm running on both desktop and mobile GPUs.

The conclusions reached in the STAP-BOY phase two program is that modern GPU API's such as CUDA lead to order(s) of magnitude power/watt/cost improvements over CPU and DSP solutions. Additionally with the new tools that are available a very wide variety of algorithms can be mapped efficiently to GPUs with modern programming APIs such as CUDA, with high productivity in application development.

IR STAP Overview

The goal for this demonstration application is to detect all people moving within a large city environment in real time. Figure 1 shows example output of our STAP algorithm within a notional graphical user interface. The box on the upper left shows a zoomed out view of the entire city, the other three boxes are zoomed in displays highlighting individual people walking in the city.

The input to our application is a live IR video feed and the output is a detection list of objects that match user specified signatures and velocity hypotheses. For our test data we have used a sequence of 5Kx5K pixel images taken of a small city at 2hz with roughly 0.6m/pixel resolution. Our goal for this application was to detect people at near real time using multiple low power GPUs.

Figure 1: IR STAP notional graphical user interface.

The STAP processing chain roughly breaks down into six steps that are described below.

1. Convert Input Data to 0 Mean Data ("Demean")
2. Adaptive Covariance Estimation
3. Compute Adaptive Matched Filter (AMF) Weights
    a. Factor Covariance Matrices
    b. Solve for adapted weights
4. Apply Weights with Convolution
5. Generate Adaptive CFAR Threshold Estimates
6. Gather Detections from Convolution Output

These six processing steps are performed entirely on the GPU using multiple CUDA kernels that exhibit a variety of memory access and processing patterns. The data flow between these six processing steps is roughly diagramed in Figure 2.

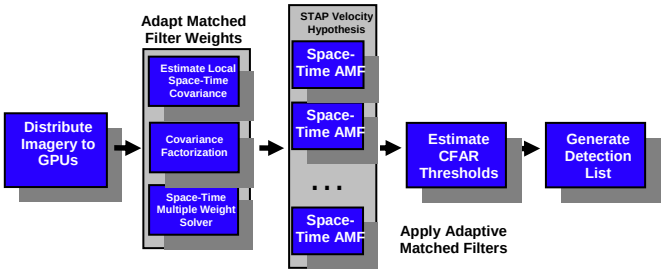

Figure 2: IR STAP data flow diagram.

Performance Measurements

Table 1 compares the performance of desktop vs. mobile GPU parts for three of our IR STAP kernels. The results we have attained with mobile parts have indicated the 8800 offers excellent power scaling opportunities for embedded computing. For many of our benchmarks the 8800M GTX has offered performance near that of the 8800 GTX despite the fact that it consumes just over 1/3 the power of the 8800 GTX.

Table 1: 8800 GTX vs. 8800M GTX

	8800 GTX		8800M GTX	
Operation	GFLOP/s	GFLOP/W	GFLOP/s	GFLOP/W
Cholesky	10.46	0.059	7.41	0.116
Weight Solve	9.77	0.055	6.55	0.102
Covariance	113.33	0.640	62.86	0.982

Figure 3 describes the relative processing times for the various processing steps of our IR STAP application. This diagram shows that two bottlenecks that were common with earlier GPU implementations, namely CPU/GPU memory transfers and scatter/gather operations, have been overcome in our implementation. Prior to CUDA, GPUS were programmed with shader programming languages that required frequent CPU/GPU memory transfers and often limited the programmer's ability to perform scatter/gather operations. Using CUDA we were able to implement our application such that it requires few memory transfers and performs a gather step (detection gather) very rapidly. In our implementation the transfer and detection steps only take 7% and 1% of our overall processing time.

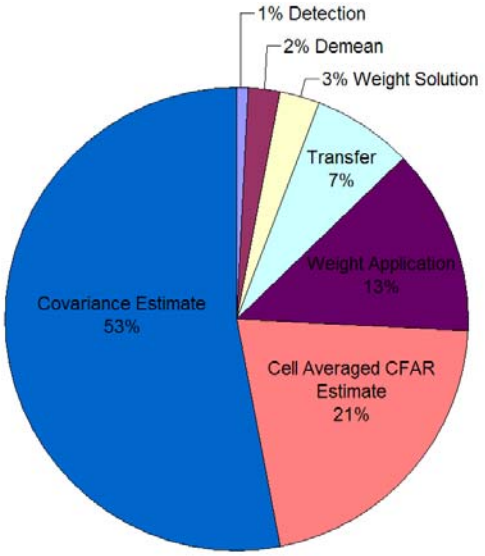

Figure 3: IR STAP execution profile.

Figure 4 shows that GPU based matrix multiplication operations exhibit significant improvements to the power/performance ratio over both traditional CPU based architectures as well as to IBM Cell architectures.

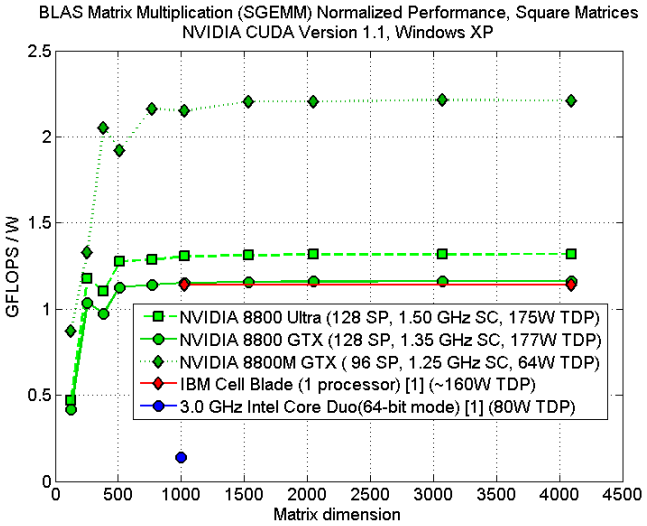

Figure 4: GFLOPs per Watt for Multiplication on Various Architectures[1]

The performance and performance/power ratios of our various kernels will be discussed in greater detail in our presentation.

Conclusions

Many of the limitations that previously existed to GPU programming have been overcome with modern programming tools such as CUDA. With CUDA a wide variety of applications and types of parallelism can be exploited with excellent performance and little CPU/GPU transfer overhead.

In addition GPU's offer excellent performance/power ratios and the ability to scale power consumption up or down by using different GPU parts (mobile vs. desktop) which make them well suited for low power high performance applications.

With these improvements to development tools and their excellent power characteristics GPUs are becoming competitive with FPGA and DSP processors for DSP applications.

References

[1] D. Healy, D. Braunreiter, J. Sillaci, D. Boe, J. Furtek, and X.Sun, "DARPA STAT BOY: Fast Hybrid QR-Cholesky Factorization and Tuning Techniques for STAP Algorithm Implementation on GPU Architectures", *HPEC 2007 Workshop*, Lexington, MA, September 2007

[2] GPU Sgemm Implementation, Vasily Volkov, NVIDIA CUDA Forums, http://forums.nvidia.com/index.php?showtopic=47689

[3] NVIDIA Corporation, "NVIDIA CUDA Compute Unified Device Architecture Programming Guide", Version 1.1, June 2007

[1] GPU matrix multiplication results generated with SGemm implementation by Vasily Volkov [2]