Benchmarking the NVIDIA 8800GTX with the CUDA Development Platform

Michael P. McGraw-Herdeg, Massachusetts Institute of Technology, *mherdeg@mit.edu*
Douglas P. Enright, The Aerospace Corporation, *Douglas.P.Enright@aero.org*
B. Scott Michel, The Aerospace Corporation, *scottm@aero.org*

Abstract

Two HPEC Challenge benchmarks, finite impulse response and QR decomposition, were implemented on a NVIDIA 8800 GTX graphics card using a data-parallel implementation approach. For the finite impulse response filter bank benchmark, a fast convolution FFT-based frequency-domain approach on the GPU performed 4 to 35 times faster than the comparable calculation on a CPU. A non-transform time-domain approach outperformed the comparable CPU calculation by a factor of 1.6 to 15. When computing the QR decomposition of a complex matrix, GPU computations are consistently 2.5 times faster than the CPU. All of these parallel algorithms were written in NVIDIA's Compute Unified Device Architecture (CUDA), a C interface that provides quick, effective parallelization.

Hardware and Software

The NVIDIA 8800 GTX video card has 16 multiprocessors, each composed of 8 SIMD processors operating at 1350 Mhz [1]. Each multiprocessor has 8192 registers, a 16KB parallel data cache of fast “shared memory,” and access to 768 MB of GDDR3 “global memory.” The card is used most efficiently in a data-parallel fashion, when the ratio of computations to memory access is high and when many computations are performed concurrently.

Table 1: FIR Test Parameters

	1	2	3	4	5
N	4096	1024	4096	4096	32768
K	128	12	4096	128	4096
M	64	20	128	512	128

Table 2: FIR Frequency-Domain Test Results

Test	1	2	3	4	5
CPU time (s)	0.12	0.0080	0.24	0.95	2.4
GPU series calculation (ratio)	1.8	0.50	1.6	1.8	16
GPU series total (ratio)	1.5	0.43	1.3	1.5	9.2
GPU parallel calculation (ratio)	26	4.4	35	24	17
GPU parallel total (ratio)	14	3.3	13	13	12

Table 3: FIR Time-Domain Test Results

Test	1	2	3	4	5
CPU time (s)	0.71	0.0052	45	5.6	375
GPU total (ratio)	6.9	1.6	7.4	7.2	15

To conserve space, only CPU time and the ratio of CPU time to GPU time are reported. The GPU performs faster than the CPU when the reported ratio exceeds one.

The benchmarks were implemented using NVIDIA’s CUDA SDK, which is a collection of C extensions and a runtime library. CUDA’s functionality primarily allows a developer to write C functions to be executed on the GPU. CUDA also includes memory management and execution configuration; with CUDA, a developer can control the number of GPU processors and threads that are to be invoked during a function’s execution.

The test system, running Gentoo Linux, contained a dual-core Athlon-64 4200+ running at 2210 Mhz, 2GB of memory, and a PCI Express x16 bus. Each Athlon core had 128KB of Level 1 cache and 512KB of Level 2 cache. The code was compiled with gcc v4.0.4 and nvcc v0.2.1221.

Finite Impulse Response: Benchmark Overview

The FIR benchmark models a set of M filters which operate on a set of M distinct input signals of length N. Each filter has K coefficients. Signal and filter elements are complex, single-precision floating point numbers. The output of filter $m \in \{0, 1, ..., M-1\}$ is given by convolution:

$$\sum_{k=0}^{K-1} x_m[i-k]\, w_m[k] \text{ for } i = 0, 1, \ldots, N-1 \tag{1}$$

A *time-domain* implementation of the FIR filter computes this convolution directly and uses 8*M*N*K floating-point operations. A *frequency-domain* approach is often preferred since convolution in the time domain is multiplication in the frequency domain, and consequently the computation time does not depend on filter size. This approach computes the FFT of the signal and filter, multiplies the transformed signal and filter, then inverts the transformation. It requires $M(10*N*\log_2 N + 8*N)$ operations [2].

Five sets of test data were chosen, with the first two taken from the HPEC benchmark; the parameters are in Table 1.

The FIR filter was implemented on the card in three ways. The first, a series approach, performs the frequency-domain task one signal at a time, using the card’s FFT capability as accessed through NVIDIA’s CUFFT library. The second, a parallel approach, uses the NVIDIA CUFFT library’s “batch mode” to directly process all the signals at once. The third, a time-domain filter, performs the convolution directly; it uses NVIDIA's CUBLAS library to dispatch each of the required convolution calculations as a Level 1 BLAS caxpy operation.

FIR Results

For the CPU, the time recorded in Tables 2 and 3 is that reported by the HPEC benchmark. For the GPU, the “total ratio” reported in Tables 2 and 3 accounts for both GPU computation time and the cost of moving the input and output data between the host computer and the card. The time required to perform the computations alone, with the required data already placed in the card's global memory, is reported as the “calculation ratio” in Table 2.

Parallel computations consistently outperform the CPU, although by a smaller factor in test 2, the short filter case, where most of the data fits in CPU cache. Series GPU computations underperform the CPU only in test 2. In test 5, where the problem size is very large, the series algorithm's calculations take only slightly longer than the parallel algorithm, though the total series time remains somewhat slower than the total parallel time.

For the time-domain benchmark, Table 3 reports CPU time and the ratio of total CPU time to GPU time[1]. GPU time-domain computation significantly outperforms the CPU on large data sets and it is competitive on the short filter test case. It is not competitive with GPU frequency-domain filtering. In limited testing, GPU time-domain filtering outperforms the parallel GPU frequency-domain filter with large M and N and small K, for instance by a factor of three with N=4096, K=12, M=1000.

QR Decomposition: Benchmark Overview

In the QR benchmark, an *mxn* matrix A is factorized into an *mxm* unitary matrix Q and an upper triangular matrix R. The matrices A, Q, and R contain complex, single-precision floating point numbers. The following properties hold after QR factorization:

$$A = QR \,; Q^H Q = I \tag{2}$$

The HPEC reference implementation performs QR via Givens rotations. A Givens rotation selectively zeroes an element of the target matrix A by updating two of its rows. In the Fast Givens QR algorithm [3], the rotations necessary to triangularize A into R are directly computed into Q.

The parallel approach used on the GPU employs Givens rotations in the standard Sameh-Kuck concurrency strategy [4]. This pattern concurrently zeroes elements that are a knight's move apart; see figure 1 of [4].

Seven sets of test data are parametrized in Table 4; the first three sets follow the HPEC QR benchmark.

Table 4: QR Test Parameters

	1	2	3	4	5	6	7
M	500	180	150	1000	1000	2000	2000
N	100	60	150	500	1000	1000	2000

Table 5: QR Test Results

Test	1	2	3	4	5	6	7
CPU time (s.)	0.79	0.062	0.087	26	65	120	209
GPU total (ratio)	1.7	0.90	0.81	2.6	2.6	2.4	2.6

CPU time and the ratio of CPU time to GPU time are reported. GPU time includes the cost of copying memory between card and host; this is less than 1% of the total time.

[1]In each time-domain test, total GPU time was no more than 7% greater than calculation time.

QR Results

On data sets much larger than the CPU's L1 cache, the GPU consistently outperforms the CPU by a factor of about 2.5.

Performance asymptotically twice as fast as Sameh-Kuck should be attainable via pipelining [5]. However, this approach was not implemented; the graphics card provides limited thread synchronization and no atomicity[2], such that an effective pipelining approach would reduce the ratio of computations to memory access.

CUDA Programmability

The CUDA paradigm, in which the GPU is a SIMD processor array, makes an efficient tradeoff between general-purpose and specialized computation. A single line of code invokes a device function with a specified thread organization. The card tightly interleaves these threads’ computations and memory accesses. There is little complexity overhead: the GPU benchmarks have roughly as many lines of code as their HPEC Challenge counterparts.

Data-parallel design for CUDA follows well-understood SIMD patterns. The key challenges of the architecture are structuring expensive memory access calls appropriately and avoiding complicated synchronization requirements.

The two NVIDIA-supplied libraries – CUFFT, for fast Fourier transforms, and CUBLAS, a set of basic linear algebra subroutines – were readily adapted to FIR and QR respectively. CUFFT peak performance of 35 Gflop/s was measured with direct tests, but these frequency-domain results represent at best 10 Gflop/s performance. CUBLAS is incomplete; the library implements some operations, such as Givens rotations, for real numbers but not for complex numbers. These shortcomings suggest avenues for future exploration. Nevertheless, these libraries are, like CUDA itself, tremendously convenient; they easily, effectively exploit the GPU’s parallel capabilities.

References

[1] NVIDIA Corporation, “NVIDIA CUDA Compute Unified Device Architecture Programming Guide”, Version 1.0, 23 June 2007.

[2] J. Lebak, A. Reuther, and E. Wong, “Polymorphous Computing Architecture (PCA) Kernel-Level Benchmarks,” MIT Lincoln Laboratory project report PCA-KERNEL-1, 13 June 2005.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations,* Third Edition, Johns Hopkins University Press, 1996.

[4] A. H. Sameh and D. J. Kuck, “On Stable Parallel Linear System Solvers,” *Journal of the ACM,* Vol. 25 No. 1, Jan. 1978 p. 81-91.

[5] M. Hofmann and E. J. Kontoghiorghes, “Pipeline Givens Sequences for computing the QR decomposition on an EREW PRAM,” *Parallel Computing,* Vol. 32 No. 3, March 2006.

[2]Atomic operations on integers exist in CUDA Compute Capability 1.1 on the newer but slower GeForce 8600 and 8500 cards. Atomic operations on floating-point numbers are not supported in any existing hardware but are referenced in the high-level PTX assembler language used by the card.