GPU VSIPL: Core and Beyond

Andrew Kerr, Dan Campbell, Mark Richards
Georgia Institute of Technology, Georgia Tech Research Institute
arkerr@gatech.edu, dan.campbell@gtri.gatech.edu, mark.richards@ece.gatech.edu

***Abstract*—GPU VSIPL[1] is an implementation of the Vector Signal Image Processing Library (VSIPL) and provides a library-based solution to developing GPU-accelerated numerical applications. GPU VSIPL now supports much of the VSIPL Core Profile as well as element-wise matrix operators corresponding to required vector operators. This is nearly a five-fold increase in function count since GPU VSIPL was first introduced and includes numerous high-level numerical procedures. GPU VSIPL is implemented by optimized kernels written in the CUDA programming language. It provides support for real and complex matrix and vector operations; signal processing operations such as FIR, FFT, correlation, and histogram; and linear algebra operations such as matrix product and QR decomposition. GPU VSIPL achieves performance comparable to CUBLAS without encumbering the developer with CUDA-specific function calls and is link-compatible with existing VSIPL applications.**

Introduction

High-performance architectures such as graphics processing units (GPUs) require a specialized approach to programming that explicitly exposes data parallelism within kernels. Languages such as NVIDIA's CUDA [1] expose thread and memory hierarchies to the programmer enabling efficient utilization of the hardware implementing the CUDA execution model, yet CUDA programs are not easily portable to architectures beyond NVIDIA GPUs. The development of efficient CUDA kernels is a time-consuming process requiring intimate knowledge of the underlying architecture.

Vector Signal Image Processing Library (VSIPL) [2] is a signal processing and linear algebra API resulting from a DARPA initiative to provide high-performance numerical processing while maintaining platform independence over a wide range of system architectures. Its abstractions for weakly consistent memory are well-adapted to systems containing GPUs in which transfer costs via the PCI Express bus need not be incurred between every function call.

In this paper, we discuss developmental progress of GPU VSIPL [3], an implementation of the VSIPL API. We present features added to GPU VSIPL since its introduction [4] and the measured performance of several high-level operations, relative to CUBLAS [5] where applicable. Additionally, we present the impact of several optimization techniques relevant to GPU computing.

Background

VSIPL explicitly decouples numerical elements from mathematical objects composed of this data. A *block* is a contiguous array of real or complex scalar elements, integer or floating point, that may be managed by the VSIPL implementation. Application data may be bound to a VSIPL block which may make an internal copy with arbitrary structure. The application must *admit* the block to VSIPL before performing VSIPL operations on this data thereby enabling GPU VSIPL to synchronize blocks in the GPU's memory.

The VSIPL API contains thousands of functions, many of which implement highly-specialized operations only needed by a small subset of potential users. The VSIPL specification includes two formally defined subsets of functions in the VSIPL API to facilitate the production of VSIPL implementations requiring only the most common functionality. The smallest of these formal subsets is known as VSIPL Core Lite [6]. It includes floating-point vector operations, FIR filtering, and FFT. VSIPL Core [7] is a superset of VSIPL Core Lite and includes matrix views, linear algebra operators, 1-D correlation, and matrix decompositions. VSIPL implementations are link-compatible with C.

GPU VSIPL

GPU VSIPL implements nearly all of VSIPL Core profile with the exception of LU, Cholesky, and several special solvers. GPU VSIPL does include, however, VSIPL functions not required in Core profile. These are element-wise matrix operators corresponding to the vector operators specified by Core profile. Table I characterizes functionality supported by GPU VSIPL. A complete list of supported functions may be found on the GPU VSIPL website *http://gpu-vsipl.gtri.gatech.edu* as well as the list of functions defined in VSIPL Core that are *not* supported at this time. GPU VSIPL passes all tests in the VSIPL Core Lite Test Suite as well as additional testing of VSIPL Core functions on matrix views with arbitrary strides, offsets, and padding. GPU VSIPL is currently available as a non-redistributable static library compiled for Linux, Windows, and Mac OS X. More information is available on the GPU VSIPL website.

TABLE I: GPU VSIPL Functionality Summary

	GPU VSIPL
Data types	real, complex, integer, boolean, index
View types	matrix, vector
Element-wise operators	arithmetic, trigonometric, transcendental, scatter and gather, comparators
Signal processing	FFT (in-place, out-of-place, batched), 1D correlation, window creation, FIR filter, random number generation (uniform, normal), histogram
Linear algebra	generalized matrix product, QR decomposition, least-squares solver

[1]This work was supported in part by DARPA and AFRL under contracts FA8750-06-1-0012 and FA8650-07-C-7724. The opinions expressed are those of the authors.

Optimization Techniques

GPU VSIPL is a performance-oriented library, and efficient mapping of kernels to the architecture has been of paramount importance. GPU VSIPL kernels that exhibit high utilization of GPUs have the following characteristics: high-intensity of floating-point instructions in the inner loop, coalesced global memory accesses, efficient data sharing among threads of the kernel, minimal divergent control flow, and synchronizations placed to achieve high SIMD utilization. Vector and matrix element-wise operators tend to be memory bound with inefficient computations effectively hidden by transfers to global memory. Higher-level linear algebra operators such as matrix product can be implemented by compute-bound kernels. Matrix product is implemented by a "fat kernel" with two uniform control paths depending on whether the CUDA block corresponds to elements on the fringe of the matrix requiring fine-grain guard conditionals or the core of the matrix in which guards are not required and loops may be heavily unrolled. GPU VSIPL kernels map elements of matrix views to threads with IDs equal to their word offset from 256-byte boundaries ensuring coalescable transactions to global memory.

QR decomposition is an $O(N^3)$ computation fundamental to many linear algebra and signal processing algorithms. Householder reflections are a common parallelizable method for QR but depend heavily on matrix-vector products which are typically memory-bound on high-performance architectures such as GPUs. In [8], we present a CUDA implementation of a blocked Householder QR algorithm in which Householder reflections are applied in batches by matrix product operations. This implementation of QR decomposition has been integrated into GPU VSIPL.

Performance

In this section, we present measured performance of GPU VSIPL's support for generalized matrix-vector product and QR decomposition. Performance results for element-wise operators are presented in [4]. These measurements have been performed on a test platform with a 2.83 GHz Intel Core2 "Penryn" CPU running Linux x86-64 with both a GeForce GTX280 and a GeForce 9800 GX2 GPU. Figure 1 illustrates the performance of GPU VSIPL's matrix-vector product in GFLOP/s. As a memory bound operation, matrix-vector product is limited by the bandwidth of the target device. GPU VSIPL's matrix-vector product achieves a maximum of 97% of the theoretical maximum performance on the GeForce GTX280. This function has less overhead than `cublasSgemv` from the CUBLAS library as illustrated by its rapid approach to the theoretical limit. Figure 2 illustrates the performance of QR decomposition on real data in terms of GFLOP/s for rectangular matrices. Performance measurements include calls to `vsip_qrdprod_f` which are themselves compute bound and achieve approximately 300 GFLOP/s sustained across matrices with 2048 rows or more. GPU VSIPL's QR exhibits a peak performance of 142 GFLOP/s, more than a $4.5\times$ speedup over a single-threaded call to Intel's Math Kernel Library (MKL) and a 20% speedup over MKL launched with four threads.

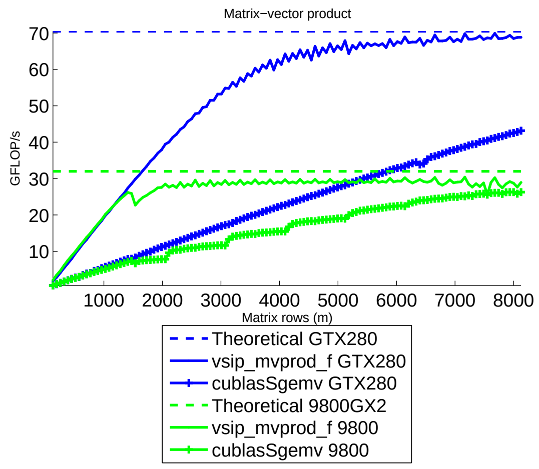

Figure 1: Performance of GPU VSIPL's vsip_mvprod_f().

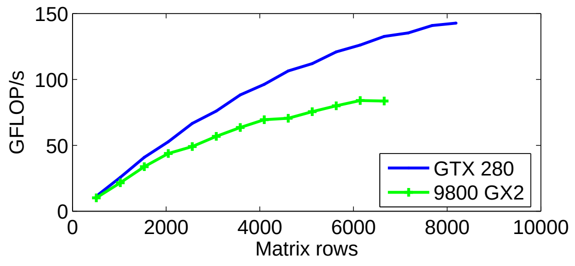

Figure 2: Performance of GPU VSIPL's QR decomposition.

Conclusions

GPUs are specialized high-performance architectures commoditized by the 3D gaming market. Their programming model is also specialized resulting in increased development costs and sacrificing portability to an extent. GPU VSIPL is a high-performance library-based approach to developing applications that are accelerated by GPUs but decoupled from its programming model. GPU VSIPL achieves performance comparable to NVIDIA's CUBLAS library yet includes numerous signal processing operations making it a worthwhile choice for GPU-accelerated application development.

References

[1] NVIDIA, *NVIDIA CUDA Compute Unified Device Architecture*, 2nd ed., NVIDIA Corporation, Santa Clara, California, October 2008.

[2] D. Schwartz, R. Judd, W. Harrod, and D. Manley, "VSIPL 1.3 API," VSIPL Forum, Tech. Rep., 2008.

[3] Georgia Institute of Technology and Georgia Tech Research Institute, "GPU VSIPL," http://gpu-vsipl.gtri.gatech.edu.

[4] A. Kerr, D. Campbell, and M. Richards, "GPU VSIPL: High-Performance VSIPL Implementation for GPUs," in *HPEC 2008*. Lexington, MA, USA: MIT Lincoln Laboratory, 2008, p. 123.

[5] NVIDIA, *NVIDIA CUDA CUBLAS Library*, 2nd ed., NVIDIA Corporation, Santa Clara, California, September 2008.

[6] D. Schwartz, R. Judd, W. Harrod, and D. Manley, "VSIPL Core Lite Profile," VSIPL Forum, Tech. Rep., 2000.

[7] ——, "VSIPL Core Profile," VSIPL Forum, Tech. Rep., 2000.

[8] A. Kerr, D. Campbell, and M. Richards, "QR decomposition on GPUs," in *GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*. New York, NY, USA: ACM, 2009, pp. 71–78.