AUTOMATIC TRANSLATION TO GPGPU
A CONCEPT PAPER SUBMITTED TO THE SCHOOL OF COMPUTING AND INFORMATICS TECHNOLOGY

0.1 Introduction

There has been increasing interest in using GPGPU based systems in the school and NVidia/CUDA seems to have become the default environment for this and this is where most of our experience lies. A general-purpose GPU (GPGPU) is a graphics processing unit (GPU) that performs non-specialized calculations that would typically be conducted by the CPU (central processing unit). Ordinarily, the GPU is dedicated to graphics rendering. GPGPU which is General-purpose computing on GPUs only became practical and popular after about 2001, with the advent of both programmable shade and floating point support on graphics processors. Notably, problems involving matrices and/or vectors especially two, three, or four dimensional vectors were easy to translate to a GPU, which acts with native speed and support on those types. The scientific computing community's experiments with the new hardware began with a matrix multiplication routine (2001). One of the first common scientific programs to run faster on GPUs than CPUs was an implementation of LU factorization (2005).

0.2 Background to the problem.

As both CPU and GPU become employed in a wide range of applications, it has been acknowledged that both of these processing units (PUs) have their unique features and strengths and hence, CPU-GPU collaboration is inevitable to achieve high-performance computing. This has motivated significant amount of research on heterogeneous computing techniques, along with the design of CPU-GPU fused chips. These early efforts to use GPUs as general-purpose processors required reformulating computational problems in terms of graphics primitives, as supported by the two major APIs for graphics processors, OpenGL and DirectX. This cumbersome translation was obviated by the advent of general-purpose programming languages and APIs such as Sh/Rapid Mind, Brook and Accelerator. This means that modern GPGPU pipelines can leverage the speed of a GPU without requiring full and explicit conversion of the data to a graphical form.

0.3 Problem Statement

There has been a problem of parallel processing multiple evaluation of a sequential task on large data sets for example when you have code involving lots of large (especially nested) loops, it became a must to calls for parallelization of that peace of code anyway. If it is a code that executes within the loops then it becomes a must for it to be reduced to a fairly simple algorithm.Putting that a side, there is also a problem were you have a large dataset that fits, or needs to be made to fit into the memory structures of the hardware you have. If your code involves manipulating large matrices, all these above problems makes a call and need of GPGPU for them to be solved.

0.4 Main Objective

To develop a high-performance computing system that can do automatic translation to general purpose computing on an FFT mapping on GPU for radar processing application.

0.4.1 Specific Objectives

To collect and analyze the requirements about the relevance and feasibility of the Automatic translation to GPGPU on an FFT mapping on GPU for radar processing application.

To implement the FFT mapping on GPU for radar processing application.
To test and validate the application.

0.5 Research Scope

This research is based on Efficient FFT mapping on GPU for radar processing application. General-purpose multiprocessors (as, in our case, Intel Ivy Bridge and Intel Haswell) increasingly add GPU computing power to the former multicore architectures. When used for embedded applications (for us, Synthetic aperture radar) with intensive signal processing requirements, they must constantly compute convolution algorithms, such as the famous Fast Fourier Transform. Due to its "fractal" nature (the typical butterfly shape, with larger FFTs defined as combination of smaller ones with auxiliary data array transpose functions), one can hope to compute analytically the size of the largest FFT that can be performed locally on an elementary GPU compute block. Then, the full application must be organized around this given building block size. Now, due to phenomena involved in the data transfers between various memory levels across CPUs and GPUs, the optimality of such a scheme is only loosely predictable (as communications tend to overcome in time the complexity of computations). Therefore a mix of (theoretical) analytic approach and (practical) runtime validation is here needed. This occurs at both stage, first at the level of deciding on a given elementary FFT block size, then at the full application level.

0.6 References

Du, Peng; Weber, Rick; Luszczek, Piotr; Tomov, Stanimire; Peterson, Gregory; Dongarra, Jack (2012). "From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming". Parallel Computing. 38 (8): 391-407. doi:10.1016/j.parco.2011.10.002.

Tarditi, David; Puri, Sidd; Oglesby, Jose (2006). "Accelerator: using data parallelism to program GPUs for general-purpose uses". ACM SIGARCH Computer Architecture News.

Jump up Che, Shuai; Boyer, Michael; Meng, Jiayuan; Tarjan, D.; Sheaffer, Jeremy W.; Skadron, Kevin (2008). "A performance study of general-purpose applications on graphics processors using CUDA". J. Parallel and Distributed Computing. 68 (10): 1370- 1380. doi:10.1016/j.jpdc.2008.05.014.

Mohamed Amine Bergach, Emilien Kofman, Robert de Simone, Serge Tissot, Michel Syska. Efficient FFT mapping on GPU for radar processing application: modeling and implementation. arXiv:1505.08067