A CONCEPT PAPER ABOUT AUTOMATIC TRANSLATION TO GPGPU.

1 Introduction.

GPGPU (general purpose computing on graphics processing units) is a methodology for high-performance computing that uses graphics processing units(GPU) to perform computation in applications traditionally handled by the central processing unit(CPU). The use of multiple video cards in one computer, or large numbers of graphics chips, further parallelizes the already parallel nature of graphics processing. In addition, even a single GPU-CPU framework provides advantages that multiple CPUs on their own do not offer due to the specialization in each chip.

Essentially, a GPGPU pipeline is a kind of parallel processing between one or more GPUs and CPUs that analyzes data as if it were in image or other graphic form. While GPUs operate at lower frequencies, they typically have many times the number of cores . Thus, GPUs can operate on pictures and graphical data effectively far faster than a traditonal CPU. Migrating data into graphical form and then using the GPU to scan and analyze it can result in profound speedup.

2 Background to the problem.

As problems involving matrices and vectors especially two, three, or four dimensional vectors became easy to translate to a GPU, which acts with with native speed and support, the development of GPGPU proved to be important. 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(Application Programming Interfaces) for graphics processors, OpenGL and DirectX. This cumbersome translation was obviated by the advent of general-purpose programming languages and APIs . 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.

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 piece 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

4 Objectives.

4.1 Main objective.

To develop algorithms that are data parallel and throughput intensive to suit well to GPGPU implemention.

4.2 Other objectives.

To increase on the speed of execution of commands other than CPUs which are slow in their operations.

To make a significant impact on high-performance computing in a wide range of application domains.

To achieve performance goals by combining relatively low-cost, multi-core nodes with a high-speed interconnect for explicit message passing.

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

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

5 Research scope

This research is based on Efficient FFT mapping on GPU for radar processing application. Generalpurpose 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.

6 References.

Byungil Jeong and Greg Abram at Texas Advanced Computing Center(TACC). Faith Singer-Villalobos, Public Relations, faith@tacc.utexas.edu, 512.232.5771

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