15-740 Project Milestone Report

Xi Liu, Fan Guo

1 Accomplished So Far

We have made a correct implementation of channel filter, which is essentially the process of FFT⇒filtering⇒IFFT on BrookGPU platform (`http://graphics.stanford.edu/projects/brookgpu/`), which provides a better programmability for general purpose GPU. We compare the performance of CPU backend and GPU/DX9 backend.

2 Difficulties on Meeting the Milestone

Our progress till now is behind the milestone and a major goal change of the project is listed in the next section. The difficulties mainly arises from limited resources, i.e. finding an appropriate programming language and compiler for the ATI Radeon X600 Graphics Card (the one installed on the graduate student machine).

The ideal choices for the tasks presented in the project proposal would be the NVIDIA CUDA toolkit (`http://developer.nvidia.com/object/cuda.html`), which exposes more underlying architecture to programmers. However, the package requires a capable NVIDIA video card, and we could not get for this project. ATI also designed a similar platform "Close-to-Metal (CTM) Device" (`http://ati.de/companyinfo/researcher/documents/ATI_CTM_Guide.pdf`) for ATI devices. But to date, there is no release accessible online yet.

Then we come to a recent release of Microsoft Research Accelerator system (`ftp://ftp.research.microsoft.com/pub/tr/TR-2005-184.pdf`) which includes a dot net based data parallel library for GPUs. It has produced run-time error (failed to initialize GPU) under the Windows OS. We could not find any solution to this problem. The developers hint that the problem may due to the x64 architecture, but changing the target CPU from any CPU to x86, as they suggested, does not help in our case.

So finally, we found an appropriate package from Stanford which already has a number of users in the research community. And we also had a successful test under Windows using cygwin (We could not get it to compile under Linux, and there is little support for that). On the other hand, BrookGPU is still rather limited: it provide a relatively high-level programmability solution but not an efficient one. The source code written in BrookGPU will be converted to Cg code first and compiled to executable.

3 Major Goal Changes and Revised Schedule

Our previous goal is to port the OFDM processing blocks of GNURadio project to GPU. Now our goal is to program the OFDM to GPU as a whole. There are mainly two reasons for this change.

First, in our experiments on GNURadio, we find that the performance of GNURadio is really bad. For example, the turnaround time can be as long as 3ms, i.e. it takes about 3ms for the device to initiate transmitting a packet after it discovers the channel is clear to send. Since there is little data processing involved in this turnaround process, we believe that the major bottleneck of GNURadio is in its modularity. The GNURadio is designed mainly for programmability and flexibility, where processing blocks (e.g. FFT) are implemented in C++ and the applications (e.g. OFDM) are implemented in Python that connect various blocks. Thus we would like to implement OFDM functionality as a whole on GPU instead of implementing different processing blocks in GPU and connecting them together in CPU, which may involve extensive copying between main memory and video memory.

Second, after understanding some techniques of GPGPU programming, we believe that directly porting the CPU code is not going to work very well. For example, most FFT implementation on CPU uses a recursive method, but on GPU, they are implemented in a iterative fashion, e.g. GPUFFTW project vs. FFTW project. Thus we would like to implement OFDM in GPU from scratch.

We also find that the performance of BrookGPU generated code is not very good for our purpose. The BrookGPU API is best suited for pure data parallel programs, but the OFDM implementation is more complicated. Thus we will also look for alternatives. One possibility is to program directly using Cg compiler.

As a result, we will focus on implementation, preferably using Cg, for the rest two weeks.

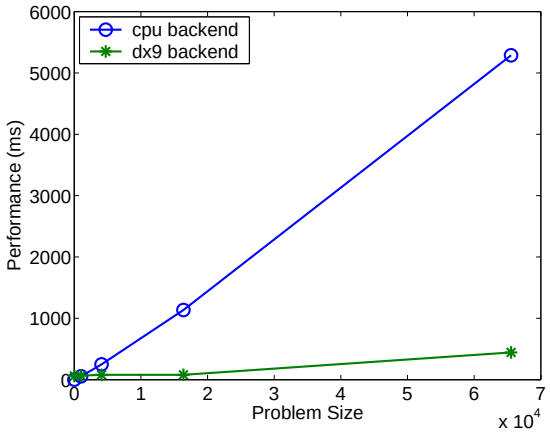

Figure 1: Execution time of channel filter

4 Preliminary Results

Figure 1 shows the execution time of the channel filter and Figure 2 shows the speedup. Note that we test the same program using GPU vs. CPU backend, instead of comparing GPU against an optimized version for CPU. Thus, the observed speedup is as expected, but the execution time is very long. Also, we observed a bad initialization performance in BrookGPU (which is not included in the execution time), and this is undesirable for our purpose because the problem size may not be as large to effectively amortize the initialization cost.

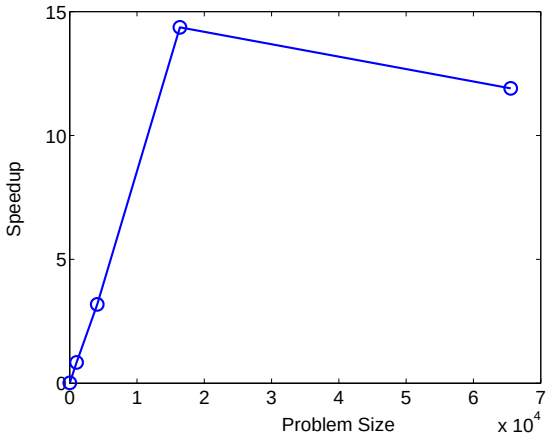

Figure 2: Speedup of gpu/dx9 backend over cpu backedn