Part II Project Proposal

GPU Acceleration of the Ypnos Programming Language

S. Pattuzzi, Robinson College

Originator: D. Orchard

9^{th} of October, 2012

Project Supervisor: D. Orchard

Director of Studies: Dr A. Beresford

Project Overseers: Dr A. Madhavapeddy & Dr M. Kuhn

Introduction, The Problem To Be Addressed

In recent years, Moore's law has begin to plateau. As a result the hardware industry has increasingly been turning to MIMD and SIMD as a solution. Some of the most high performing SIMD implementations today can be found in GPUs and can be harnessed via GPGPU languages such as CUDA and OpenCL. However, taking advantage of this hard as it requires knowledge of the low-level concept of these languages. It is also not portable between different hardware and methods of concurrency.

Stencil computation
What is Ypnos
How it works
Compile Ypnos to GPU

Starting Point

I have had experience of functional programming from the course as well as having followed complete some Haskell tutorials. More familiarity with Haskell will need to be developed before starting this project in earnest.

During the course of an 11 week internship I was in charge of my own project. These skill will be transferable to the planning, implementing, documenting and testing needed in this project.

I have already read the Ypnos paper and familiar with the constructs of the language as well its primitives.

At present Ypnos has been partially implemented in a single threaded fashion on the CPU. The proof-of-concept leaves many primitives including some that would benefit greatly from the pipelining a GPU can provide. This implementation can be taken as both the starting point and the benchmark for the new implementation.

A Haskell ESDL already exists for compiling array computations to CUDA code. The library takes an AST and produces code to run on the GPU. We will use this library as a back-end to avoid writing a compiler to CUDA code directly.

Resources Required

For this project I shall require my own laptop computer that runs Arch Linux for the bulk of development work. Backup will be to github, the SRCF and/or the MCS. Should my computer fail I will be able to use the MCS computers for the project.

I require an Nvidia GPU in order to test the code produced. This will be provided by Dominic Orchard and I will have access to the machine via SSH for testing purposes.

Work to be done

The project breaks down into the following sub-projects:

1. The implementation of the main compilation. This involves writing a compilation pass that can take the Ypnos AST and produce a correspondent "accelerate" AST.

2. The implementation of the basic "run" primitive. The will possible be written in the "accelerate" language directly.

3. The testing of the implementation to check that it works correctly and is faster than the original. This will need a test bench to be constructed

that includes various well know stencil computations. E.g. the Laplace transform and the Gaussian blur filter.

Success Criterion for the Main Result

The project will be a success if it can take Ypnos code and run it using the "run" primitive on the GPU. This should also run faster than the current single threaded implementation on large work loads—taking into account the time required to copy data on and off the GPU.

Possible Extensions

If the main aim for this project is achieved then I shall try to implement further primitives of the Ypnos language. The programmer will then be able to take advantage of the speed gains of the GPU pipeline. I will attempt them in this order:

1. The "iterate" primitive that allows the programmer to specify a stopping condition and iterate "run" until it is achieved. This will eliminate the need to copy data between the CPU and GPU at each step.

2. The "reduce" primitive that is similar to the "iterate" primitive in the gains achieved. However, it allows us to achieve functions such as max, min, sum or mean of a grid. This is a useful stopping condition for the "iterate" primitive.

3. The "zip" primitive that will allow the programmer to use the values from multiple grids in the calculation of the next. This is particularly useful in scientific applications where we are calculating multiple quantities (force, direction, speed) from multiple variables.

Timetable: Workplan and Milestones to be achieved.

Planned starting date is 19/10/2011 when the proposal is accepted.

1. **Michaelmas weeks 2-4** Learn to write and read Haskell code. Write some programs in Ypnos. Try to understand the existing code base.

2. **Michaelmas weeks 5-6** Get familiar with the "accelerate" ESDL by reading the paper and writing some toy programs.

3. **Michaelmas weeks 7-8** Start implementation of the compiler from Ypnos to "accelerate".

4. **Michaelmas vacation** Finish the compiler and begin work on implementing the run primitive.

5. **Lent weeks 0-2** Finish the run primitive if necessary. Write the progress report. Start work on the basic test bench.

6. **Lent weeks 3-5** Finish main test bench and run experiments. Make improvement to the code as necessary to achieve the main aim of the project.

7. **Lent weeks 6-8** If there is time then the main extensions may be implemented at this point.

8. **Easter vacation:** Write the main chapters of the dissertation.

9. **Easter term 0-2:** Elaborate on the existing tests bench and run final experiments. Complete the dissertation.

10. **Easter term 3:** Proof reading and then an early submission.