Imagine Discussion and StreamC/KernelC Introduction

Lecture #4: Tuesday, 16th April 2002
Lecturer: Bill Dally, Mattan Erez
Scribe: Henry Fu (hwfu@stanford.edu), Harn Hua Ng (harnhua@stanford.edu)
Reviewer: Mattan Erez

In this session, we went through an example stream program written in StreamC and KernelC. We then tested the demo program using IDebug and ISim. We also explored various scheduling directives in StreamC and KernelC and examined the resulting schedules using the Schedule Visualizer.

1 Logistics

Here are the class announcements for today's session:

- Get class software from Packard 124 or other campus resources.
- Tools work on Microsoft Windows and Microsoft Visual Studio 6.0.
- Download tools distribution from the class webpage.
- Go through the tool flow described in the Beginner's Guide.

2 Stream Program Demo

The example stream program can be found in im_apps/demo of the ee482c tools installation at http://cva.stanford.edu/ee482c/downloads.html

2.1 Characteristics of a StreamC program

StreamC and KernelC are the two programming languages used in creating Imagine applications. They are essentially adaptions of C++, and we will use them with the Microsoft Visual C++ development tools. A couple of points to note when creating Imagine applications are:

- StreamC and KernelC are similar to C++, e. g. `#define`, `macros`, `extern` definitions can be used.
- The usual method of data I/O into the application is via text files for data input and output.

2.2 General steps to create a StreamC program

The logical flow of data and control in a StreamC program is as follows:

- Read input arguments from stdin.
- Declare stream variables and allocate space for them.
- Write StreamC code to define the comunication among all the kernels, and decide which kernel to call.
- Write KernelC code to describe the function of each kernel.
- Write output to stdout or to a text file for comparison checks.

2.3 General steps to create a KernelC program

A KernelC program is usually a loop which processes the elements in a stream. Due to the requirement for parallelism, conditionals are extremely hard to implement in kernel functions.

2.4 Setting up variables and data for an Imagine application

One of the first things to do in creating an Imagine application is to declare and define the data variables. There are also types which are specially defined for Imagine applications, such as the ”im_float” type. Some things to take note of are:

- Declare data types in a header file. There is a particular format to adhere to - see header file in example application.
- Use ”im_float” instead of ”float” when declaring floating point data in StreamC in order to let compiler assign values to all eight arithmetic clusters.
- Use ”uc” prefix when declaring microcode variables for kernels.
- Take a look at sample data_input files to learn how to create sample data.

3 Simulation and Tools

The two simulators we will use to run Imagine applications are IDebug and ISim. However, kernel microcode must be produced before simulation, using IScd. To see the results of kernel scheduling, Schedviz is the provided visualization tool. A brief overview of these tools are as follows:

- IDebug - Functional simulator which compiles, runs and debugs in Visual Studio.
- ISim - In-house developed simulator for Imagine applications. It is both cycle and output accurate.
- IScd - In-house developed utility to schedule kernel microcode.
- Schedviz - In-house developed tool to evaluate the results of kernel scheduling, in terms of processor resource utilization and performance estimates.

4 Optimization

Optimization is an important aspect of Imagine applications, and besides exploiting what we know about the architecture of the Imagine processor, we can make use of the following techniques provided by the tools:

- Use ”loop_count” instead of ”loop_stream” if the actual length of the stream is known.
- Use ”pipeline” option to make the compiler adopt software pipelining techniques (in kernel code).
- Use ”unroll” option to make the compiler do loop unrolling (in kernel code).
- Use ”schedviz” after scheduling kernels to view the utilization and performance estimates of the kernel code in the Imagine processor.
- If the tools do not schedule the code well, it may be necessary to optimize the code manually, as shown in the demo example in class.

5 Miscellaneous Notes

This section covers some of the nuances of programming Imagine applications that were discovered through prior experience. They range from debugging methods to the behavior of the simulator in special circumstances.

- Arguments are passed by reference into kernels.
- ”float” type can be used in KernelC code.
- Suggested debugging techniques: Dump streams or the contents of the components (e.g. the SRF) to files and examine the output.
- Code that works correctly in IDebug doesn’t necessarily work in ISim, e.g. loop unrolling with incomplete iterations will run in IDebug, but will loop endlessly in ISim.
- Ensure that the simulator receives the exact amount of data that it is expecting, otherwise the output might be wrong.
- Remember to use ”stream”, instead of ”array” when declaring vectors.
- In general, StreamC code runs on the host processor; KernelC code runs on simulated Imagine processor (which acts as a co-processor).
- If a stream program works in IDebug but doesn’t work in ISim, try to dump the contents of the SRF in ISim. This will solve many ISim problems.
- Sometimes, compiler errors are hard to understand. It is best to compare your code against working code and try to find out what’s the difference that prevents your code from compiling.
- Always assume the host processor can keep up with the Imagine processor. It means that it takes zero time for the host processor to issue commands and to provide data to the Imagine processor.

6 Questions and Answers

- *Why do the developers of Imagine need to use StreamC and KernelC to implement stream program? Why can't the developers create an Imagine library on top of C++ to achieve this function?*
  If the stream program is implemented using a C++ library, it makes the stream program execute instructions sequentially, one by one, but in fact, the Imagine processor supports dynamic scheduling and executes instructions in parallel. Also, running on top of C++ hinders the benefits of consumer-producer locality.

- *If the stream processor is efficient working with streams of data, why can't it work with streams of characters such as a program that does "diff" or "lexicographical compare"?*
  Streams of characters do not obey the fundamental property that is required by stream processors - they are not record independent. It is very challenging to implement such programs because each record cannot be processed in parallel.

7 Summary

In this session, we ran a demo of a Stream program through the tools provided and discussed the general methods for creating, debugging and optimizing stream programs. Students are encouraged to get familarized with the tools and working environment as soon as possible, so as to get started on the first assignment.