Performance of Alternative Topologies for Solving Laplace's Equation using Auto-Pipe

Joe Wingbermuehle, wingbej@wustl.edu (A paper written under the guidance of Prof. Raj Jain)

Download 

Abstract

The Auto-Pipe system allows one to evaluate various resource mappings and topologies for streaming applications. In this paper we use Auto-Pipe to gather timing information for individual blocks in a streaming application to solve Laplace's equation. We then use the timing information to seed a queueing model to predict the performance of various topologies and resource mappings. Finally, we use the Auto-Pipe system to measure the performance of the topology to verify the model and rank the topologies and resource mappings based on throughput.

Keywords: Auto-Pipe, Laplace's Equation, Performance Analysis, Queueing Theory, Monte-Carlo Simulation, X-Language

- 1. Introduction
  - 1.1 Auto-Pipe
  - 1.2 Laplace's Equation
  - 1.3 Methods for Solving Laplace's Equation
  - 1.4 The Application
- 2. Performance Analysis Methodology
  - 2.1 Metrics
  - 2.2 Parameters
  - 2.3 Factors
  - 2.4 Experimental Design
- 3. Analytic Model
  - 3.1 The Model
  - 3.2 Block Timings
  - 3.3 Mappings
- 4. Measurement
  - 4.1 Results
  - 4.2 Analysis
  - 4.3 Interpretation
- 5. Conclusion
- References
- Acronyms

1. Introduction

We attempt to determine the best topology and resource mapping for an Auto-Pipe application to solve Laplace's equation. Auto-Pipe allows one to experiment with various topologies of the application and obtain timing information. We use the Monte-Carlo method for solving Laplace's equation since it represents a small,

but useful application that fits nicely into the streaming framework.

1.1 Auto-Pipe

Auto-Pipe is a system for creating computing pipelines across heterogeneous compute platforms [Tyson06, Franklin06, Chamberlain07]. A heterogeneous compute platform may contain a combination of CPUs, GPUs, FPGAs, or other accelerator devices. Auto-Pipe provides the user with the ability to author compute blocks to operate on a data stream. The user can then specify the topology and resource mapping of the application using Auto-Pipe's X language. This X language description is fed to the Auto-Pipe X compiler to build the application. The Auto-Pipe approach allows one to access timing information for individual blocks and to easily modify the application topology and block-to-resource mapping of the application.

1.2 Laplace's Equation

The goal of the application is to solve Laplace's equation in two dimensions (shown in figure 1.1). Laplace's equation is a second-order partial differential equation (PDE) [Strauss92]. This equation has several uses including modeling stationary diffusion (such as heat) and Brownian motion. For heat, given the temperatures at the boundaries of a two-dimensional object, solutions to Laplace's equation provide the interior temperatures after the temperatures have stabilized. The ease of solving Laplace's equation depends on the nature of the boundary conditions. An analytic solution exists for simple boundary conditions, however, for many boundaries conditions, no analytic solution exists and numerical solutions must be sought [Farlow93].

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Figure 1.1: Laplace's Equation in Two Dimensions

1.3 Methods for Solving Laplace's Equation

There are numerous ways to solve Laplace's equation. As previously stated, for certain simple boundary conditions a straight-forward analytical solution exists. However, it is often the case that a numerical method is required. One numerical approach is Gauss-Seidel iteration [Strauss92]. This method converges to the correct solution quickly, but it requires that the complete grid be stored in memory. Another method is Monte-Carlo simulation. This technique is provably correct [Reynolds65], but converges slowly. Nevertheless, this method is useful if only a few interior points are needed. This is because the Monte-Carlo method does not require storing the whole grid since the grid is implicit and only those points that are of interest need be computed. Further, Monte-Carlo methods work well in a pipelined framework such as Auto-Pipe [Singla08].

1.4 The Application

The Auto-Pipe application to solve Laplace's equation is made up of several compute blocks. We can replicate some blocks to achieve better performance via parallelism by dividing the work across multiple compute resources. All of the blocks considered here are implemented in C for a general purpose processor. However, Auto-Pipe would allow any of these to be re-implemented for an FPGA. Table 1.1 lists the blocks used in this application. Note that there are a few other blocks used to seed the random number generator and to read the boundary conditions, but these blocks don't contribute much to the running time so we don't consider them here.

Table 1.1: Blocks

Block	Description
RNG	Random number generator.
Split	Block to distribute numbers equally among two outputs.
Walk	Block to perform random walks.
AVG	Block to take the average of two inputs.
Print	Block to output the result.

The application works by generating random numbers which are then fed to a block to perform a random walk starting from the coordinates of interest. For this application we consider all grid coordinates. The random numbers determine the direction for each stage of the random walk. The walk continues until a boundary is crossed. Once the boundary is crossed, the temperature at the boundary point is forwarded to either a print block, which saves the result to a file, or an average block, which averages the results of two inputs. The application continues this process many times for each coordinate in a rectangular region. Note that in a real use of this application one would probably only be interested in few locations, but for the purpose of performance analysis, we consider the whole grid.

There are multiple ways to divide the random walks among compute resources. One method is to configure the block used to generate random numbers to feed a block that splits the output among multiple walk blocks. A second method is to replicate the random number generation blocks along with the walk blocks by using different random number seeds for each random number generation block. In either case, we feed the results from the random walk blocks to a block that averages them before we send the averages to a print block to be recorded. Given a single compute node, the topology of this application is straight-forward. However, even with only two processors, there are several possible block topologies and several resource mappings that may make sense. Our goal is to determine the best topology and resource mapping.

2. Performance Analysis Methodology

As stated above, our goal is to determine the best block topology and resource mapping. To do this, we must first define what it means to be the the best. Next, we determine what is to be measured and isolate the parameters that may affect that measurement. Finally, we select a method for evaluating the performance.

2.1 Metrics

The first step in determining the best mapping is to define the metrics of interest. We assume no application errors and that the service is always performed correctly. Given this, the performance metrics are response time, throughput, and resource utilization [Jain91]. For this application, we will focus on throughput. Throughput is defined by the number of jobs that can be completed per unit time. Here we define a job as a point whose temperature is to be evaluated.

2.2 Parameters

System parameters affecting performance include:

- CPU Type
- Number of Cores
- System Memory

Workload parameters include: