Runtime Prediction of Parallel Programs

Project Report

Submitted in partial fulfillment of the requirements of
CS F376 Design Oriented Project

By

Pathikrit Ghosh
ID No. 2014A7PS0105G

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS
May 2017

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS

Abstract

Runtime Prediction of Parallel Programs

by Pathikrit Ghosh

Programs nowadays are being increasingly parallelized and run on graphical processing units(GPUs). However, running programs on such architectures is expensive due to the energy costs. Hence, scientists find it useful to predict the energy requirements before they run their code. One part of the problem to do that is to predict the runtime of of the code, which we focus on in this project. Firstly, we extract features from a CUDA code. These features are extracted by running a model on the ptx code that CUDA supplies. The aim is to extract features that help in differentiating between each program. The features are then used to train a machine learning model and subsequently predict run-times of other CUDA programs.

Contents

List of Tables

Abbreviations

GPU	**G**raphical **P**rocessing **U**nits
ptx	**p**arallel **t**hread **e**xecution
ILP	**I**nstruction **L**evel **P**arallelism
DFG	**D**ata **F**low **G**raph
CFG	**C**ontrol **F**low **G**raph
DAG	**D**irected **A**cyclic **G**raph
RAW	**R**ead **A**fter **W**rite
SVM	**S**upport **V**ector **M**achines

Chapter 1

Introduction

GPUs are increasingly being used to accelerate scientific computing applications. However using them come at a cost. This involves massive amounts of energy and time requirements, which may not always be viable. Scientists as result ideally want to find run time of this programs without executing them, to save on these resources. This is a difficult task since many factors determining the run time are decided only at run time. However we can still find a prediction algorithm/heuristic that can predict us a value that is close to the actual running time of the program.

In this project we have used CUDA ptx code as an input for the prediction model to give us the desired output for its corresponding parallel CUDA code. The main idea has been to parse the ptx code to get input features. These features have then been used as an input to a Machine Learning algorithm, which then predicts the running time of the code. These features include the number of arithmetic computation , memory and other miscellaneous instructions, upper bound and lower bound of the latency of the code, Instruction Level Parallelism (ILP) and the total number of basic blocks. Programs from Rodinia benchmarks have been taken to train and test the model.

Subsequently, any new ptx code when taken as input has its features extracted and fed to to the learning model. This then predicts the execution time of the new parallel program. The machine learning used here has been logistic regression, due to its ability to work within a fixed data constraint and also to recognize non-linear patterns in the training data and build a model accordingly. However as we collect more data better ML algorithms like SVM will be used to capture the finer patterns that logistic regression will more likely to miss out on.

Chapter 2

Challenges

2.1 Feature Extraction

Extracting features from a ptx code is first among the challenging aspects of this project. Ptx code gives the representation of the one thread of the program. Hence it is difficult to capture features that are dependent on multiple threads like bank conflicts. In addition to that some features like loop prediction(no. of times loop runs) are also difficult to find due to these varying within a program and hence difficult to capture one value for the entire code. Other features that can be found using a single representation of the thread are Code Latency, ILP, instruction counts.

2.1.1 Code Latency

Calculating latency of the code was another difficulty because the absolute path taken in control flow graph(CFG) in never known until inputs are given to a problem and certain run time decisions are made. These include the input problem size, decisions made depending on run time variables which decides how many iterations of each loop are run in the program.

2.1.2 Instruction Level Parallelism

For finding ILP, we need to find the longest dependency chain in the data flow graph(DFG). The hard part was to do this for the entire program rather than just computing the ILP for each basic block. The main roadblock here was that finding the longest path in a graph is a

NP-complete problem. Hence coming up with a good heuristic that works for this given problem domain was a difficult task. In addition to that, the issue of finding instruction dependency in a parallel scenario was a difficult task as it involved ensuring all instructions that needed to be completed before the current one are done in sequential manner and are not parallelized.

2.1.3 Instruction Count

Separating the instructions into arithmetic , memory and misc. categories was also a minor challenge. This involved parsing the entire ptx code and going through each instruction and classifying it into one of the three categories.

2.2 Data Collection

And finally the data collection for the entire process was the biggest of all the challenges. This was majorly because the different algorithms require different inputs. Hence automating the entire process is a big roadblock.

Chapter 3

Algorithms

3.1 Instruction Counts

This is needed because memory instructions in general occupy a lot more cycles as compared to arithmetic instructions. Hence grouping all in one would not have captured this feature.

- Each instruction in ptx code is decoded and put into one category if the instruction contains any of arithmetic or memory instruction.
- These set of instructions for each of the three categories(arithmetic, memory and miscellaneous) are present in three different enumerations.
- Hence for each instruction type, we have a class, where one of its members (variables) tell in which enumeration type it belongs to.

3.2 Code Latency

- Form basic blocks using standard set of rules.
- Now, for each block which is represented by a basic block object, we represent it by a node in the graph.
- Connect two nodes when it follows a basic block or we jump into the current basic block from any other basic block. These form the edges in the graph.

- The graph is represented using a standard adjacency list representation.
- Compute the shortest path using Breadth First Search algorithm (since all edges are equally weighted). And sum up the latency values on this path. This gives a lower bound on the latency.
- Also since the graph is Directed Acyclic Graph(DAG),sort all the nodes in a topographic order.
- Using Depth First Search on this topographic order, find the longest path of the graph. Compute latency along this path. This gives an upper bound on latency.

3.3 Instruction Level Parallelism

- A DFG was formed using all the instructions, where each instruction is a node in the DFG.
- Dependencies were created between instructions iff one instruction had a jump statement(or any other control instruction) to the other or two instructions had a RAW(Read After Write) dependency.
- Barrier synchronization, function calls also created dependencies. Each dependency resulted in an edge between two nodes.
- Longest dependency chain was found in this graph using topographical sort and DFS.
- Dividing this by the total number of blocks gave the ILP value of the code.

3.4 Learning model

- The features that we got using the above algorithms were put in as input features to the ML algorithm
- For each ptx code these set of features formed the input vector for training the ML model.
- The ML model used is logistic regression.

Chapter 4

Future Work

The first work that has to be done is collecting a big enough dataset. This will help us using a better machine learning algorithms like SVM, Bayesian Models, Markovian Models without actually overfitting the data. These capture the non-linearity in data much better than logistic regression. Secondly some more features have to be collected. These include bank conflicts , loop iterations,loop divergence, thread divergence and also get number of thread and thread block used for each program. Fine-tuning the ILP is another area where this project can be improved. This will mainly depend on if a better way of capturing dependencies are possible. Also feeding in the hardware parameters as features will benefit a lot. And last of all this would be to automate the entire process, to cap off the project.

Chapter 5

Results and Conclusion

5.1 Results

The following table (see 5.1) shows the predicted value vs actual execution time for some samples. Some of them have been used for training hence could not be used for testing purposes.

5.2 Conclusion

The project has been almost completed with respect to finding features from ptx code that includes ILP, latency bounds, instruction of different types and basic blocks. Some other features that can be extracted from ptx code that include thread divergence and getting total number of blocks and threads will need to be done next. Collection of data and automating the process is also a big roadblock, that needs to be tackled first as a major priority. Then all that will be left will be choosing an appropriate ML model to train on the rich data and get a good prediction.

KERNEL NAME	EXECUTION TIME	PREDICTED TIME
shq : Z8kernel	0.064	0.07
shq : sum	0.05	0.55
fdtd : FiniteDifferencesKernel	2.256	2.0024
mergesort :mergeSortSharedKerne	0.72	0.7994
mergesort : mergeRanksAndIndicesKernel	1.28	1.316
mergesort : mergeElementaryIntervalsKernel	1.1	1.129
qrg : quasirandomGeneratorKernel	0.1377	1.1767
qrg : inverseCNDKernel	0.1199	0.055
scan : scanExclusiveShared	0.051	0.07
scan : uniformUpdate	0.64	0.7994

TABLE 5.1: Runtime prediction results

Appendix A

References

- NVIDIA Corporation, “NVIDIA CUDA C Programming Guide,” 2013
- A. Resios, “Gpu performance prediction using parametrized models,” 2011
- . Mei, K. Zhao, C. Liu, and X. Chu, “Benchmarking the Memory Hierarchy of Modern GPUs”
- NVIDIA Corporation, “NVIDIA Management Library,” 2015.
- NVIDIA, “Optimizing Application Performance with CUDA Profiling Tools,”
- Donghwan Jeon, Saturnino Garcia, Chris Louie, and Michael Bedford Taylor, “Parkour: Parallel Speedup Estimates for Serial Programs”, HOTPAR,2011