Modeling the Impact of Thread Configuration on Power and Performance of GPUs

Tiffany Connors[1], Apan Qasem (advisor)[1], Qing Yi (advisor)[2]
[1]Deptartment of Computer Science, Texas State University
[2]Department of Computer Science, University of Colorado at Colorado Springs

Abstract

Because graphics processing units (GPUs) are a low-cost option for achieving high computational power, they have become widely used in high-performance computing. However, GPUs can consume large amounts of power. Due to the associated energy costs, improving energy-efficiency has become a growing concern.

By evaluating the impact of thread configuration on performance and power trade-off, energy-efficient solutions can be identified. Using machine learning, the effect of applying a thread configuration to a program can be predicted in terms of the relative change in performance/power trade-off of a GPU kernel. This enables us to establish which dynamic program features are used to predict the impact of a thread configuration on a program's performance.

Experimental Setup

This study investigated the impact of 20 feasible thread configurations on the power and performance of four CUDA programs. These programs solve the quadratic assignment problem using 4 different algorithms: tabu search, simulated annealing, and 2 variations of 2opt. In addition, three different program input datasets were used from the QAPLIB [1]: lipa20, lipa30, and tai25a. Each dataset varies in size and structure, and thus affects the program's behavior.

Table I. Set of Thread Configurations

64x32	128x16	256x8	512x4	1024x2
64x64	128x32	256x16	512x8	1024x4
64x96	128x48	256x24	512x12	1024x6
64x126	128x64	256x32	512x20	1024x8

Using a script, the 20 thread configurations were applied individually to the four different codes, resulting in a total of 20 different versions of each program. The 80 different GPU kernels were then executed using the three input datasets. For each run, the average power consumption was measured using the built-in power sensor of the NVIDIA Tesla K20c GPU and the execution time was recorded.

The change in execution time and power consumption of each kernel was calculated by comparing the measured values to those of all other versions of the same program. The trade-off was then computed using the following formula:

$$trade\text{-}off = \frac{\Delta\ execution\ time}{\Delta\ power\ consumption}$$

Based on the trade-off value, the row of data was assigned a classification of either good or bad.

Good: trade-off ≥ 1 Bad: trade-off < 1

In order to create a machine learning model that works with more than one code, features which provide a good description of the kernel's characteristics must be determined. To do this, runtime behavior was analyzed using NVIDIA's GPU profiler, `nvprof`, and a set of 52 dynamic features was extracted. To normalize the values collected, each feature was divided by the number of instructions executed.

The set of features was then standardized by scaling the values between -1 and 1 by computing the z-score:

$$z_i = \frac{x_i - \mu}{\sigma}$$

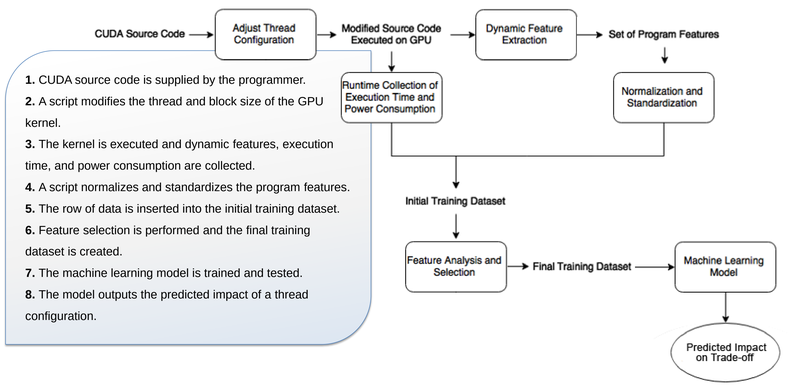

Figure 1. An overview of the process of collecting data and training the initial machine learning model.

Feature Selection

To reduce our set of features to include only those with the highest predictive power, feature selection was performed. First, any features with zero variance were removed. Features were then analyzed in R using correlation matrices. A feature is considered good if it is not highly correlated to the other relevant features [2]. Therefore, any features with a correlation of 95% or higher to another feature were removed from the set. The remaining features were compared with each of the models' variable importance values to determine if the features identified as significant factors in remained in the new set of features.

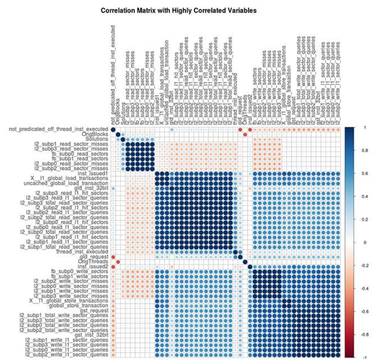

Figure 2a. Correlation matrix with all 52 features included. Dark blue represents a high correlation.

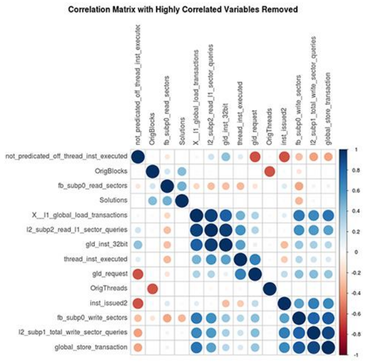

Figure 2b. Correlation matrix after highly correlated features have been removed.

Table II. Set of 14 Significant Predicting Factors

not_predicted_off_thread_inst_executed	Number of instructions executed by all threads
OrigBlocks	Number of blocks per grid
fb_subp0_read_sectors	Number of read requests sent to sub-partition 0
Solutions	OrigThreads * OrigBlocks
l2_subp2_read_l1_sector_queries	Accumulated read sector queries from L1 to L2 cache for slice 2
gld_inst_32bit	Number of instructions sent to 32bit global memory
thread_inst_executed	Number of thread instructions executed
gld_request	Number of executed global load instructions per warp
OrigThreads	Number of threads per block
inst_issued2	Number of cycles that issue 2 instructions
fb_subp0_write_sectors	Number of write requests sent to sub-partition 0
l2_subp1_total_write_sector_queries	Total write sector queries sent to slice 1 of L2 cache
global_store_transaction	Number of global store transactions
l1_global_load_transactions	Number of global loads in L1 cache

Machine Learning

Nine different machine learning algorithms available from the R caret package were used in this work. The purpose of using varied methods was to determine if the same features were significant factors across all models, as well as identify which algorithms worked best with our data.

The data was sorted into separate files based on target thread configuration. Each file was treated independently and separate models were built, trained, and tested for each of these configurations. The model accuracy was determined using repeated *k*-fold cross-validation. The machine learning algorithm which performed the best across all models was selected to be used for building the final predictive model.

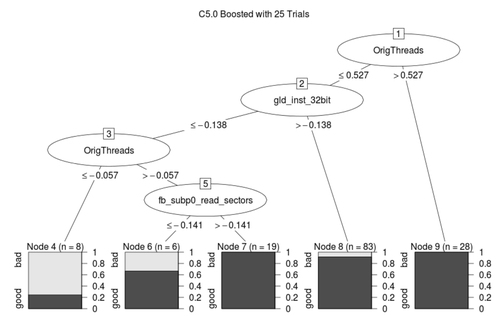

Figure 5. Final C5.0 decision tree for 128x16 target thread configuration.

Results

Through feature selection, the initial set of features was reduced from fifty-two to fourteen. The feature most frequently identified as the most significant factor was OrigThreads. It appears that if the original thread count is large, reducing the thread count to a lower number will be more beneficial.

The configuration with the greatest number of good instances was 128x16, producing desirable tradeoff 89.58% of the time. Changing the thread configuration to 1024x8 was always bad, therefore we were unable to build a model for this configuration since the response label had no variance.

The boosted C5.0 tree algorithm, with an average accuracy rate of 96%, was selected for building the final predictive model. The tree predicted that modifying the thread configuration to 128x16 had the highest probability of being bad if the original thread size is small, the number of instructions sent to 32-bit global memory is low, and a smaller number of read requests are sent to subp0. Otherwise, the change will likely result in improved performance/power trade-off.

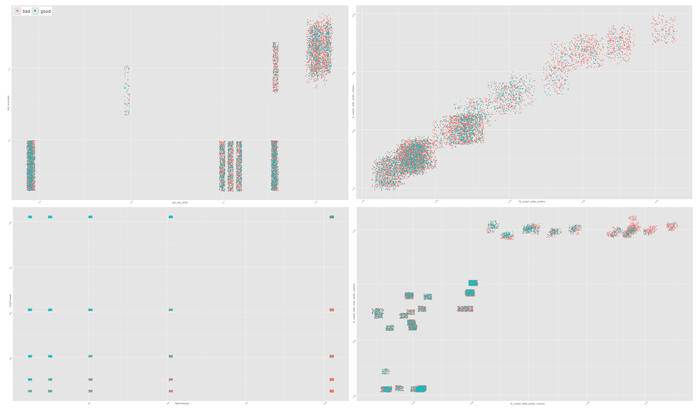

Figure 4. Scatter plots showing the distribution of good and bad configurations in terms of program features used in making predictions.

Discussion

Although GPUs provide exceptional performance speedup, they can consume large amounts of power. To improve the energy-efficiency of GPU systems, this poster proposes using machine learning to aid in the selection of thread configurations which increase performance while maintaining minimal increase in power consumption. By using dynamic feature extraction and selecting features most closely related to thread configuration, a machine learning model can accurately predict the impact of a thread configuration on the performance/power trade-off of a program.

Manually determining which thread configuration will provide optimal results is time consuming and involves modifying and running the program using each candidate thread configuration. This study addresses this issue by eliminating the need to modify and test the program for each candidate thread configuration, thus improving the time-efficiency of the process. Once set-up, the framework presented in this study requires only a single run in order to extract the dynamic features used by the predictive model.

Future Work

In addition to using machine learning to predict if a change in thread configuration will impact trade-off either positively or negatively, we intend to create a model whose output will be which thread configuration will have the greatest positive impact. Future work will expand the training dataset to include features from stencil code and the Parboil benchmark suite, allowing us to determine if the same features important for predicting the impact of thread configuration on QAP solvers are also significant factors for other code. Additionally, this will lead to a more versatile predictive model.

References

[1] "QAPLIB - A Quadratic Assignment Problem Library" http://anjos.mgi.polymtl.ca/qaplib/, 2014

[2] L. Yu and H. Liu, "Feature Selection for High-dimensional Data: A Fast Correlation-Based Filter Solution," in *Proc. of the 12th International Conference on Machine Learning*, ICML'03, Washington, DC, 2003, pp. 856-863.

Acknowledgment
This material is based upon work supported by the National Science Foundation REU program under Grant No. 1359275.

Scan Me!
Explore interactive graphs online.