STATuner: Efficient Tuning of CUDA Kernels Parameters *

[Extended Abstract]

Ravi Gupta
Purdue University
gupta237@purdue.edu

Ignacio Laguna
Lawrence Livermore Nat Lab
lagunaperalt1@llnl.gov

Dong H. Ahn
Lawrence Livermore Nat Lab
ahn1@llnl.gov

Todd Gamblin
Lawrence Livermore Nat Lab
gamblin2@llnl.gov

Saurabh Bagchi
Purdue University
sbagchi@purdue.edu

Felix Xiaozhu Lin
Purdue University
xzl@purdue.edu

ABSTRACT

CUDA programmers often need to decide the block size to use for a kernel launch that yields the lowest execution time. However, existing models to predict the best block size are not always accurate and involve a lot of manual effort from programmers. We identify a list of static metrics that can be used to characterize a kernel and build a Support Vector Machine (SVM) classifier model to predict block size that can be used in a kernel launch to minimize execution time. We use a set of kernels to train our model based on these identified static metrics and compare its predictions with the well-known NVIDIA tool called *Occupancy Calculator* on test kernels. Our model is able to predict block size that gives average error of 4.4% in comparison to *Occupancy Calculator* that gives error of 6.6%. Our model—called *STATuner*—requires no trial runs of the kernel and lesser effort compared to *Occupancy Calculator*.

1. INTRODUCTION

CUDA is a widely used programming model in heterogeneous computing systems containing GPUs. The code is usually divided into two sub-sections: *host code* and *kernel code*. Kernel code is the sub-section that runs on the GPU and each instance of the kernel code is called a thread. To launch the execution of a kernel on GPUs, the programmer must give number of threads to be grouped into a block (*block size*), which runs on a multiprocessor of the GPU. As the number of threads in a block affect different resources, such as shared memory and registers, kernels use GPU multiprocessors differently and thereby result in performance variation. For key kernels, it is important to understand the constraints of the kernel and the GPU it is running on to choose a *block size* that will result in good performance (usually low execution time).

2. BACKGROUND

A common practice to choose a good *block size* is to aim for high occupancy, which is the ratio of the number of active warps (i.e., 32-thread groups) per multiprocessor to the maximum number of warps that can be active on the multiprocessor at once. However, in our experiments we find that occupancy does not always map well to performance and that there are certainly other metrics to explore and to correlate to predict optimal block size.

*This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DEAC52-07NA27344 and supported by Office of Science, Office of Advanced Scientific Computing Research (LLNL-ABS-676066).

3. METHODOLOGY

We identify a list of static metrics that play an important role in the performance of a kernel code on GPU, which we call our *feature set*, and use them to characterize a kernel code. We build a pool of training set consisting of kernels from Rodinia Benchmark Suite [2] and use them to train a machine leaning model, which we use to classify the optimal block size of new kernels.

3.1 Static Metrics and Training Set

We extract CUDA kernels from the application and place them in separate files. We then compile them using LLVM to generate LLVM bitcode and LLVM IR. CUDA specific directives (e.g., `__global__`) are re-declared to allow Clang recognize them. We then use an LLVM pass to collect these static metrics:

- num_branch_instructions / num_instructions
- num_load_instructions / num_instructions
- num_loops
- num__syncthreads_operations
- registers_usage
- shared_memory_used_per_block
- num_int_instructions / num_instructions
- num_float_instructions / num_instructions

We use the above procedure on a variety of kernels from the Rodinia Benchmark Suite to characterize them and build a repository of feature sets. We then run these applications for different block size and record the minimum execution time and corresponding optimal block size. The feature set vector and optimal block size as label together form the input matrix for the Machine Learning algorithm.

3.2 Training STATuner

We use a *Support Vector Machine classifier* on the training set to build our model. We use a total of 13 kernels in to train our model. To predict optimal block size for a new application, we extract its kernel and use our LLVM pass to generate its feature set vector. This feature set vector is then put as an input in our model and the predicted block size is the output. The training workflow is shown in Figure 2 and the training input matrix is shown in Figure 1.

SVM : Training Input Matrix								
	S1	S2	S3	S4	S5	S6	S7	Label(supervised)
App_1	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}	X_{16}	X_{17}	Y_1
App_2	X_{21}	X_{22}	X_{23}	X_{24}	X_{25}	X_{26}	X_{27}	Y_2
App_3	X_{31}	X_{32}	X_{33}	X_{34}	X_{35}	X_{36}	X_{37}	Y_3
.	.	.	.	.	.	.	.	Y_4
.	.	.	.	.	.	.	.	Y_5
App_n	X_{n1}	X_{n2}	X_{n3}	X_{n4}	X_{n5}	X_{n6}	X_{n7}	Y_6

Figure 1: Training Input Matrix

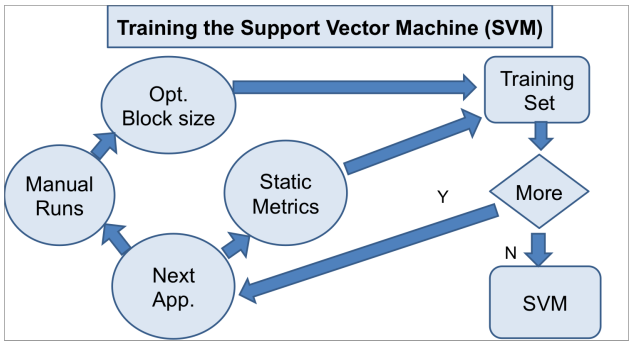

Figure 2: Training Support Vector Machine

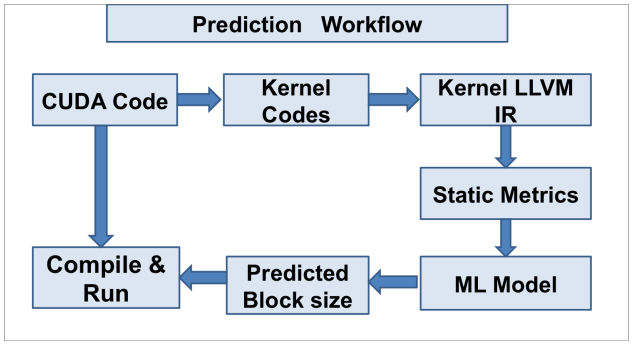

Figure 3: Prediction from STATuner Workflow

3.3 Prediction from STATuner

To predict optimal block size for a new application, we extract its kernel and use LLVM pass to generate its feature set vector. This feature set vector is then put as an input in our model and the predicted block size is the output. The prediction workflow is shown in Figure 3.

4. RESULTS

To gauge the accuracy of our model, we build a test set containing the test applications feature set vector and its recorded normalized execution time. We use 5 new kernels in our test set. The test set vector is used only for prediction. We calculate the error of the prediction as follows:

$$Error = \left[\frac{Execution\ Time_{Predicted\ block\ size}}{Execution\ Time_{Optimal\ block\ size}} - 1 \right] * 100$$

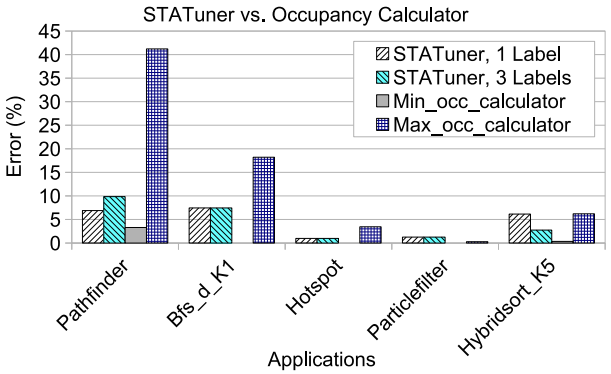

Figure 4: Comparison of the error results between STATuner and the NVIDIA Occupancy Calculator.

This error is representative of how far off the predicted block size execution time is from the minimum execution time recorded. We compare our results with a well-known tool called *Occupancy Calculator*. We see that average error recorded with our model is 4.4%. With *Occupancy Calculator* average error recorded is 6.6 % on the test set. Another advantage of *STATuner* over Occupancy Calculator is that there is single prediction from STATuner while there can be multiple predictions from Occupancy Calculator and each of the predictions has to be run to identify the best out of them. The results are shown in Figure 4.

Note that, with the Occupancy Calculator, programmers get multiple block sizes by providing input, such as registers per thread and per-block shared memory. To obtain the optimal block size (i.e., the one that yields the lowest execution time), programmers must run the application with these distinct block sizes. In contrast, STATuner gives a single block size. In our experiments, we use five block sizes from the Occupancy Calculator, and in the Figure 4 we show the minimum and maximum error from these values.

5. CONCLUSIONS

We conclude from our experiments that occupancy does not map very well to performance and other static metrics need to be considered while predicting optimum Block size. With enough metrics and training datasets, it is possible to define a good model and to get accurate predictions for optimal block size with minimal manual efforts.

6. REFERENCES

[1] CUDA Occupancy Runime API, `docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__OCCUPANCY.html`

[2] Che, Shuai and Boyer, Michael and Meng, Jiayuan and Tarjan, David and Sheaffer, Jeremy W and Lee, Sang-Ha and Skadron, Kevin*Rodinia: A benchmark suite for heterogeneous computing*. IEEE International Symposium on Workload Characterization, 44–54, IISWC 2009.