The Boat Hull Model: Adapting the Roofline Model to Enable Performance Prediction for Parallel Computing

Cedric Nugteren Henk Corporaal
Eindhoven University of Technology, The Netherlands
{c.nugteren, h.corporaal}@tue.nl http://parse.ele.tue.nl

Abstract

Multi-core and many-core were already major trends for the past six years, and are expected to continue for the next decades. With these trends of parallel computing, it becomes increasingly difficult to decide on which architecture to run a given application.

In this work, we use an algorithm classification to predict performance *prior* to algorithm implementation. For this purpose, we modify the *roofline model* to include class information. In this way, we enable architectural choice through performance prediction prior to the development of architecture specific code. The new model, the *boat hull model*, is demonstrated using a GPU as a target architecture. We show for 6 example algorithms that performance is predicted accurately without requiring code to be available.

Categories and Subject Descriptors C.1.4 [*Processor Architectures*]: Parallel Architectures; C.4 [*Performance of Systems*]: Modeling Techniques

General Terms Performance

Keywords Parallel computing, performance prediction, many-core accelerators, the roofline model

1. Introduction

For the past five decades, single-core performance has shown an exponential growth, enabling technology to become pervasive and ubiquitous in our society. The exponential growth of single-core performance has ended in 2004, making place for a parallel and heterogeneous computing era. Trends such as multi-core and many-core are expected to continue for the next decades. Although many-core architectures such as the Graphics Processing Unit (GPU) might be suitable for one type of application, other applications might prefer multi-core processors. This creates a heterogeneous computing environment, with both types of processors in one system or even on a single chip.

With current and future processors, it becomes increasingly difficult to decide on which processor to run an application or algorithm. Existing performance prediction techniques such as mathematical models or simulators require code to be available and optimized for a target processor. However, programming such processors has become increasingly challenging and time consuming [1]. Therefore, a performance prediction method which does not require code to be available is desirable.

In this work, we present the *boat hull model*. We modify the *roofline model* [3] such that it generates multiple inverse rooflines. Each of these inverse rooflines is specific for an algorithm class. Available algorithm classes are defined using an algorithm classification, of which more details are found in [2].

2. Background: the roofline model

The roofline model was introduced as an easy to understand performance model capable of identifying performance bottlenecks [3]. This model gives a rough performance estimate based on the assumption that performance is limited by either peak memory bandwidth or by peak ALU throughput. The roofline model is processor specific: for each processor there is a specific instance of the model.

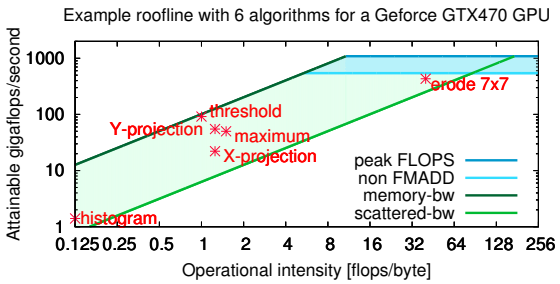

Figure 1. Applying the roofline model to 6 example algorithms.

We map 6 algorithms onto the roofline model, of which the results are shown in figure 1 for a GeForce GTX470 GPU. The location of each algorithm is based on two aspects: 1) the performance of a CUDA implementation executed on the GPU, and 2), the operational intensity in ALU operations per off-chip load/store.

Two obstacles for using this model to predict performance are observed: 1) the execution time is not directly visible, and 2), the range of the predicted performance is very wide. For example, as shown in figure 1, the performance of the `X-projection` algorithm is a factor 7 beneath the memory bandwidth roof.

3. The boat hull model

Selecting which processor architecture is best suited for a given application can be done using architecture models or hardware simulators. These methods do however require the presence of optimized target architecture code, which is often not available when making an architectural choice. The roofline model does give an indication of the expected performance without requiring code, but falls short when an application's compute or data-access patterns

PPoPP'12, February 25–29, 2012, New Orleans, Louisiana, USA.
ACM 978-1-4503-1160-1/12/02.

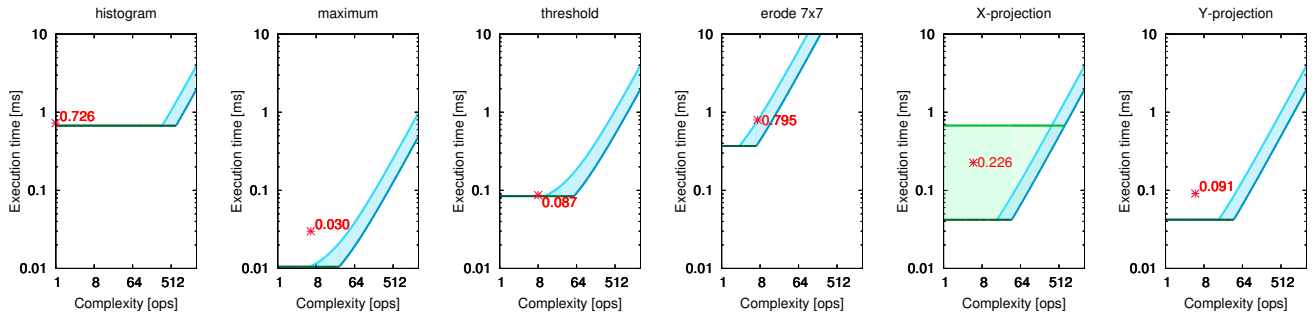

Figure 2. Applying the boat hull model to 6 example algorithms for the GeForce GTX470 architecture. Red star symbols show the measured performance, while the lines show the predicted performance. The legend is as shown in figure 1.

are non-ideal. To enable performance prediction prior to algorithm implementation, we introduce a modified version of the roofline model based on an algorithm classification presented in [2].

The modified model, referred to as the *boat hull model*, makes the following changes to the roofline model:

- For each algorithm class, we fine-tune the original model to match the properties of such a class. Because the amount of off-chip data accesses is inherent to a class-architecture combination, the metric on the horizontal axis of the model can be changed from 'operations per byte' into 'complexity': the number of operations given for a class' operator $f()$ (see [2]).
- The metric on the vertical axis of the model is changed from 'flops per second' into 'execution time'. This creates an inverse view of the roofline model, resembling a boat's hull.

We have developed a tool based on the boat hull model which requires as inputs processor parameters and an algorithm class, and outputs a visual model. The required processor parameters are high level and are similar to those required for the original roofline model, e.g. peak ALU performance and peak memory bandwidth. The tool currently supports both CPUs and GPUs. For each algorithm class, a set of parameters is pre-defined, limiting the original roofline model's roofs and ceilings. For example, on a GPU, a convolution type of operation might perform a certain number of scattered memory accesses based on its neighbourhood size. These scattered accesses will limit the maximum achievable bandwidth, which is taken into account in the class parameters.

4. Example application

To illustrate the use of this work, we present 6 example image processing algorithms from a computer vision application. They are targeted for GPU acceleration using a GeForce GTX470 and are classified as shown in table 1. The classification is according to the algorithm classification's grammar and vocabulary, which is explained in [2].

primitive	classification
histogram	1024x1024|element → 256|shared
maximum	262144|element → 1|shared
threshold	1024x1024|element → 1024x1024|element
erode 7x7	1024x1024|neighb(7x7) → 1024x1024|element
X-projection	1024x1024|tile(1x1024) → 1024|element
Y-projection	1024x1024|tile(1024x1) → 1024|element

Table 1. Classification of 6 example image processing algorithms.

For each algorithm we generate a boat hull model based on the corresponding algorithm class. The results are shown in figure 2, in which the measured performance is marked with a red star symbol. From the results, we observe that the performance of the algorithms `histogram`, `threshold` and `erode` are accurately predicted. The `maximum` primitive shows a higher execution time compared to the predicted time, which is caused by its small problem size, making the algorithm less suitable for GPU acceleration. Similarly, both the `X-projection` and `Y-projection` algorithms suffer from additional overheads for small problem sizes. Because memory accesses might be scattered for algorithms such as `X-projection`, a wider prediction range is given.

We compare the boat hull model (figure 2) with the roofline model (figure 1). Both figures show the performance of the 6 example algorithms. The roofline model appears not suitable for performance prediction, while the boat hull model predicts performance with a small error in most cases. Comparing to performance models and architecture simulators, we observe that the boat hull model has the following advantages: 1) it is straightforward to understand, 2) it requires little architectural information, and 3), most importantly, it requires no code implementation nor mapping for the target architecture to be available.

5. Summary

In this work we presented a processor and algorithm class specific visual model to predict an application's performance. This model is based on the roofline model, which is adapted to include algorithm class information. With the new *boat hull model*, we are able to predict performance without requiring code to be available. Programmers are not required to port and optimize code for the target processors, which enables rapid processor selection early in the design process.

We have given an overview of the concepts and ideas behind the boat hull model. We have also demonstrated the use of the model for an example application, for which we showed the predicted and measured performance.

References

[1] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco. GPUs and the Future of Parallel Computing. *IEEE Micro*, 31:7–17, September 2011.

[2] C. Nugteren and H. Corporaal. A Modular and Parameterisable Classification of Algorithms. Technical Report No. ESR-2011-02, Eindhoven University of Technology, 2011.

[3] S. Williams, A. Waterman, and D. Patterson. Roofline: an Insightful Visual Performance Model for Multicore Architectures. *Communications of the ACM*, 52:65–76, April 2009.