PhD Research: Anti-Parallel Patterns
Start Date: January 2011

Jan G. L. Cornelis
ETRO
VUB
Brussels, Belgium
Email: jgcornel@vub.ac.be

Promoter: Jan Lemeire
ETRO
VUB
Brussels, Belgium
Email: jan.lemeire@vub.ac.be

***Abstract*—This paper gives an overview of our PhD research which investigates the usefulness of the concept of anti-parallel patterns in the domain of parallel performance analysis. We believe this concept is a promising method that complements other methods that assist the developer in analyzing, understanding and improving the performance of her parallel programs. Anti-parallel patterns are common parts of parallel programs that cause the performance of the program to be less than ideal, namely a speedup of p when running on p processing elements.**

First, the patterns provide a way of structuring research and can be used to create an inventory. Next, we will analyze whether a parallel application can be completely characterized by the anti-parallel patterns it contains. This depends on the orthogonality of the decomposition. If orthogonality holds, the impact of each pattern can be modeled separately. If a weaker form of orthogonality holds, we might also have to model the interaction between the patterns. For putting patterns into practice, an important question is whether they can be detected automatically. The impact on the performance will be studied by benchmark programs. These benchmarks can then be used to compare the performance behavior of parallel platforms.

***Keywords*-performance analysis; performance modeling; patterns; orthogonality**

I. Introduction

Parallel computing power is not for free. It puts an extra burden on the software development and must be backed by a performance analysis to ensure efficient parallel execution. Our research aims at enhancing and automating the performance analysis by introducing a novel angle to look at the software-on-hardware performance.

II. Approach

We propose *Anti-Parallel Patterns* (APPs) as a new way to carry out a performance analysis. We define anti-parallel patterns as common parts of parallel programs that cause these programs to have less than ideal parallel performance, where the ideal speedup equals the number of processors. We will study the performance of a parallel program by identifying the anti-parallel patterns it contains.

This approach is complementary to the one adopted by traditional performance analysis approaches and tools. These try to help programmers by finding the main bottlenecks that cause their programs to have less than ideal performance and then mapping the overheads to parts of the program that could be optimized. For example, Paradyn [3] automatically locates potential bottlenecks in parallel code and tries to explain why, where and when the problems occur. We analyze performance in the opposite direction by mapping parts of the program corresponding to APPs to sources of overhead. We hypothesize that the relevant program characteristics can be determined from the APPs it contains, and that they can help us analyze, estimate and improve the performance of a given program running on a parallel platform, as well as compare different hardware platforms and find the best platform match for a given application.

The initial focus will be on GPUs and data-parallel programs.

III. Methodology

To investigate the potential of our approach, the following deliverables have been defined:

A. Inventory

Since APPs capture the program properties influencing the performance, they can be used to organize a detailed inventory of these influences. This inventory will contain a record for each APP that we identify and will contain at least:

- the pattern's description;
- its behavior and the overheads it generates;
- the parameters that determine its behavior and
- the remedies that alleviate the performance degradation it causes.

Existing work will be 'attached' to this inventory according to the patterns they describe. The inventory will serve as a single point of information.

B. Performance Modeling

First we will study the individual impact of an APP on the parallel performance. We will try to model the performance of each pattern. To do so, we will create a set of benchmark programs for each pattern. These benchmark programs allow us to objectively measure the behavior of a pattern in function of its parameters. They will be dummy programs that exhibit only one APP and that are configurable to any parameter setting (any instance of the APP). The results of running these benchmark programs, together with our understanding of the parallel platform on which the benchmark is run will allow us to model the performance of each APP qualitatively and quantitatively.

Next, we will model the impact on the performance of combinations of patterns. The complexity of such a global model depends on *the orthogonality of the APP decomposition*.

C. Orthogonality Study

The question of the orthogonality study is whether the decomposition of a program into its APPs results in a decomposition of the performance, i.e. whether the total performance degradation is a simple combination of the degradation resulting from each pattern individually, as shown in Figure 1. We suspect that a weaker form of orthogonality must be considered. This means that we also have to model the interaction of patterns.

The orthogonality question can be expressed as follows. Overhead is expressed as the deviations from the optimal parallel execution:

$$overhead = p.T_{par} - T_{seq}. \tag{1}$$

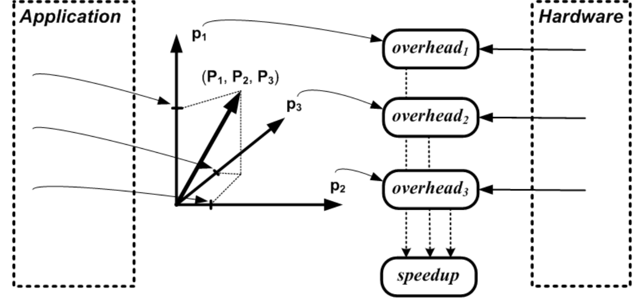

Figure 1. APPs add an extra layer to the performance analysis. The performance of an application can be completely characterized by the APPs it contains, if the decomposition is orthogonal with respect to these APPs.

where p is the number of processors and T_{seq} and T_{par} respectively the sequential and parallel run time. Ideally, when having p processors, the run time needed to complete the task is T_{seq}/p. The additional processor cycles used by the parallel execution are considered overhead or lost cycles, as conceived by the *lost cycle approach* [2]. Orthogonality would mean that the overhead generated by each pattern is independent from the occurrence of other patterns. The overhead $overhead_i$ generated by a single pattern i having parameter configuration $\mathbf{P_i}$ is modeled as

$$overhead_i = f_i(\mathbf{P_i}). \tag{2}$$

Note that $\mathbf{P_i}$ is a vector and f_i a function which also depends on the hardware. Orthogonality would mean that the total overhead $overhead_{total}$ can be calculated as

$$overhead_{total} = \sum_i overhead_i \tag{3}$$

for any parallel application. A weaker form of orthogonality is expressed by the possibility to model the performance as

$$overhead_{total} = f(\mathbf{P_1} \ldots \mathbf{P_n}) \tag{4}$$

The orthogonality properties will be derived by searching and constructing counter examples and examining them. Ideally, we will come up with a mathematical model that approaches reality close enough.

D. Automation

Positive answers to the orthogonality question determine the usefulness of APPs in a performance analysis. If useful, we will study to what extent APPs can be

integrated into a performance tool. For each pattern we will try to establish whether it can be identified automatically by a tool either at compile time (static) or at run time (dynamic). A proof of concept for static identification can be developed by extending existing compilers or by using existing compiler frameworks. For dynamic identification it should be possible to start from existing trace analyzers.

E. Performance Study of Parallel Platforms

The benchmark programs developed in III-B serve a double purpose: apart from helping us model the behavior of an APP on a given platform, they can also serve to measure the performance of a given architecture in the face of a given APP. In this way, the hardware's behavior can be characterized. The detailed results of running each benchmark on different parallel platforms will allow us to compare parallel platforms in the context of APPs.

The success of these benchmarks for characterizing the performance behavior for parallel applications in general will depend on the degree of orthogonality of the APPs. If perfectly orthogonal, the benchmark results fully characterize the hardware. If not, the interactions should be taken into account by making them explicit.

F. Application - Platform Match

Finally, the work performed up to this point will be used to match platforms to applications. This can be done by establishing which APPs are present in an application and to which degree and by determining which platform has the best performance for this specific combination of APPs. A number of case studies will be performed to determine the validity of this approach.

IV. Significance

We approach performance analysis and estimation from a new angle. The novelty is the introduction of an intermediate layer between program and performance analysis such that once a program is decomposed into its APPs, the performance analysis is given by the specific combination of APPs (Figure 1). Moreover, the performance analysis of APPs can be performed independently from the applications. The significance of the approach will depend on the orthogonality of the decomposition.

We will try the "anti-parallel pattern" concept for many parallel platforms and programming paradigms, thus hopefully providing a unified way to reason about parallel performance that is complementary to current approaches. The answers to the questions asked above will determine the significance of APPs. The most important questions are whether there exists an exhaustive list of patterns that explain all performance degradation, and to which degree the decomposition of a program into its APPs results in a decomposition of its performance. Also, questions about the automatability of the detection and remediation of APPs need to be answered. All these questions will be tackled during our research.

V. Related Work

As far as we know there is no work that uses this perspective. Furthermore, a great deal of existing and future work can be integrated in/organized according to this APP framework.

As mentioned earlier in section II, our approach is complementary to the already large volume of work in the domain of parallel performance analysis. An approach that comes close to ours is found in [4]. They try to predict the performance of an application using algebraic mappings - convolutions - of application profiles to machine signatures of the machines on which the application will run. The difference is that we decompose this characterization into APPs. This might make the task of analyzing the performance of parallel applications easier. [2] fits overhead measurements to analytical forms in order to predict the performance of a given application.

VI. Past and Future Work

In the context of a master's thesis [1], we explored the anti-parallel pattern concept for fine-grain data parallel programs that run on NVIDIA GPUs. We identified 4 APPs and tried to model their behavior using simple benchmarks. The following subsections briefly present each APP and give the highlights for each of them.

A. Branching

Definition: Branching refers to code that contains conditional statements, such that, when run in parallel, it causes different processing elements to follow different execution paths.

Discussion: From the benchmarks it became clear that the behavior of branching on NVIDIA GPUs is very deterministic. As a consequence it is easy to model the lost cycles caused by branching. Furthermore, we presented a generic solution to alleviate its effect and showed that branching can be detected in a straightforward manner.

B. Limited Parallel Data Access

Definition: Limited parallel data access refers to the lack of concurrent access by different processing elements to shared memory.

Discussion: We looked at the effect of non-ideal access patterns for the two main memory types found on NVIDIA GPUs. To do so we created benchmark programs that are configurable by their memory access pattern. We provided guidelines that can be followed to optimize the memory access, but did not provide a generic solution. Finally, detecting less than ideal memory access is best done at runtime.

C. Synchronization Points

Definition: Synchronization points are points in the program at which processing elements join up to ensure that a part of the work has been done.

Discussion: As can be expected and as shown by the benchmarks, the effect of synchronization on performance depends upon the amount of computation that is available between synchronization points. It is not possible to provide a generic solution to alleviate the effects of synchronization, given that these are generally necessary in a given application. Detecting synchronization points, however, can be easily done by analyzing the application's source code.

D. Data Partitioning and Mapping

Definition: Data partitioning and mapping refers to the way data is mapped to the processors that execute the computation.

Discussion: The way we partition the data in blocks and the way data elements are mapped to processing elements can have a dramatic impact on the performance of a program running on a GPU. This was shown by running a few simple benchmarks. For a given kernel it should be possible to calculate the optimal partitioning, for example NVIDIA provides an Excel sheet with similar functionality. Determining the ideal mapping of data elements to processing elements might be more intricate given the number of conflicting aspects that are involved. In this case it could be interesting to instrument the code to measure the average runtime of a thread block.

E. First Orthogonality Results

Branching on GPUs is orthogonal with respect to the other patterns. The overhead is independent from the occurrence of other patterns. The last two patterns determine to which degree the latency caused by the second pattern can be hidden. As such, these patterns are not orthogonal. It might be possible to add latency hiding as an extra dimension of our decomposition, and to expand Equation 4 with the parameters of latency hiding.

F. Conclusion

We used a case study - a parallel prefix sum - to try the usefulness of the APP concept. This case study showed that the concept is a promising approach to look at parallel performance, but also that more work is needed to elaborate this approach.

The master's thesis introduced the anti-parallel patterns concept and provided a first exploration of it. The section about the methodology explains how the in-depth study of the PhD will be carried out.

References

[1] Jan Cornelis. Anti-parallel patterns in fine-grain data-parallel programs. Master's thesis, Vrije Universiteit Brussel, Belgium, 2010.

[2] Mark E. Crovella and Thomas J. LeBlanc. Parallel performance prediction using lost cycles analysis. In *Proceedings of Supercomputing '94*, pages 600–609, 1994.

[3] Barton P. Miller, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, R. Bruce Irvin, Karen L. Karavanic, Krishna Kunchithapadam, and Tia Newhall. The paradyn parallel performance measurement tool, November 1995.

[4] Allan Snavely, Nicole Wolter, and Laura Carrington. Modeling application performance by convolving machine signatures with application profiles, 2001.