Proceedings of the First PhD Symposium on Sustainable Ultrascale Computing Systems (NESUS PhD 2016)
Timisoara, Romania

Jesus Carretero, Javier Garcia Blas
Dana Petcu
(Editors)

February 8-11, 2016

Application Partitioning and Mapping Techniques for Heterogeneous Parallel Platforms

Rafael Sotomayor, J. Daniel Garcia

University Carlos III, Spain
rsotomay@inf.uc3m.es, jdgarcia@inf.uc3m.es

Abstract

Parallelism has become one of the most extended paradigms used to improve performance. Legacy source code needs to be re-written so that it can take advantage of multi-core and many-core computing devices, such as GPGPU, FPGA, DSP or specific accelerators. However, it forces software developers to adapt applications and coding mechanisms in order to exploit the available computing devices. It is a time consuming and error prone task that usually results in expensive and sub-optimal parallel software.

In this work, we describe a parallel programming model, a set of annotating techniques and a static scheduling algorithm for parallel applications. Their purpose is to simplify the task of transforming sequential legacy code into parallel code capable of making full use of several different computing devices with the objetive of increasing performance, lowering energy consumption and increase the productivity of the developer.

Keywords Parallel computing, heterogeneous computing, programming models, kernel partitioning

I. Introduction

In recent years, traditional approaches to improving CPU performance have reached a limit due to the limitations of sequential programming models as well as the physical constraints related to clock speed (such as heat dissipation or power consuption). As a result, efforts have turned to developing heterogeneous hardware architectures, combining several computing devices other than CPUs (such as GPUs, FPGAs or DSPs), programmed in a highly parallel fashion.

This approach, however, has limitations. Firstly, each kind of device has a different architecture, and it is usually necessary to follow a very specific programming model. This makes it very difficult to write code that makes full use of these heterogeneous architectures. Secondly, a very intimate knowledge of both architectures and programming models is necessary to make an efficient use of these devices with regards to high performance and low energy consupmtion.

The purpose of this work is to develop a unified programming model that can be used in this kind of heterogeneous parallel platforms in order to: (1) reduce power consumption, (2) improve performance and (3) increase productivity realizing designs.

The rest of the paper is organized as follows. In section II, the related work is summarized. Section III presents the proposed model. Finally, section IV shows the results and conclusions drawn so far, and outlines some future work.

II. Related work

In the literature we can find pragma-based frameworks that allow executing code in multi-devices. Some works take advantage of open standards in order to execute legacy code block in GPUs. Some examples are Wienke et al. [2], based on OpenACC, and Bertolli et al. [3] who use the newest version of OpenMP 4.0.

From a semantic viewpoint, C++11 attributes provide some advantages over pragma-based frameworks

[4]. They do not need support from the preprocessor, they can be applied to every syntactic element in the code, and they provide a portable way of annotating code.

Automatic kernel selection techniques is an important research field for automatic serial code parallelization [5], including GPU source code transformation. Multi-core as target devices is also considered for automatic source code transformation. For example, polyhedral tools are used in order to create source code that improves cache accesses with tiling optimizations [6]. However, all of these tools focus on one particular kind of optimization , such as CPU-only, accelerators-only)

The Open Computing Language (OpenCL) [1] is a C-based programming model, used for different computing devices (e.g. CPUs, GPGPUs, DSP, FPGA, accelerators) that has become widely accepted and supported by major vendors. OpenCL is based on parallel code regions, called kernels, that can be executed on a device. OpenCL allows the development of heterogeneous parallel applications that could use more than one computing device, improving application efficiency. It's use with CPU for HPC systems has been studied in recent years [9], concluding that the performance is close to OpenMP and other library solutions.

III. Thesis project

As explained before, the goal of this work is to develop a unified programming model targeting several programming devices under a single annotation system based on C++11 attributes. This model draws heavily from OpenCL. The different sections of the code susceptible to parallelization are refered to as kernels. Also, the memory model is host-centric, and the CPU, acting as said host, is in charge of orchestrating memory transfers to and from device memory space. Also, a set of code annotation techniques are developed to allow the transformation and optimization of sequential code into parallel heterogeneous code.

To this end, the following milestones have been set:

III.1 Hardware description tool

In order to split the parallel kernels between the different devices in an efficient manner in line with the goals of performance and energy consumption, it is necessary to know the capabilities and limitations of the different devices that make up an heterogeneous parallel platform. From now on, we will refer to an heterogeneous parallel platform as one made of multicore, GPGPU,with CPU for HPC systems has been studied in recent years [9], concluding that the performance is close to OpenMP and library solutions FPGA, DSP or combinations of all the previous.

The Heterogeneous Parallel Platform Description Language (HPP-DL) is a specification of a description language that provides all the relevant details of an heterogeneous parallel platform. It is designed to be human readable, so that automated and non-automated descriptions of platforms can be made. JSON (JavaScrip Object Notation) format has been adopted to represent the HPP-DL information.

HPP-DL allows to express the characteristics of a hardware system via a hierarchical model. Its intended use is making sure that platform-specific information is made available to (1) expert programmers and (2) tools such as auto-tuners, compilers or run-time systems. The HPP-DL format is independent of the programming model used. This means that it can be used as a virtual platform for other offline simulations. HPP-DL makes use of existing tools, mainly Hardware Lister (lshw) and Hardware Locality (hwloc) tool.

With the HPP-DL, the hardware parallel platform can be described in terms of:

- **Components**: each of the parts that make up the whole HPP, such as processors, memory banks, cache or different devices, and they are interconnected in various ways. Different devices contain different information.
- **Links**: this entity represents the relationships between two different components of the HPP. It describes a one-way connection in terms of throughput and latency. An example of link is the PCIe between the board and a GPGPU. It currently does not cover connections between different computers.
- **Resources**: refer to IS-specific information about resources used by/allocated to a component, such as I/O ports, IRQs or address ranges. Their main

use is to develop code for FPGA boards, where low-level memory operations are necessary.

III.2 Software annotation mechanisms

This mechanisms are based on an ad-hoc set of C++11 attributes [10]. Their purpose is to include semantic information about the kernels, so that the sequential code may be automatically optimized and, ultimately, transformed for a specific device.

These attributes can be used to define kernels in a code base, their behaviour (e.g. `rpr::map`) and their parameters (e.g. `rpr::in`). We refer to these parallel regions as *kernels*. An attribute is attached to a syntactic entity (e.g. a statement, loop, or definition), as defined by the standard C++ grammar. In general, an annotation precedes the syntactic element it is annotating to and does not require any preprocessing (a key difference with *pragma* based solutions).

Listing 1: Block-based matrix multiplication with `REPARA` attributes.

```
[[rpr::kernel, rpr::map,
  rpr::in(A,B,C,AN,BN,CN,b),
  rpr::out(C)]]
for(long i=0;i<mblocks;++i)
 for(long j=0;j<nblocks;++j)
  for(long k=0;k<pblocks;++k) {
   double *Aik = &A[b*(i*AN + k)];
   double *Bkj = &B[b*(k*BN + j)];
   double *Cij = &C[b*(i*CN+j)];
   MMul(b,Aik,AN,Bkj,BN,Cij,CN);
  }
```

Listing 1 shows a basic map computations using our attributes. The attribute `rpr::kernel` annotates the subsequent single or compound statement expressing the programmers intent of marking it as a kernel region. Kernel nesting is not considered in our model, therefore when two or more kernels are nested, inner annotations are ignored.

Additional attributes may be applied to a kernel region to refine intentions and to provide additional information. For example, `rpr::map` or `rpr::farm` can be used to express the expected parallel pattern transformation.

Additionally, the `rpr::in` and `rpr::out` attributes are used to identify input and output parameters of the kernel, respectively. Input/output sets do not need to be disjoint, allowing a parameter to be both input and output when needed.

A tool has been created to automatically detect kernels and transform the sequential code to OpenCL code [11]. We propose a workflow containing four different stages:

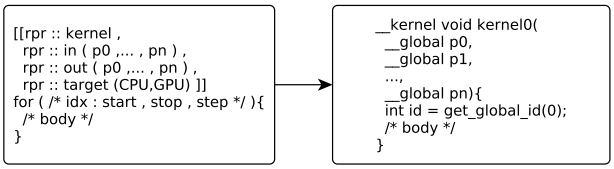

Figure 1: Basic source-to-source transformation process.

Kernel detection. Applies a set of rules to find potential kernels in a legacy C++ source code.

IR multi-version generation. This stage takes the output from the previous stage and generates a set of possible versions for each kernel.

Multi-version selection. For each set of versions, we apply Multiple Attribute Decision Making (MADM) techniques to filter the most promising versions.

OpenCL code generation. For each version provided in the previous stage, we generate the *OpenCL* equivalent code that performs the configuration of the kernel parameters, executes the kernel, and performs the cleanup process. An example of code generation from an independent for-loop to an equivalent kernel is shown in Figure 1.

III.3 Static software partitioning and scheduling techniques

After profiling both hardware and software, it is necessary to schedule the kernels marked in an application to be run into specific devices. Currently, a static, offline scheduling algorithm has been implemented [8].

This algorithm is based on four key aspects: kernels, input/output size, devices and transfer rates. Each pair of kernel and input/output size takes a certain time to run. Also related to the data size is the transfer rate. Lastly, each device has its own strengths and limitations, and as such their performance will vary from kernel to kernel.

With this, we represent the different possible schedules as nodes in a tree, where the root node is an empty schedule, and the leaf nodes are full schedules where all kernels have been assigned to a device. With this, we take the schedule that takes the least time. It is possible to add feedback on energy consuption and, by introducing weights for each measure, prepare the model so that the user can configure it as needed.

IV. Conclusions and future work

In this work, we have developed a unified heterogeneous model that allows to describe heterogeneous parallel platforms composed of different computational devices. It also allows to orchestrate different kernels into said devices to be executed in parallel.

We also have developed several tools that automatize the hardware description, code annotation and scheduling optimization. The last two have been tested in several existing benchmarks. In the first case, we compared our automatic kernel detection against an existing OpenMP version of the tested benchmarks. We had a 95% success, with 3% of the misses being false negatives due to manually introduced constraints. As for the static scheduling algorithm, our predicted schedules are usually in the top 5%.

We will expand the work done on the static scheduling by introducing dynamic scheduling techniques. In order to test this techniques, we will integrate our work with a parallel programming framework, such as FastFlow [7].

Acknowledgments

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305, 'Network for Sustainable Ultrascale Computing (NESUS)'

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007- 2013) under grant agreement n. 609666 and by the Spanish Ministry of Eco- nomics and Competitiveness under the grant TIN2013-41350-P.

References

[1] The Khronos Group, The OpenCL Specification, http://www.khronos.org/ (Sep. 2014).

[2] Wienke, Sandra et al., OpenACC: First Experiences with Real-world Applications, Proceedings of the 18th International Conference on Parallel Processing, Euro-Par'12.

[3] Bertolli, Carlo et al., Coordinating GPU Threads for OpenMP 4.0 in LLVM, Proceedings of the 2014 LLVM Compiler Infrastructure in HPC, LLVM-HPC '14.

[4] B. Kolpackov, C++11 generalized attributes, apr, 2012.

[5] Nugteren, Cedric and Corporaal, Henk, Bones: An Automatic Skeleton-Based C-to-CUDA Compiler for GPUs, ACM Trans. Archit. Code Optim., January 2015.

[6] Johannes Doerfert and Clemens Hammacher and Kevin Streit and Sebastian Hack, SPolly: Speculative Optimizations in the Polyhedral Model, jan, 2013.

[7] Danelutto, Marco and Torquati, Massimo, Structured Parallel Programming with "core" FastFlow, 2015.

[8] J. Daniel Garcia et al., Static partitioning and mapping of kernel-based applications over modern heterogeneous architectures, Simulation Modelling Practice and Theory, 2015.

[9] Sanchez, Luis Miguel et al., A Comparative Study and Evaluation of Parallel Programming Models for Shared-Memory Parallel Architectures, New Generation Computing, 2013.

[10] Marco Danelutto et al., Introducing Parallelism by using REPARA C++11 Attributes, 2016, ACCEPTED, PENDING PUBLICATION.

[11] Rafael Sotomayor et al., Automatic CPU/GPU Generation of Multi-versioned OpenCL Kernels for C++ Scientific Applications, High-level Parallel Programming and Applications, 2015, ACCEPTED, PENDING PUBLICATION.