ab-Stream: A Framework for programming Many-core

Abstract. The common approach to program many-core processor is to write processor-specific code with low level APIs for different processors, which could achieve good performance but would result in serious portability issues; programmers are required to write a specific version code for target architecture. Therefore, we present ab-Stream, an extensible framework for programming many-threaded processor based on SUIF Intermediate Representation. ab-Stream abstracts many-core many-threaded processor into a unified architecture and ab-Stream program is an OpenMP-like program with different directives for many-core processor. Furthermore, a prototype of ab-Stream was implemented to map ab-Stream programs into many-core GPU. Experiments show that our implementation can execute transformed code correctly and efficiently on CUDA-enabled GPUs. Furthermore, performance of ab-Stream version code produced by our prototype can outperform original GPU code and is close to hand-optimized GPU code.

Keywords: Many-core, ab-Stream, Intermediate Representation, GPU

1. Introduction

Recently, many-core architectures are increasingly used in client computing systems, such as desktop computers and notebooks, are frequently equipped with one multi-core CPU and one many-core GPU. Accordingly, a lot of traditional scientific applications have been ported to many-core architectures such as GPU [1-2], CELL [3,4] and Imagine [5,6]. However, there are still plenty of applications that are reluctant to be ported to these many-core architectures because of difficulties of re-writing programs for Specific architecture.

To cure the pain of re-writing multiple copies of code, an ideal programming model for these architectures should be architecture-independent. OpenCL is proposed to ensure source level portability among GPU, CELL and CPU [7], while develop OpenCL programs should use low level APIs. Similarly, MapCG [8] provides higher APIs to program these architectures based on MapReduce model, however, MapCG loses a few low level hardware features which are useful for performance optimization.

Differently, ab-Stream is proposed for programming different architectures using C-like language with architectural optimization would be appended by adding operations on intermediate representation (IR) of ab-Stream program. In practice, architecture-independent ab-Stream programs are transformed into ab-Stream IR similar to SUIF IR and then compiled into architecture-dependent executable code accordingly to target architecture with architecture optimization would be appended into ab-Stream IR.

In proposed ab-Stream framework, programmers would dedicate to parallel algorithms instead of side-burdens incurred by emergent architectures. Without losing generality and availability, we choose many-core CUDA-enabled GPU as example and implemented an architecture optimizer and a compiler backend for CUDA-enabled GPU. But the methodology described in ab-Stream framework can be also applied to other many-core processors in ease by appending architecture-dependent compiler backend and optimizer for target architecture.

In the rest of the paper, we first introduce related works including SUIF Intermediate Representation and programming models for many-core processors. In Section 3, we would describe ab-Stream architecture and programming framework. A prototype of ab-Stream implementation would be detailed in Section 4. Section 5 evaluates ab-Stream programming framework and Section 6 concludes the paper.

2. Related work

2.1 SUIF

SUIF (Stanford University Intermediate Format) system is compiler infrastructure based on program IR, which is motivated by maximizing code reuse by providing useful abstractions and frameworks. SUIF IR is a language-independent and architecture-independent program representation, which would hold maximal generalities for architectures and preserve essential program information. Therefore, SUIF provides an excellent set of flexible modules and machine-dependent optimizations.

Accordingly, we would add new nodes to generate extra IR and extend passes or modules to transform and optimize program behaviors and data structures for building a prototype of ab-Stream based on SUIF system.

2.2 Programming Models for Many-core

In the past few years, many-core processors have emerged in traditional scientific applications [1-6], and future mainstream microprocessors will likely integrate specialized processor, such as GPUs [9].That is because many-core processors would attain much higher peak performance. However, differences in instruction set architecture and programming model make it hard to execute existing code directly and efficiently on many-core architectures.

 Consequently, new programming models have been proposed to program many-core processors, such as CELLs [10], Brook [11], CUDA [12], BSGP [13] and StreamC/KernelC [14]. And emerging programming modes are dominated by many-core GPUs and CUDA gains most popularity in programming NVidia many-core GPUs. However, these programming models are architecture-specific. In order to bridge the gap among architectures, some new programming modes are proposed.

OpenCL[15] tries to provide unified view of ISA between CPU, GPU and other many-core processors. It enables the programmer to write the same kernel code to execute on different processors including CPU and GPU. However, develop OpenCL programs should use low level APIs.

Similarly, EXOCHI[16] and Merge [15] try to provide a uniform framework to program heterogeneous many-core System. But, they require programmer to write different versions code for different architectures.
Differently, MapCG [8] provides source code level portability between different architectures such as CPU and GPU based on MapReduce model. However, it loses a few low level hardware features which are useful for performance optimization.

Accordingly, we present ab-stream framework to program many-core processors based on program IR with runtime optimizer. Theoretically, programs in ab-stream IR format could be efficiently mapped into different architectures on demand as long as target compiler and optimizer have been added into ab-stream framework.

3. Overview of ab-stream

3.1 ab-stream Architecture

Architecturally, ab-stream architecture simplifies unified architecture of OpenCL as shown in Fig. 1.

![Fig. 1 ab-stream architecture](image)

As illustrated in Fig. 1, ab-stream abstract many-core processors into a unified architecture, in which one many-core processor is divided into one or more thread kernels(kernel), which further are divided into one or more thread groups(group) and each group consists of one or more thread cores(core). And memory hierarchy in many-core processors is composed of private memory attached into core, shared memory shred between cores in same group and one good-sized global memory for all cores.

3.2 ab-stream Pipeline

Fig. 2 illustrates the pipeline of ab-stream system, in which ab-stream program would be compiled into ab-stream IR using ab-stream IR compiler, and then ab-stream IR could be passed into runtime optimizer before generating target version code. Many-core architecture is high performance architecture and is also a memory-bound architecture, especially for CUDA-enabled GPU [12, 17]. So, runtime optimizer should focus on memory optimization. Currently, we have deployed a runtime optimizer for CUDA-enabled GPU, in which several useful passes are included, as listed in Table 2.

<table>
<thead>
<tr>
<th>Type</th>
<th>Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement</td>
<td>KernelCallStatement</td>
<td>Function calling</td>
</tr>
<tr>
<td>Symbol</td>
<td>global</td>
<td>Directives</td>
</tr>
<tr>
<td></td>
<td>shared</td>
<td></td>
</tr>
</tbody>
</table>

These language constructs would be described in one hoof file, which would contain new classes derived from basic classes in SUIF to represent new nodes. And the hoof file parsed and translated by a macro processor called smgn [16] to generate ab-stream IR.

4. Design and implementation

In order to validate ab-stream framework, a prototype of ab-stream was implemented. Currently, an ab-stream IR compiler has been deployed. Without losing generality and availability, we also implemented a runtime optimizer and a target compiler for CUDA-enabled GPU.
5. Experimental results

In order to validate ab-Stream framework, we evaluated our prototype of ab-Stream on CUDA-enabled GPUs. In this section, we first briefly introduce the experiment setup and benchmarks. Then we show performance evaluation and comparison is presented.

5.1 Experiment Setup

The experimental configuration is provided as follows.
(1) Intel Core2 Quad 2.33 GHz, 4GB main memory.
(2) GeForce GTX280, 1GB video memory or global memory.

The selected benchmarks are from Parboil developed by UIUC [18], as listed in Table 3. There are a C version code and a CUDA version code (original version) for each application in Parboil benchmark suites. Hence, we customize an ab-Stream program based on C version code for our prototype of ab-Stream, and hand-tune CUDA version code (hand-tuned version) for optimization.

5.2 Performance Evaluation

We test original version code, ab-Stream version code produced by our prototype (ab-Stream version) and hand-tuned version code for each application in Parboil benchmark suites on CUDA-enabled GTX280, and performance comparison is demonstrated in figure 3.

![Normalized running time](image)

Fig. 3 Performance comparison

Practically, ab-Stream version code produced by our prototype can execute correctly on GTX280, and the performance of ab-Stream version code can outperform original version code and can be close to hand-tuned version code, as illustrated in figure 3, in which application execution time are normalized with logarithm function for holding clear comparisons of all applications.

6. Conclusion

With the increasing dominance of many-core architectures, an idea programming model is that writing program once using C-like high language and running program everywhere. Accordingly, we abstract many-core processors into ab-Stream architecture, and introduce ab-Stream programming framework, in which application programmer could write ab-Stream program using OMP-like language, system programmer could customize runtime optimizer and target compiler for specific many-core processor in ab-Stream framework. So that ab-Stream IR could be mapped into target processor efficiently.

Furthermore, we have built a prototype of ab-Stream system, in which a runtime optimizer and a target compiler for CUDA-enabled GPU have been deployed. Experimental results demonstrate that ab-Stream version code produced by our prototype can outperform original GPU code and is close to hand-optimized GPU code.

Acknowledgments

This work is partly supported by the National Grand Fundamental Research Foundation of China under Grant No. 2007CB310901 and 2012AA010909, National Natural Science Foundation of China under Grant No. 60803041 and NSFC-61025009, the Innovation Program for Excellent Graduates Foundation of national University of Defense Technology of China under Grant No. B090603, and Hunan Provincial Innovation Foundation for Postgraduates under Grant No.CX2010B031. 

REFERENCES


[18] CUDA benchmark suite.


Authors:
Xinbiao Gan received MS degree in computer system architecture from National University of Defense Technology of China in 2008. He is currently pursuing PhD degree in computer system architecture from National University of Defense Technology of China. His research interests include High performance computing, Computer architecture, GPGPU and Compiler Optimization. E-mail: xinbiaogan@163.com.