Xinbiao GAN[1, 2], Zhiying WANG[2], Li SHEN[2], Qi ZHU[1, 2]

State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410073, China (1), School of Computer, National University of Defense Technology University, Changsha 410073, China (2)

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.*

***Streszczenie.** Zaprezentowano szkielet (framework) ab-Stream do programowania wielordzeniowych procesorów. System bazuje na formacie SUIF (Standford University Intermediate Format). (**ab-Stream – struktura do programowania procesorów wielordzeniowych**)*

Keywords: Many-core, ab-Stream, Intermediate Representation, GPU
Słowa kluczowe: procesory wielordzeniowe, programowanie.

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 t 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 CELLSs [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[7] 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[9] 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.

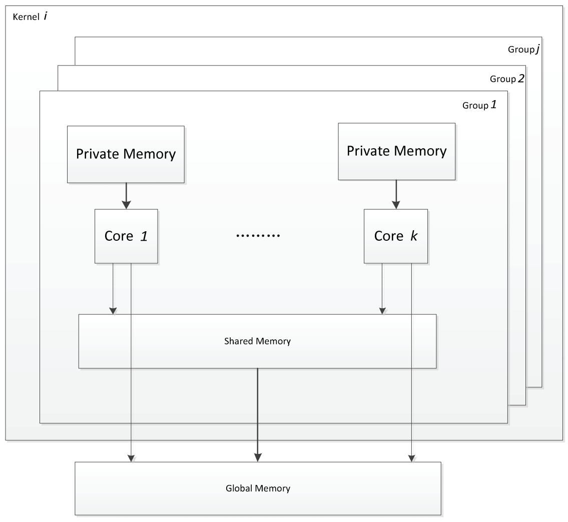


Fig. 1 ab-Stream architecture

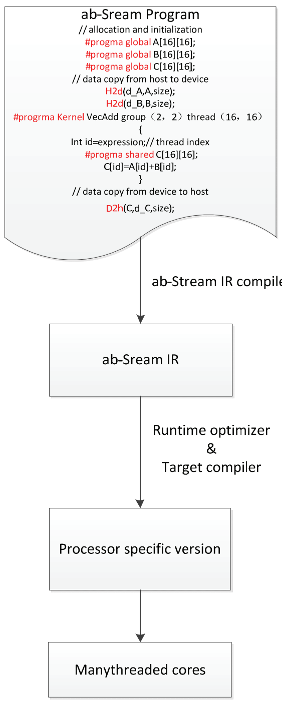
As illustrated in Fig. 1, ab-Stream abstract many-core processors into a unified architecture, in which one many-core processor is divided into one or more thread kernels(kernel), which further are divided into one or more thread groups(group) and each group consists of one or more thread cores(core). 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 for transformation and optimization on demand for specific processor, at last architecture-dependent version code could be generated for target processor. Practically, architecture-dependent runtime compiler and target compiler could be configured and added into ab-Stream framework if needed.

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.



```
ab-Sream Program
// allocation and initialization
#progma global A[16][16];
#progma global B[16][16];
#progma global C[16][16];
// data copy from host to device
H2d(d_A,A,size);
H2d(d_B,B,size);
#progrma Kernel VecAdd group（2，2）thread（16，16）
{
Int id=expression;// thread index
#progma shared C[16][16];
C[id]=A[id]+B[id];
}
// data copy from device to host
D2h(C,d_C,size);
```

ab-Stream IR compiler
ab-Sream IR
Runtime optimizer & Target compiler
Processor specific version
Manythreaded cores

Fig. 2 ab-Stream pipeline

4.1 Frontend Extended

The ab-Stream IR compiler extended SUIF compiler to define language constructs including kernel calling and directives for ab-Stream as shown in Table 1.

Table 1. Extended constructs for ab-Stream

Type	Case	Description
Statement	KernelCallStatement DeviceCallStatement	Function calling
Symbol	Kernel global shared	Directives

These language constructs would be descripted 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.

4.2 Pass Extended

As descripted in Fig. 2, ab-Stream IR usually would be passed into runtime optimizer before generating target version code. In our prototype of ab-Stream, a runtime optimizer is composed of useful passes, and a pass is a module with a particular structure used to traverse SUIF trees for optimization and transformation.

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.

To write a pass, we must derive from the *Pass* class which is derived from the *Module* class. In our runtime optimizer, we built optimizer based on pass class derived from *PipelineablePass* instead of directly from *Pass*, in which *do_file_set_block* method must be included, and the *do_file_set_block* method will walk SUIF tree and perform optimizations and transformations with the aid of an Iterators class. In Practice, other useful pass could be added into your runtime optimizer on demand.

Table 2. Main pass and library extended

Type	Case	Description
Library	H2d	data transfer between host(CPU) and device(many-core)
	D2h	
Pass	DataflowAnalysis	identify data dependence
	FalseSharingElimination	eliminate false sharing of data
	DataRelayout	redeploy data into advisable space according to data structure[17]

4.3 Backend Extended

In order to generate processor-specific version code, ab-Stream IR should be parsed correctly with the aid of target compiler. Fortunately, there is a compiler backend named s2c to parse SUIF IR for generating target version code in SUIF system [16]. So that one target compiler could be built for your target processor by extending *s2c*.

As detailed in ab-Stream IR compiler, architecture-dependent language constructs are extended into system frontend, currently, those language constructs have been declared in *s2c* according to prescriptive format for parsing ab-Stream IR correctly in our target compiler.

Table 3. Benchmarks

Application	Description
Cp	A benchmark for computing the columbic potential at each grid point over one plane in a 3D grid in which point charges have been randomly distributed.
Mri-Fhd	Computation of a matrix FHd used in a 3D magnetic resonance image reconstruction algorithm in non-Cartesian space. The vector d represents the scan data. The vector FH represents the scan trajectory.
Mri-Q	Computation of a matrix Q, representing the scanner configuration, used in a 3D magnetic resonance image reconstruction algorithm in non-Cartesian space.
Pns	A benchmark for Petri-net Simulation. This benchmark implements a generic algorithm for Petri net simulation.
Rpes	A benchmark which implements a Rys Polynomial Equation Solver.
Sad	This benchmark computes SADs for pairs of blocks, where an SAD is one metric for how closely two images match.
Tpacf	Tpacf The Two Point Angular Correlation Function (TPACF) describes the angular distribution of a set of points. This benchmark computes the angular correlation function for a data set of astronomical bodies.

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.

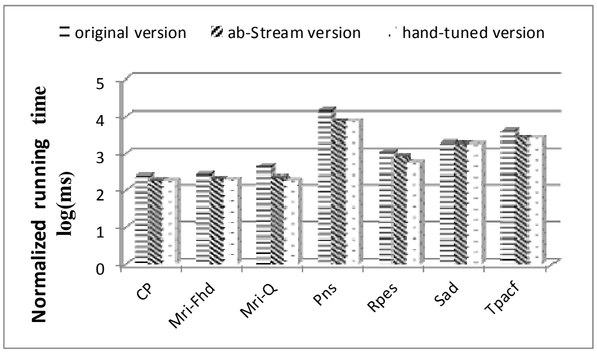


Fig. 3 Performance comparison

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 2012AA010905, 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

[1] Bo Zhang, Zheng-Hui Xue, Ren Wu, et.al, "Acceleration of FDTD algorithm based on GPU computing", Chinese Journal of Radio Science, Vol.26, No.1, pp108-112, 2011.

[2] K.V. Kalgin, "Implementation of algorithms with a fine-grained parallelism on GPUs", Numerical Analysis and Applications, Vol.4, No.1, pp 46-55, 2011.

[3] F. Dehne, G. Hickey, A. Rau-Chaplin, et.al, “Parallel catastrophe modeling on a Cell/BE”, International Journal of Parallel, Emergent and Distributed Systems, Vol.25, No.5, pp 401-410, 2010.
[4] H. Vandierendonck, S. Rul, M. Questier, et.al, “Experiences with parallelizing a bio-informatics program on the Cell BE”, Proceedings of the third International Conference on High Performance Embedded Architectures and Compilers (HiPEAC'08), pp 161-175, 2008.
[5] Hai-yan Li, Chun-yuan Zhang, Li Li, et.al, “Transform coding on programmable stream processors”, Journal of Supercomputing, Vol.45, No.1, pp 66-87, 2008.
[6] Xue-Jun Yang, Li-Fang Zeng, Yu Deng, et.al, “Optimization approaches of organizing streams on imagine processor”, Chinese Journal of Computers, Vol.31, No.7, pp 1092-1100, 2008.
[7] Khronos Group. OpenCL specification.
[8] Chuntao Hong, Dehao Chen, Wenguang Chen, et.al, “MapCG: Writing parallel program portable between CPU and GPU”, Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT'10), pp 217-226, 2010.
[9] P. H. Wang, J. D. Collins, G. N. Chinya, et.al, “ EXOCHI: Architecture and programming environment for a heterogeneous multi-core multithreaded system”, Proceedings of International Conference on Programming Language Design and Implementation (PLDI'07), pp 156–166, 2007.
[10] Pieter Bellens, Josep M Perez, Rosa M Badia, et.al, “CellSs: a programming model for the CELL BE Architecture”, Proceedings of ACM/IEEE Conference on Supercomputing (SC'06), pp 86–98, 2006.
[11] Ian Buck, Tim Foley, Daniel Reiter Horn, et.al, “Brook for GPUs: stream computing on graphics hardware”, ACM Transaction on Graphics, Vol.23, No. 3, pp 777–786, 2004.
[12] NVIDIA, “CUDA programming guide”, 2008.
[13] Qiming Hou, Kun Zhou, Baining Guo, “BSGP: Bulk-synchronous GPU programming”, ACM Transactions on Graphics, Vol.27, No.3, 9, 2008.
[14] Peter Matton, et.al, Imagine Programming Systems User’s Guide, 2002.
[15] Michael D. Linderman, Jamison D. Collins, Hong Wang, et.al, “Merge: A programming model for heterogeneous multi-core systems”, ACM SIGPLAN Notices (ASPLOS'08), Vol.43, No.3, pp 287-296, 2008.
[16] David L Moore, “The SUIF Programmer Guide”, 1999.
[17] Xinbiao Gan, Zhiying Wang, LiShen, et.al, “Data Layout Pruning on GPU”, Applied Mathematics &Information Sciences, Vo.5, No.2, pp 129S-168S, 2011.
[18] CUDA benchmark suite.
[19] http://www.crhc.uiuc.edu/impact/cudabench.html. 2010.

***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.coml