Journal of Theoretical and Applied Information Technology
15th September 2017. Vol.95. No.17

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A NOVEL FRAMEWORK FOR SERIAL PROGRAM FULLY AUTO PARALLELIZATION BASED ON OPENACC

[1]WANG XIAORUI, [2]JIANG HUIFANG, [3]CAI DA

[1] School of Computer Science and Technology, China University of Mining and Technology, Jiangsu, Xuzhou 221116,China

E-mail: [1]445023924@qq.com, [2]jhf@cumt.edu.cn,[3]neko1990@gmail.com

ABSTRACT

On the basis of OpenACC programming standard (for open accelerators), a new framework was proposed, whose name is GENerate OpenACC, or GENACC for short. GENACC can automatically accelerate the serial code. The static program of the source code are analyzed aimed to recognize the hot snippets and analyze the computation property. And finally OpenACC directives are added to the source code in order to accelerate the serial code. According to the experiments on the NPB test set, the results show that GENACC can accurately produce compiler directive, and the source codes show fine performance on different amount of data.

Keywords: *OpenACC, Auto-Parallelization, LLVM translator, Inheritance code, NPB test set*

1. INTRODUCTION

As the representative of the GPU products, NVIDIA and AMD has rapidly developed. Nowadays, an ordinary video card has the computing performance like a super computer. CUDA (Compute Unified Device Architecture) is a programming platform for GPU parallel computing under the trend of CPU / GPU heterogeneous computing, and cites different parallel programming models and instruction sets. The more and more general language OpenCL was introduced later, which could support more processors and do accelerator operation on the bottom of the hardware compared with the CUDA. But the development is more difficult.

For a large number of inheritance codes, the development process has cost a lot. Years of stable usage also proved the reliability of these codes. If it has been replaced, more costs need to be paid. Also, the reliability of the new code could not be guaranteed. Therefore, only the parallelization could speed up the program in the case that computing performance of single-core processor is difficult to upgrade. How can this code take advantage of the ability to accelerate the device (including GPU, FPGA, DSP, MIC) to achieve a higher rate has become a hotspot [1]. Among the existing special acceleration equipment, GPU has the characteristic of large computing density and high bandwidth. And it has an advantage of processing performance, cost performance and programmability. With the combination of the GPU and general-purpose processors, more and more applications can achieve high-performance computing to greatly optimize computing efficiency.

For the past GPU programming, developers need to understand the underlying structure of the GPU, and to manually implement some of the details, such as data transmission, computing task allocation, cache structure optimization [2]. It brought software development a great burden. On the other hand, GPU hardware development is very rapid. The implementation of the software may not be suitable for new hardware. So development costs and maintenance costs are very high. If inheritance codes can be accelerated by GPU with automatic parallelization method, it will greatly benefit developers. Under such a background, graphics chip maker NVIDIA and compiler company PGI, CRAY proposed the OpenACC standard together. Developers only need to use compiler instructions to indicate parallel processing region and the transfer data in the source code. While such operations as data block and transmission, the scheduling task are left to the OpenACC compiler to achieve during runtime. The same instructions can be run on a variety of general-purpose parallel processors, such as multi-core CPUs or GPUs. Because it can achieve hardware independence. And the time costs less. So more developers choose this way.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2. RELATED WORK

The appearance of the CUDA architecture abstracted based graphics operations in the graphics and became a more powerful data processing tool for developers in 2006. Volkov and Kazian in the University of California used Cooley-Tukey framework to solve FFT computing problems. The computing speed of core GPU in the G80 was close to the equipment peak 144GFlop/s, which is 18 times of the best CPU results [3,4]. Kaushik et al. achieved a template (nearest neighbor) computation rate of 36 GFlop / s by parallel discovery and caching optimization, which is 5.3 times of the fastest CPU speed [5]. In 2004, a new method for calculating the radiometric calculation using GPU was proposed in the Tsinghua University. Also Jacobi iterative method was proposed, which could solve the linear equations quickly [6]. GPU acceleration was attempted to use in the nuclear power simulation in the literature [7], access to 10-20% performance improvement.

Automatic parallelization technology of serial program can be divided into automatic vectorization and automatic parallelization. Automatic vectorization technology combined multiple data operations through the SIMD instruction to improve the speed. It has been very mature to achieve in the modern compiler. Literature [8] summarized the compilation optimization and the related optimization algorithm of SIMD instruction automatic vectorization. According to whether developers need to mark compiled guidance statement, it can be divided into semi-automatic parallelization and fully automatic parallelization. Semi-automatic parallelization marked parallel compilation guidance statement in serial code and then compiled them into parallel executables. Guidance statements are OpenMP, OpenACC for accelerators, OpenHMPP for heterogeneous systems, and so on. Fully automatic parallelization does not require the participation of developers. The system can analyze the program and generate the parallel execution on the basis that the source program has not compiled guidance statement. Previous research included Polaris compiler from the University of Illinois [9] and SUIF compiler architecture from the Stanford University [10].They are mostly the nature of the compiler. Domestic early research work includes an automated parallelization system Agassiz developed by the Fudan University Parallel Processing Institute, Purdue University and the University of Minnesota Computer College [11]. Agassiz made a certain degree of supplements in the system versatility and scalability compared to other compilation systems in the same period (such as SUIF, Polaris). In addition, Agassiz system can effectively integrate a variety of parallel technology, and can effectively integrate multiple languages and back-end instruction layer compiler. Zhejiang University also designed an interactive parallel translation system and compilation method [12]. According to the information obtained from the automatic analysis, combined with the interactive information provided by the user, the serial application in the multi-core architecture could get good performance. A Cyclic Workload Evaluation Algorithm Based on Program Static Analysis was proposed in the literature [13], which used SUIF framework and established GPU parallel overhead model based on CUDA. Based on LLVM / Clang, CTMP was designed to convert serial code into OpenMP programs in [14].

One of the reasons why OpenACC was introduced was because of its openness. As a standard for development, OpenACC is a compiler instruction set that allows to specify the code loop and code area to be unloaded from an host CPU to an accelerator in standard C / C ++ and Fortran languages. OpenACC also has portability across operating systems, accelerators, or host CPUs. Developers can use instructions that allow the same code to run on different parallel hardware, such as multicore CPUs, GPUs, or other hardware that has compiler support. In the C or C ++ language, we use the #pragma instructions, and in the Fortran language, it is the line compiler directive! $acc. Compiler instructions have a portable portability feature that instructions can be ignored. Even if the code compiler platform does not support OpenACC, it can still compile the code and run well. The code seems to never exist. OpenACC executes parallel code on the accelerator, abstracts out the storage and implementation of two parts, and sets run-time behavior through runtime libraries and some OpenACC-specific environment variables.

3. ARCHITECTURE OF GENACC

A novel fully automated parallelization tool, Generate OpenACC Directive (GENACC) was designed in this paper, which primarily implements two purposes: to identify the parallelizable parts of the program and to automatically mark OpenACC instructions.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.1 Overall Design of GENACC

The complete automatic parallelization of the serial code is divided into two steps. The first step is to identify the parallelizable code area. The second step is to program the parallelizable region. Through an overview of the existing automated parallelization tools, we have found that there are already some fully automated parallelization tools. They enter serial code without instruction in order to generate an executable file which can take advantage of GPU operations. However, the quality of the tools are not stable. Some programs cannot guarantee the correctness, the implementation may be wrong; or generate the correct parallel implementation, but the operation time has increased. In addition, some compilers can automatically generate OpenMP instructions or Guided Auto Parallel. But it have not yet developed for OpenACC.

In view of the above problems, the GENACC framework firstly identified automatically the code part that needs to be processed by GPU parallelism in the source code, then used the existing semi-automatic parallelization tool to handle the loop optimization and generate the executable file. In GENACC, these two steps can be separated independently. This focuses of paper are the implementation of automatic parallelization and the generation of OpenACC instructions. For the relevant part of the accelerator's underlying hardware optimization, it was handled by the corresponding OpenACC compiler.

To better implement different functions, we used two compilers in the GENACC: the LLVM compiler suite and the OpenACC compiler. The LLVM compiler suite includes front tools Clang and Polly for the analysis of multiple loops. Clang is a subproject of LLVM and is a compiler front tool for C, C ++, and Objective-C. Polly is an optimizer based on LLVM which enables localized cache optimization based on polyhedral models and automatic vectorization. In addition, LLVM includes multiple subprojects. Its core is to handle analysis, transformation, optimization and compilation of LLVM intermediate code. Because LLVM uses a good modular design, it has a strong scalability. The opt command is a modular LLVM optimizer and parser which makes it easy to access the generated abstract syntax tree (AST). Users can analyze programs, extract information, or convert custom intermediate code according to their own needs.

This article will use the LLVM compiler suite to complete the static analysis of the source program, and through the program analysis to extract the cost model required parameters. The OpenACC compiler implements specific back-end optimization and execution of files.

3.1.1 GENACC Functional Framework

The functions of the GENACC are divided into four parts: analyzing the serial program, obtaining the relevant hardware parameters, adding the OpenACC guidance instruction to the hotspot code, and issuing the executable program file. As shown in

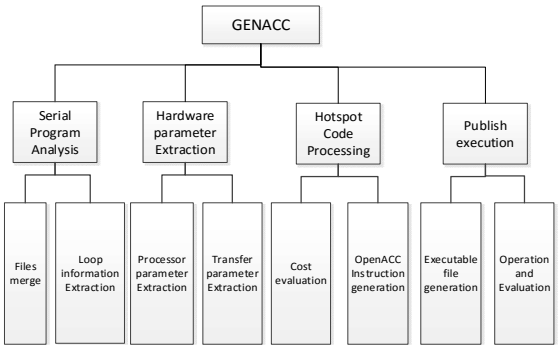

Figure 1:

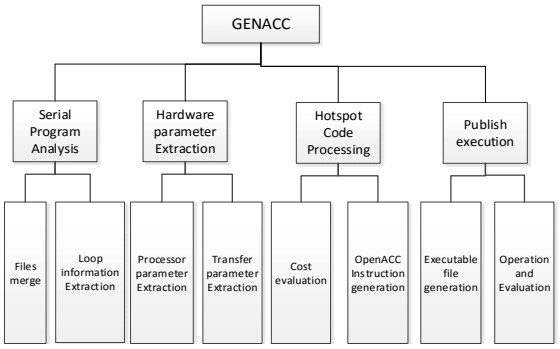

Figure 1 GENACC Framework

Serial Program Analysis module is to compile the target serial code into LLVM Intermediate code. If there are more than one program file, it will be merged. We need to extract the necessary cycle information as a hot code in the compilation process. The hardware parameter extraction module obtains the transfer parameters between the processor parameters and the CPU-GPU. The hotspot processing module contains the cost evaluation and the OpenACC instruction generation. It is used to evaluate the cost of the operation, add the OpenACC directive and generate the patch file. The distribution execution module executes the source file after the patch is added, and evaluates the execution result.

3.1.2 GENACC Workflow

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

According to the above function, there are three stages in the process of serial code completely automatic parallelization with the GENACC framework.

The first stage is the preparation stage. Before analyzing the original serial code, we need to do some preprocessing work. The target serial code should be transformed into LLVM intermediate code through the Clang. If the program consists of multiple files, we should firstly compile the individual files into the corresponding LLVM target file, and then link them to a large LLVM intermediate code. This can be done constant diffusion, function embedded in the global scope to get more accurate alias analysis results. In the process of conversion, the required loop information is extracted using the line static program analysis. The parallelizable code can be separated from the serial code, which we call as hot code.

The second stage is the evaluation process stage. The evaluation model mainly combines the Roofline model and the LogGP model to model the operational energy consumption in two layers, and establishes a new computational cost model. Then, based on the given cost configuration, the cost of the hotspot code is evaluated on the CPU and the GPU respectively. If there are obvious advantages on the GPU, the code should be parallelized and added OpenACC instructions. This stage will analyze the data dependence of the loop, extract the loop iteration number and the instructions of the loop body, which will be input to the computational cost model. Finally, a patch file is generated. And the corresponding files are modified by the Linux patch.

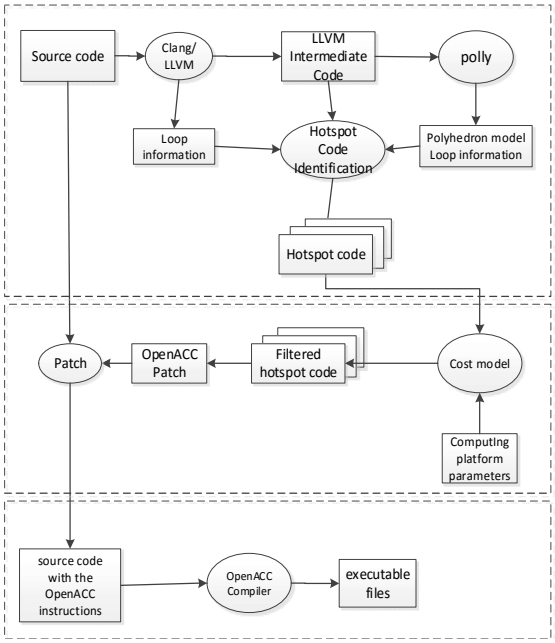

Figure 2 GENACC Workflow

The third stage is the implementation stage. Firstly, the patched source code is compiled through the OpenACC compiler. Then, the generated target files and related data should be transferred to the target machine and executed. This stage will perform all loop-related compiler optimizations, including the deletion or elevation of common subexpressions. It also needs to use loop of acceleration to achieve data transmission and parallelization because of the OpenACC instructions in the previous stage. Three stages of the specific process are shown in *Figure 2*.

3.2 Parallel Identification Method

Parallel recognition, also known as parallel detection, is a common technique in program analysis. The loop is the primary parallel source in the serial program. So we identify the parallel by identifying the loops in the program. Based on the code analysis in the LLVM tool, a new algorithm of hot spot code separation was realized in this paper. The following steps are as follow:

Step1: The source code was compiled into the LLVM intermediate code, which was realized by the LLVM C/C++ front tool Clang. To get more accurate program analysis results, we compile multiple files from the source program into a large LLVM intermediate code without any compilation optimizations. Some of the most confusing source code was modified to match the specific criteria using the Clang format tool. Although the original code has been largely modified, this step does not affect the semantics of the program.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Step2: Single-layer loop workload analysis. On the basis of the unoptimized LLVM intermediate code, opt is used to load a custom LoopInfoDump analysis process to get a basic loop information. The information includes the start and end conditions for all loops. This can analyze the workload of a single layer loop.

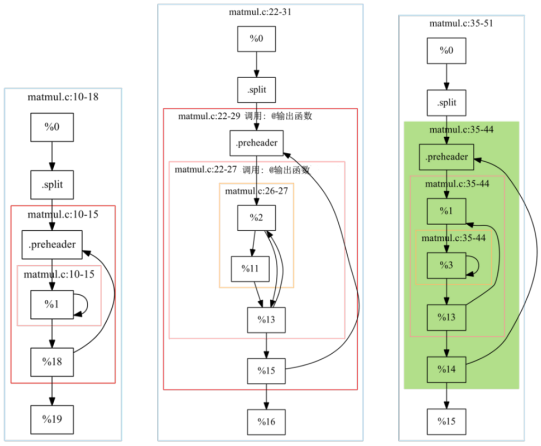

Figure 3 Static Control Part of the Matrix Multiplication Program

extraction. The polygon model of the source program was obtained by the polly tool. As is shown in *Figure 3*.

This is the matrix static control part obtained by the analysis of the matrix multiplication example program. The left is the analysis result of the function init_array. There is a double loop in line10 to line 15. And the two nested boxes inside are correspond to the two loops; the middle is the analysis result of the function print_array. There is also a double loop in line 22to line 29. The outer loop would call fprintf once when each iteration and output a blank line. On the right is the analysis result of the entrance function main. Tri-loop is from line 35 to line 44 in the program to achieve matrix commensurate function.

Step4: Hotspot code information generation. Information obtained from above two steps is written into temporary files for the next stage. This information includes the loop load, the number of the loops. The number of loops can be an expression of the input parameter, since it is not certain in some cases.

3.3 Automatic Labeling of OpenACC Instructions

Step3: Multi-layer loop workload analysis. This step is complementary to the previous step to complete the nesting multi-layer loop information

GENACC has gotten multiple hotspot code snippet information from the previous section. This section will use the computational cost model to evaluate the cost of running these hotspots on the CPU and GPU and choose the corresponding OpenACC guidance instructions which run on the GPU and generate from the appropriate portion. Steps are as follows:

Step1. Initialization of the computational cost model. *Table 1 Parameters of the computational cost modelTable 1* is the parameters that the model requires from the target computing platform. It is obtained from the performance test program.

Step2. Cost evaluation. When the size of the data to what extent, the use of GPU revenue will be greater than the transmission costs. The computing cost of the hotspot code fragment is evaluated. And the OpenACC guidance instructions are added only if its cost can be reduced when executed on the GPU. The hot codes obtained by the analysis were evaluated separately to screen out greater revenue hotspot codes with the GPU.

Step3. If there is no hot codes to accelerate with the GPU, that is, all parts of the program are not suitable for GPU to accelerate, then it should be exited directly.

Table 1 Parameters of the computational cost model

Parameters	Meaning	Access
Π	size of dataset	program analysis
$\Gamma(\Pi, ISA)$	instructions in the loop	program analysis
ω_1, ω_2	instructions on a specific schema ISA	program analysis

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$Overhead$	weight of time and power	Given
G_{CG}	CPU-GPU transmission overhead	test program
G_{GC}	interval when transmitting large chunks of data from main memory to video memory	test program
BW_{CPU}	interval when transmitting large chunks of data from video memory to main memory	test program
BW_{GPU}	main memory bandwidth	test program
f_{CPU}	graphics memory bandwidth	test program
f_{GPU}	CPU frequency	hardware parameters
$IdlePower$	GPU frequency	hardware parameters
TDP_{CPU}	power of the platform	hardware testing
TDP_{GPU}	CPU frequency thermal design power consumption	hardware parameters
N_{CPU}	GPU frequency thermal design power consumption	hardware parameters
N_{GPU}	CPU operational module numbers	hardware parameters

Step4. OpenACC instructions generation. The OpenACC directive consists of three kinds:

(1) OpenACC initialization instructions and shutdown instructions. First of all, OpenACC header files should be added at the beginning of the corresponding files. Next, the device initialization instructions, acc_init (device), should be written at the entrance of the function. In the same way, the device shutdown instructions should be added when the device is no more required after completing the calculation.

(2) Data transfer instructions, "pragma acc data". The hot program also has input and output. And the data is stored in the main memory. So the input data needed to be marked with copyin to copy into the memory. While the output data needed to be marked with copyout.

(3) Parallel instructions, "pragma acc loop", "pragma acc kernels". Appending this instructions to specific loops means making OpenACC compiler accelerate the current loop and construct kernel functions executed on GPU.

Step5. Patch files generation. The relevant instructions generated by the previous step needed to be modified in some form. Open source patch programs just meet this demand. Where OpenACC instructions generated was recorded by GENACC in the format of the patch files. As shown in *Figure 4*, it is the corresponding patch file of matrix multiplication program:

```
@@ -1,3 +1,6 @@
+#ifdef _OPENACC
+#include <openacc.h>
+#endif
 #include <stdio.h>

 #define N 512
@@ -34,6 +37,12 @@
 {
     int i, j, k;
     init_array();
+    acc_init(acc_device_default);
+    #pragma acc data copyin(A[0:262144], B[0:262144]), copyout(C[0:262144])
+    {
+    #pragma acc kernels
+    {
+    #pragma acc loop independent
     for(i=0; i<N; i++)  {
         for(j=0; j<N; j++)  {
             C[i][j] = 0;
@@ -41,9 +50,12 @@
                 C[i][j] = C[i][j] + A[i][k] * B[k][j];
         }
     }
+    } // end of kernels
+    } // end of copy

 #ifdef TEST
     print_array();
 #endif
+    acc_shutdown(acc_device_default);
     return 0;
 }
```

Figure 4 Matrix multiplication of the serial program

Seen in the figure, there are three modifications in the source program. Each place has a template as "@@-beginning line of source file, end line of source file + beginning line of new file, end line of new file-@@". The symbol "+" in the first column indicates that the line is not existed in the source program and new added. The openacc.h header file was added in the first place and protected using conditional compiler. So that there will be no error when the compiler does not support OpenACC guidance. The device initialization instructions were added in the second to initialize the default device. Next, copyin illustrates that A and B need to transfer into memory and copy the results C back to the RAM. At the last place, acc_shutdown was added to take off the device before the function returning.

Step6. Generating the program with OpenACC instructions. This step can be accomplished through

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the Patch program. The target file should be set as the corresponding directory when the source code contains multiple files. Finally, the OpenACC compiler is called to compile the source code into an executable file.

4. EXPERIMENTS AND ANALYSIS

The NPB (NAS parallel Benchmark) test sets was used to test the functionality and performance of the GENACC architecture in this paper. GENACC was deployed on Linux relying on a number of Linux platform tools. The software and hardware parameters of the runtime environment are shown in the *Table 2* and *Table 3*.

Table 2 Experiment software version

Types	Version
Operating System	OpenSUSE 13.2
CUDA version	7.5
PGI compiler version	2016.03
LLVM version	3.8
gcc compiler version	4.8.3

Table 3 Experimental hardware Platform

Hardware	Platform1	Platform2
CPU Type	Core i7 4710MQ	Core i5 3340M
CPU Core Frequency/Ghz	2.5-3.5	2.7-3.4
CPU Operations Cores	8	4
CPU Access Bandwidth/GB/s	25.6	25.6
CPU Cache/MByte	6	3
GPU Type	GTX 850M	GT750M
GPU Operations Cores	640	384
GPU Core Frequency /Ghz	0.86	0.95
GPU Access Bandwidth /GB/s	80	80
CPU-GPU Bus Type	PCI-E3.0	PCI-E 3.0

NPB consists of five core programs and three computational fluid dynamics simulation programs. The EP (Embarrassingly parallel) program is used to calculate the Gauss pseudo-random numbers. EP is very suitable for parallel computing. Because it has few requirements for intercommunication between processors. And the results can often be the upper bound for a particular parallel system floating point performance. The MG (MultiGrid) program is to compute the discrete periodic approximation solution of the three-dimensional Poisson equation with four V-loop multiple grid algorithms. The CG (Conjugate Gradient) program is used to get the approximation of the minimum eigenvalues of large sparse symmetric positive definite matrices. And it represents the problem of non-structural style calculation and non-regular remote communication computing. The FT (Fast Fourier Transformation) program is used to solve the three-dimensional partial differential equation based on the rapid Fourier transform spectrum analysis. Also, it requires a lot of remote communication. The IS（Integer sort）program is a kind of bucket sort, which is used to sort the two-dimensional large integer and requires a large number of fully switched communications. The three computational fluid dynamics simulations include LU (lower upper triangular) which is used to solve block sparse equations based on symmetric super relaxation method, BT(Block Tri-Diagonal) which is used to solve the three-diagonal system, and SP which is used to solve the five diagonal system.

SP, EP, MG, CG, four procedures were successfully parallelized by GENACC in the serial version of the NPB test set. The comparison results are shown in *Figure 5*:

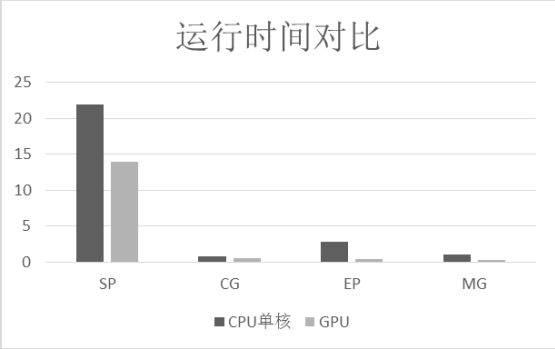

Figure 5 Runtime Comparison of GENACC Parallel Program and Serial Version

The automatic parallelization handler is 1.5 to 6 times faster than the original serial version. In addition, this article compares these five versions with the corresponding OpenMP programs. As is show in *Figure 6* that MG and CG have an advantage on the GPU only when the input data is larger.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

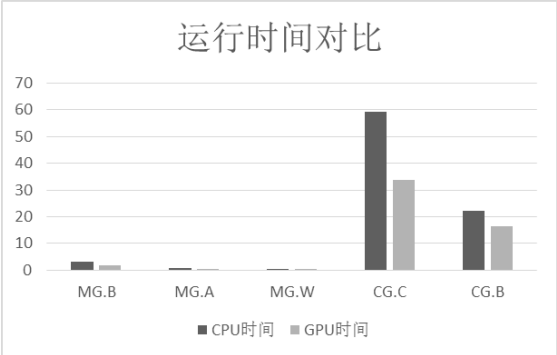

Figure 6 Runtime Comparison of GENACC Parallel Program and OpenMP Version

In these programs, although the A-level program of CG is 7% slower than the optimized OpenMP version, the others are 1.3 to 1.8 times faster.

5. CONCLUSION

This paper completes the fully automatic parallelization framework GENACC, which could parallel detect the serial code, optimize loops, analyze data dependency, generate the patch file of the source program, and ultimately generate the executable program used GPU. With experiments on the NPB test sets, compiler generated by GENACC guide commands to test. The experimental results show that the GENACC fully automatic parallelization system has been able to identify the parallelizable part on the serial code and can convert the source code into the code with the OpenACC instruction in order to realize the automatic parallelization of the serial code. The generated codes have a high readability. However, GENACC cannot deal with the situation that programming needs more storage. As a part of future research, the optimization of OpenACC code parameters need more study.

REFERENCE

[1] Liu Y, Lü F, Wang L, Chen L, Cui HM, Feng XB. Research on heterogeneous parallel programming model [J]. Journal of Software, 2014,25(7):1459−1475 (in Chinese).

[2] Wang H.F., Chen Q.K. General Purpose Computing of Graphics Processing Unit: A Survey[J].Chinese Journal of Computers,2013.36(4):757-772(in Chinese).

[3] Volkov V, Kazian B. Fitting FFT onto the G80 architecture [J]. University of California, Berkeley, 2008, 40.

[4] Volkov V, Demmel J W. Benchmarking GPUs to tune dense linear algebra[C]. Proceedings of the 2008 ACM/IEEE conference on Supercomputing (SC '08). IEEE Computer Society, 2008:1-11.

[5] Datta K, Murphy M, Volkov V, et al. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures[C]. Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press, 2008: 4.

[6] Hu W. and Qin K.H., A New Rendering Technology of GPU-Accelerated Radiosity [J] . Journal of Computer Research and Development, 2005, 42(6): 945-950(in Chinese).

[7] Wang X., Wu F., Zhang X., Feasibility of CPU/GPU heterogeneous computing in nuclear power plant simulator [J].Computer Applications. 2014, 34(S2): 73-77(in Chinese).

[8] Gao W, Zhao RC, Han L, Pang JM, Ding R. Research on SIMD auto-vectorization compiling optimization [J]. Journal of Software, 2015,26(6):1265−1284 (in Chinese).

[9] Blume B, Eigenmann R, Faigin K, et al. Polaris: The next generation in parallelizing compilers[C].Proceedings of the Seventh Workshop on Languages and Compilers for Parallel Computing. 1994: 141-154.Wilson R P, French R S, Wilson C S, et al. SUIF: an infrastructure for research on parallelizing and optimizing compilers[J]. Acm Sigplan Notices, 1994, 29(12):31-37.

[10] Wilson R P, French R S, Wilson C S, et al. SUIF: an infrastructure for research on parallelizing and optimizing compilers[J]. Acm Sigplan Notices, 1994, 29(12):31-37.

[11] Zheng B,Tsai J Y,Zang B Y,et al. Designing the Agassiz Compiler for Concurrent Multithreaded Architectures[C]. Proceedings of the 12th International Workshop on Languages and Compilers for Parallel Computing. Springer-Verlag,1999:380-398.

[12] Li Y., Sun X.X., Yuan X.Y. Interaction-based speculative thread-level parallelization [J].Application Research of Computers. 2010, 27(6):2123-2126(in Chinese).

[13] Wang T. GPU-based program analysis and parallelization research [D]. People's Liberation Army Information Engineering University, 2010(in Chinese).

[14] Zhang D.Y. Clang-based C language code parallelization conversion tool design and implementation [D]. Jilin University, 2015.