PARALLEL OPTIMIZATION OF ACCELERATOR TOOLBOX BY OpenMP AND MPI*

K. Wang#, C.M. Luo, S.Q. Tian, M.Z. Zhang, Q.L. Zhang, B.C. Jiang
SINAP, Shanghai, China

Abstract

Parallel computing adopted in the simulation program of accelerators improves the computational efficiency, especially tracking with multi-particle and multi-turn which always takes a lot of time. Through investigating the operational principle and running flow of Accelerator Toolbox(AT) embeded in Matlab for accelerator design and simulation, Open Multi-Processing (OpenMP) and Message Passing Interface (MPI) were studied on and adopted in the parallel computing. The key code of particle tracking in AT has been transplanted and test with OpenMP-MPI as well as estimated reasonably, which improve the calculation efficiency largely. It is indicated that OpenMP-MPI hybrid parallel structure is in good agreement with AT programs by avoiding the internal communication and improving load balance. A multi-core computing platform based on OpenMP-MPI has been developed and contributed to the deep optimization of accelerators. And it shows good extensibility, which could speed up by adding computing nodes.

INTRODUCTION

Accelerator Toolbox(AT), developed by Stanford Synchrotron Radiation Light source (SSRL), is a toolbox embedded in Matlab for accelerator design and simulation [1]. The experiment measurements agreed well with AT has been carried out using a plenty of functions and applications assisted by Matlab. In recently years, AT has been used for the design and operation at Shanghai Synchrotron Radiation Facility (SSRF), which is one of the advanced 3rd light sources worldwide [2, 3]. In order to calculate the dynamic aperture and optimize the lattice design of synchrotrons, computation-intensive algorithms are required. Based on the progress of the programme, we can improve the computational efficiency largely. Because of the abundant repeated calls of particle tracking functions used in AT, the computing speed will be postponed without parallel computing. It is expected that we propose the parallel optimization of AT [4].

The two common methods for parallel optimization are as follows. One is based on using GPU. The other one is using multi-core CPU with multi-thread. Due to the frequently data exchange between computer memory and GPU in particle tracking assisted by cache operations, the speed-up efficiency cannot be effective obviously with a few data such as a small number of particles. As a result, it is verified that parallel optimization with multi-core CPU is a better choice.

*Work supported by National Natural Science Foundation of China under Grant No. 11105214
wangkun2013@ sinap.ac.cn

OpenMP is a standard model of share memory computing, which supports C/C++ compiler. With a local qual-core computer, the speed-up can be almost 4 times. Based on the message passing model, MPI is used for a dual CPU server to speed up further. And it shows good extensibility that the speed grows linearly along with the number of computing nodes.

EXPERIMENTAL

Accelerator Physics and Optimization Analysis

Considering that a particle can be represented in AT with a point in 6-dimension phase space, particles transport from one accelerator element to the next can be computed by a second-order transport matrix. It is assumed that the particles are relativistic and there are no interaction with each other, which obtain the parallel computing request [5]. With particle tracking assisted by different *passmethod* functions, AT can calculate the particles trajectories by *ringpass* function. The *passmethod* functions have million calls in particle tracking process and the calculation of multi-particle with multi-turn through accelerator elements is called as Single Instruction Multiple Data (SIMD) operation, which has a good agreement with OpenMP and MPI processing [6]. Parallel computing can work simultaneously for the different part of the same programme by the use of multiple computing resources, which increase the computing speed efficiently [7]. However, it is expected that the parallel computing is compatible with OpenMP and MPI.

OpenMP is an Application Programming Interface (API) for multi-thread programming with C/C++ and FORTRAN, which offers a highly abstract description for parallel computing. OpenMP includes compiler directives, library routines and environment variables which affect the run-time behavior [8]. Using OpenMP routines and directives for the existing AT source code, AT adopts the Uniform Memory Access (UMA) model which all the cores of processors share the same physical memory uniformly. With the moderate changes to the *passmethod* functions written in C, OpenMP can be carried out by Matlab MEX compiling function [9].

MPI is a standard of distribution model adopted in the control of parallel computing by explicit ways. Matlab MPI is a Matlab implementation of the MPI standard, which allows any Matlab program to exploit the multiple processors [10]. It is called NUMA architecture model built with OpenMP and MPI that multi machines run independently with local memory and communicate each other with Bus Interconnect. Figure 1 shows the NUMA model schematic diagram.

ISBN 978-3-95450-136-6

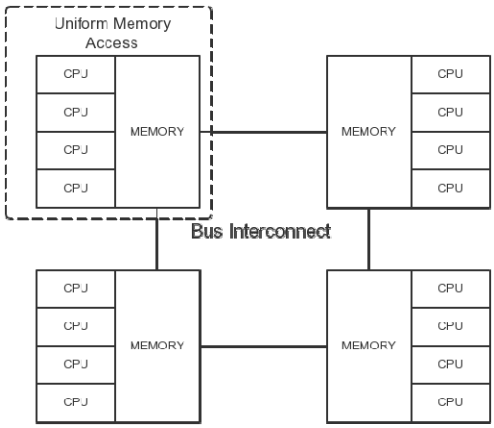

Figure 1: The NUMA schematic diagram.

Here we use a general distributed memory model. The upper computers use Matlab MPI for controlling the parallel computing and communication. And the lower computers use OpenMP for compute. The data is distributed to the lower computers with message-passing functions. And then the computing results from the lower computers can be transferred to the upper computers. Finally, they are combined into the final result. The local task distribution for every CPU can be managed by the shared memory computing interface, so that a global distributed memory model is adopted in AT. Through the way, the data conflict caused by two processors access to the same memory can be avoided. Only with UMA model, the CPU spin time will be raised up. And sometimes the data conflict delay takes two-processor working time longer than that one-processor.

Parallel AT with OpenMP and MPI

OpenMP and MPI are used to the parallel computing of the *passmethod* functions, which can increase the increase the speed mentioned above. To find sections of the codes processed simultaneously, OpenMP and MPI program can be created with existing code. The changings and the compilation of the codes are in the appendix. Library functions *omp_get_num_threads()* and *omp_get_thread_num()* can be used for the parallel part of the function to obtain the number of threads and the id of the working thread numbered from zero. The start_index is the offset of each computing core. Based on the thread id number, different thread works on different data., which realize the paralleling of data and tasks.

For instance, the DriftPass.c [2] can be used to calculate the particle tracking through a drift element. Intel VTune Amplifier [11] is adopted to test the efficiency of the parallel DriftPass.c. The test number of the particles is 3920, with 500000 loops on a qual-core computer. The result is shown in Fig. 2 and Table 1.

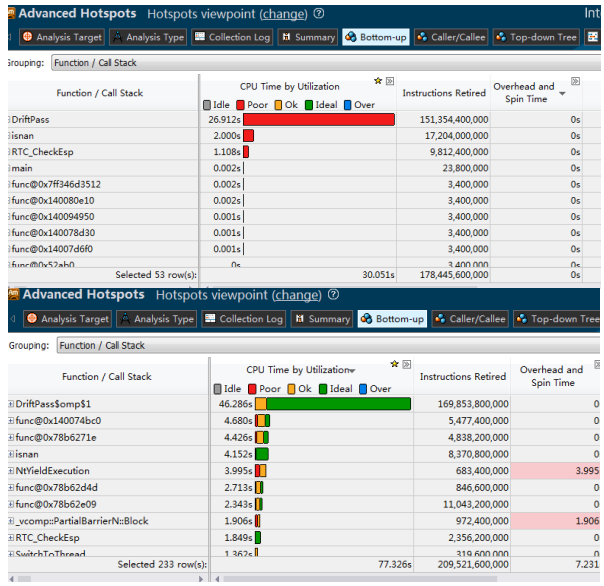

Figure 2: Compute times for parallel and non-parallel DriftPass.

As seen in Fig. 2, the CPU have a large free time during the non-parallel compute because only one core is used for compute, others are free. The parallel compute has some extra time for multi-thread open and so on.

Table 1: Compute Times for Parallel and Non-parallel DriftPass

	Elapsed time/s	CPU time/s	Overhead & Spin time/s
Parallel	10.28	77.83	7.22
non-parallel	30.09	30.05	0.00

CPU time is the sum time for all threads; Overhead & Spin Time is the time which an active thread takes to get a synchronization construct. They take over the most CPU time. Overhead & Spin Time takes about 9.3% of all CPU time. So the speedup of the parallel program is 2.98. And it is not reach the limit value 4. All the *passmethod* functions called in *ringpass* can be parallelized by OpenMP, so that *ringpass* is a parallel program.

RESULTS AND DISCUSSION

Frequency map analysis (FMA) is an analysis method for a dynamic aperture to find the amplitude of frequency shifts. The FMA works as follows: (1) the program flow carry out particle tracking, (2) then through N turns and gets the particles output data, (3) finally uses a first-order Hamming filter and filter the data [12]. It is a frequency scanning tool which is used to reveal information about nonlinear resonances and guide frequency optimization [13]. It is takes lots of time that particle tracking computing. As a result, OpenMP is used to save time and improve the efficiency. Figure 3 shows the result that the time costs when using parallel and non-parallel methods

ISBN 978-3-95450-136-6

to compute FMA with different number of particles. An Intel i7-3770 CPU was used with 4G RAM.

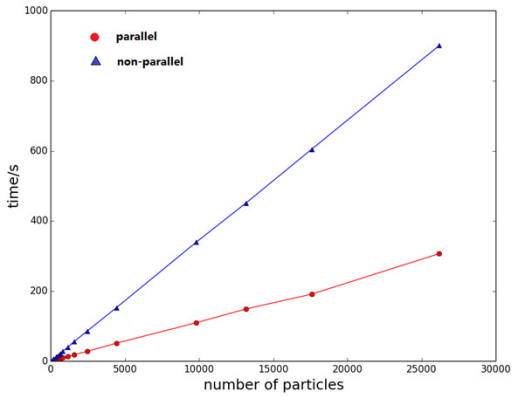

Figure 3: FMA execution time for non-parallel and parallel computing using different numbers of particles.

The number of particles is shown on the x axis and the execution time on the y axis. It is indicated that the time grows almost linearly with the number of particles. The time of non-parallel method takes up to 3.16 times as long as that of the parallel method, so that the speed using parallel computing is at most 3.16 times the speed using non-parallel computing. Based on Amdahl's law [14]:

$$S = \frac{1}{f_{par}/P + (1 - f_{par})}, \quad (1)$$

where f_{par} is the parallel fraction of the program. P is the speedup of the parallel part and S is the speedup of all programs. The formula is used to calculate the times for the parallel and non-parallel parts of the program flow. The parallel part of the program is the *ringpass* function and the main non-parallel part is the FMA function. The results for the parallel and non-parallel programs is shown in Table 2 and Table 3, where N is the number of particles, T_f is the time taken by the FMA function and T_{all} is the time of all program.

Table 2: Time Profile Using Parallel Computing

N	T_f /s	T_{all}/s	T_f/T_{all}/s
512	1.19	5.87	0.202
1128	2.66	12.58	0.207
2450	5.75	28.04	0.205
4418	10.18	51.28	0.206
9800	22.47	109.74	0.207
177578	50.57	191.24	0.212

Table 3: Time Profile Using Non-parallel Computing

N	T_f /s	T_{all}/s	T_f/T_{all}/s
512	1.25	17.52	0.071
1128	2.82	39.11	0.072
2450	6.16	85.92	0.071
4418	11.01	152.52	0.072
9800	24.40	339.43	0.072
177578	43.50	604.51	0.072

As listed in Tables 2 and Table 3, the value Tf/Tall is almost constant for both types of computing. From Eq. (1), the speedup of the parallel part can be calculated as follow:

$$3.16 = \frac{1}{(1 - 0.07)/P + 0.07}. \quad (2)$$

From Eq. (2), 0.07 is the non-parallel part of the computing process and the value of P is about 3.77. Along with the number of particles increasing, P will be close to 4, but not never equal to 4 because of the quad-core CPU with 4 threads. And the synchronization between threads of different cores reduces the compute speed.

A Dell R720 server platform has been developed to use the speedup raised by OpenMP and MPI. R720 has 2 processors with 16 CPU cores each, which can be regard as two compute nodes during the computing process. The speedup of one node is 6.23. And the speedup of 2 nodes is 12.18, which is almost double of one node. It is obvious that because the computing process is independent each other, the communication costs about 2.3% of the total time. The speedup of the computing process can grow linearly with the number of CPU using OpenMP and MPI, which shows good extensibility.

CONCLUSION

This paper describes parallel and non-parallel results with use of OpenMP and MPI, which the parallel optimization of AT improves the computational efficiency. OpenMP and MPI can be carried out to the similar way for other accelerator physics programs if the program parallelized. A multi-core computing platform has been developed and contributed to the deep optimization of accelerators, which is convenient to speed up by adding computing nodes.

APPENDIX

```
#include<omp.h>
…… some computation and initialization
Omp_set_num_threads(4)
#pragma omp parallel private(i) share(start_index,n)
{
thread_id=omp_get_thread_num();
num_threads=omp_get_num_threads();
start= start_index + n*thread_id /num_ threads;
if(thread_id==num_threads-1)
end=n-1;
else
end=n*(thread_num+1)/ num_threads-1;
for(i=start;i<=end;i++){
…computation
}
}
```


ISBN 978-3-95450-136-6

REFERENCES

[1] Terebilo. A, “SLAC-PUB-8732”, 2001.
[2] S.Q. Tian et al., Nuc. Sci. Tech. 25 (2014): 10102-010102
[3] B.C. Jiang et al., HIGH ENERGY PHYSICS AND NUCLEAR PHYSICS. 31(2007):956-961.
[4] X.Y. Yan et al., Journal of South China University of Technology (Natural Science Edition), 40(2012):71-78. DOI:10.3969/j.issn.1000-565X.2012.04.011.
[5] H. Grote et al., “The MAD Program (Methodical Accelerator Design) Version 8.13/8 User’s Reference Manual”, Geneva, Switzerland. January, 1994, 18.
[6] M.D. Salt et al., “Beam Dynamics using Graphical Processing Units”, 2008.
[7] G. L. Chen et al., Chinese Science Bulletin. 54(2009): 1845-1853.
[8] L. Dagum et al., Computational Science & Engineering, IEEE. 5(1998): 46-55.
[9] Y. Zhang, Optimization Methods and Software. 10(1998): 1-31.
[10] J. Kepner et al., arXiv preprint astro-ph/0107406, 2001.
[11] Marowka.Ami, “On Performance Analysis of a Multithreaded Application Parallelized by DiKerent Programming Models Using Intel Vtune Parallel Computing Technologies”, 2011:317-331.
[12] J. Laskar, “Frequency map analysis and particle accelerators”, Particle Accelerator Conference, Portland. 2003: 12-16.
[13] S.Q. TIAN et al., Chinese Physics C. 533(2009): 224
[14] Chandra, “Parallel programming in OpenMP”, Morgan Kaufmann, 3(2001):16-17.

ISBN 978-3-95450-136-6