Implementation of Parallel Convolution Based on MPI

Jin Lu, Kaisheng Zhang, Ming Chen, Ke Ma
Xi'an Electronic Engineering Research Institute
Xi'an, China, 710100
xdmake206@qq.com

***Abstract*—Convolution computing plays an important role in scientific computing. However, traditional Message Passing Interface (MPI) model has the disadvantages such as massive message passing and load-unbalancing. According to these problems, this paper propose a new parallel convolution based on MPI model, which is able to effectively balance the load as well as bring great reduction on message passing. By analyzing the speedup ratio and efficiency, conclusions can be drawn that the new MPI parallel programming, which obtains obvious promotions on parallel efficiency and large-scale sequence's speed, can better exert the advantages of parallel computing and distributional memories between nodes. And it is shown by the experimental results that the proposed method is a feasible and effective parallel strategy.**

***Keywords*—convolution computing; parallel; MPI; load-balancing**

I. Introduction

Confined by insufficiency of process techniques, single core processors have reached their limitations of performance. It has been almost impossible to achieve enhancement on performance of computers by simply increasing the clock frequency of CPU [1, 2]. Therefore, multi-core technique has become a universal solution for manufacturers to obtain notable improvements on CPU performance. Multi-core devices provide a hardware platform of parallel computing for application programs, thus bringing a tremendous promotion on computation speed of computers. In general, there are two basic model of parallel programming. One is based on the memory share where communications between processes are accomplished through CPUs sharing the memories. Typical applications are OpenMp and Pthreads. The other basic model is based on message passing, for example the Message Passing Interface (MPI). In this model, processors use the bus to transmit messages to execute the communication. In most cases, MPI parallel programs based on high quality message passing will gain better performance than OpenMp based on memory share [3].

Meanwhile, during the recent years, engineering fields such as communication, aerospace, biomedicine, radar signal process and so on have proposed much higher requirements on massive data and real-time performance. And convolution has been one of the most important methods of time-domain analysis of signal and system. In this situation, using parallel ideas to achieve enhancement in speed and efficiency of convolution computation is of great engineering significance.

II. MPI Parallel

MPI is one of the standards of message passing parallel program. And it has been a main parallel programming environment. MPI applies to the message passing models of parallel computer systems based on distributed memories. It can be transparently employed in heterogeneous systems and has advantages in portability, function and efficiency [2]. Besides, since there are a great number of available implements, most of which are free, effective and practical, MPI has the superiority that it is supported by almost every computer manufacturer.

MPI has two main communication modes: point-to-point [4] communication mode and collective communication mode. Point-to-point mode refers to communication between processes, including blocking communication and non-blocking communication. In blocking communication, both the standard sending and receiving functions of MPI are blocked, which means that receiver will be blocked after calling for the MPI_Recv until the matching message arrives. In comparison, non-blocking communication is designed for the implementation of overlap of computing and communicating, which will enhance the execution efficiency of the program. In non-blocking communication, programs are able to return without having to wait for the completion of the communicating operation. Instead, the operation will be accomplished by specific hardware. So the computing operations in processor and the communicating operations in hardware can then be performed simultaneously, thus realizes the overlap of communication and computation.

Collective communication undertakes the tasks of communication, synchronization and computation, among which the communication is mainly about the transmitting of group data, synchronization keeps all the processes at the same pace when in specific part of the system, and computation accomplishes operations required on given data.

One of the most essential differences lying between the two communication methods of MPI is that collective communication requires the participation of all the processes in a specific group. And these processes have the same calling methods. While in comparison, point-to-point method only involves the processes who execute the sending and receiving actions. These processes are distinguished as senders and receivers, and have different calling methods.

III. Parallel Model of Convolution Computation

Suppose that x(n) is a sequence of length N1, h(n) is another sequence of length N2, y(n) is the convolution of x(n) and h(n). Then y(n) can be achieved as following:

$$y(n) = x(n) * h(n) = \sum_{m=-\infty}^{+\infty} x(m)y(n-m)$$
$$= \sum_{m=low}^{top} x(m)y(n-m) \begin{cases} top = \min(n, N_1 - 1) \\ low = \max(0, n+1-N_2) \end{cases} \tag{1}$$

As shown in Figure 1, for better understanding of the convolution equation, firstly rotate x(n) 180 degrees on x(0) when h(n) stays the same. Then y(0) equals to the product of h(0) and x(0). Secondly, move x(n) right one bit, and calculate the sum of the products of the overlapped bits, thus y(1) can be got. In this way, the whole sequence of y(n) can be achieved.

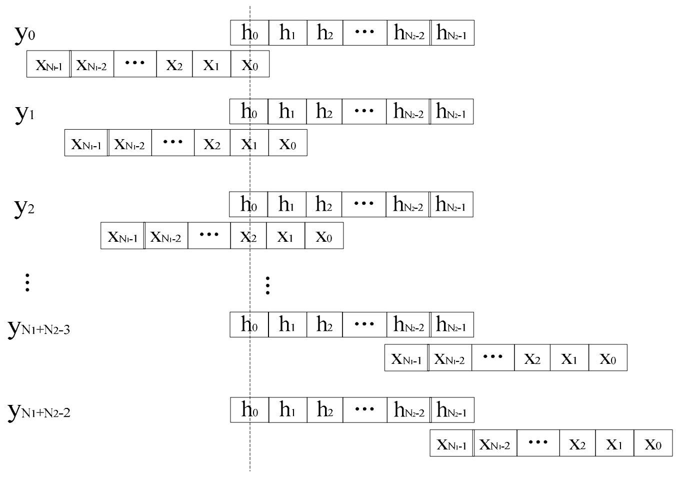


Fig. 1. Convolution computation process.

A. Traditional MPI Model Based on y(n) Allocation

Traditional parallel convolution method based on MPI allocates tasks according to y(n). Its basic procedure is as followed:

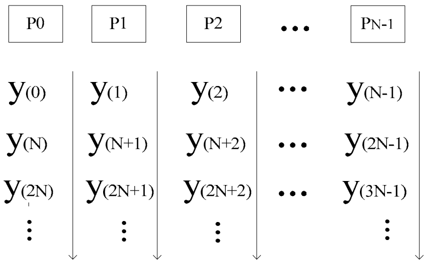
Step 1: Broadcast x(n) and h(n) respectively to processes of the number of N. The main process applies for N1+N2-1 storage spaces to store the result of y(n);

Step 2: Apply for storage spaces of the scale of ceil ((N1+N2-1)/N) named temp for each process, except for the main process (ceil means rounding up);

Step 3: Allocate tasks to each process using loop distribution as shown in figure 2. The k-th processor calculates sequences y(k), y(k+N), y(k+2N)， y(k+3N) ,··· Then uses formula (1)to obtain the convolution results and store them in temps;

Step 4: Synchronize all the processes through MPI Barrie, where each process will have to wait until all the processes have reached this step;

Step 5: Repeat above operations for (N1+N2-1)/N times, and uses MPI_Gather [6] to move data from temp to main process y in sequence.



P0
P1
P2
PN-1
y(0) y(1) y(2) ··· y(N-1)
y(N) y(N+1) y(N+2) ··· y(2N-1)
y(2N) y(2N+1) y(2N+2) ··· y(3N-1)

Fig. 2. Task allocation of each process.

Though traditional convolution computing method has speed advantages over general serial computing in most cases, it still has following shortages:

1. Load unbalancing, which will result in each process having different amount of calculation. y(0), for example, has only one multiplication to complete when y(1) has to finish two multiplications and one addition and y(3) needs three multiplications and two additions. And this will lead to the increasing of the idle time of processes.

2. Massive communications [7] exist between processes, which may severely decrease the efficiency of executing, since there are (N1+N2-1)/N gathering operations except for the initial broadcast. These gathering operations will occupy huge computation resources.

B. MPI Model Based on Matrix Partition

Aiming at the above shortages of the traditional convolution method, this paper proposes a new MPI model based on matrix partition. For convenience, a matrix is established using x(n)s of number N1 and h(n)s of number N2, so that the matrix is N1*N2. As shown in figure 3, it can be easily observed that the convolution result y(k) equals to the sum of the elements on the k-th diagonal. As a result, the convolution computation is then transformed into calculations of the elements on the k-th diagonal and their sum.

These elements can be calculated using parallel computing method, which is represented in figure 4. The matrix is separated into N sub-matrixes by column, where N is the number of processes (round up when N2 cannot be parted equally. For example, if there are N2=100 columns and N=8 processes, then each of the first seven sub-matrixes will have 13 columns and the 8thcolumn will have the last 9 columns). After that, each sub-matrix will be allocated to the N-th process, which will calculate its sum of the elements on the diagonal, as shown in figure 4. At last, the reduction of the results of the sub-matrixes will be calculated by collective communication. Thus, the sum of the elements on the diagonal of the matrix which is also the result of the convolution can be achieved.

It can be observed that throughout the whole computing procedure, each process is independent with each other. And there is only one collective communication operation in the final reduction. So the communication work is largely reduced. In spite of this, when N2 can be divided exactly by N, the load of each task will be the same, which will lead to a great save of computing resource consumed by synchronization of the processes.

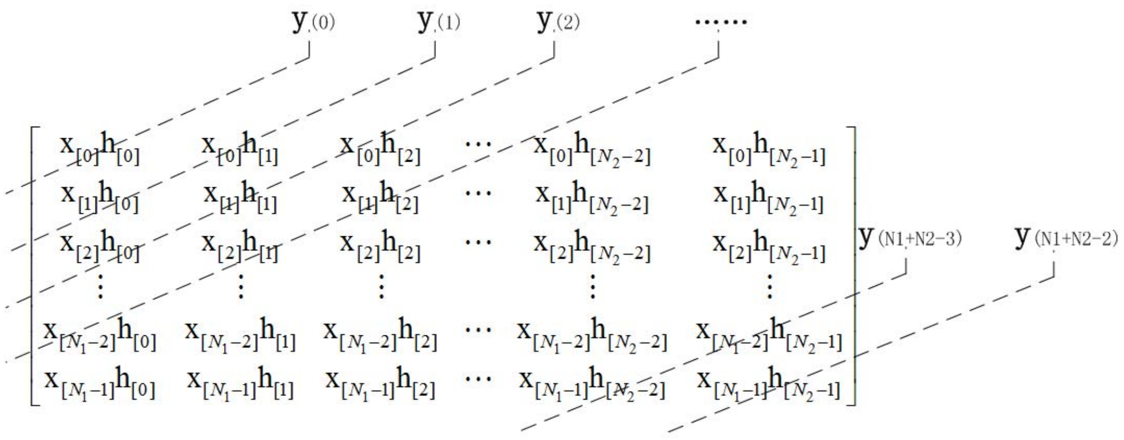


Fig. 3. Matrix partition.

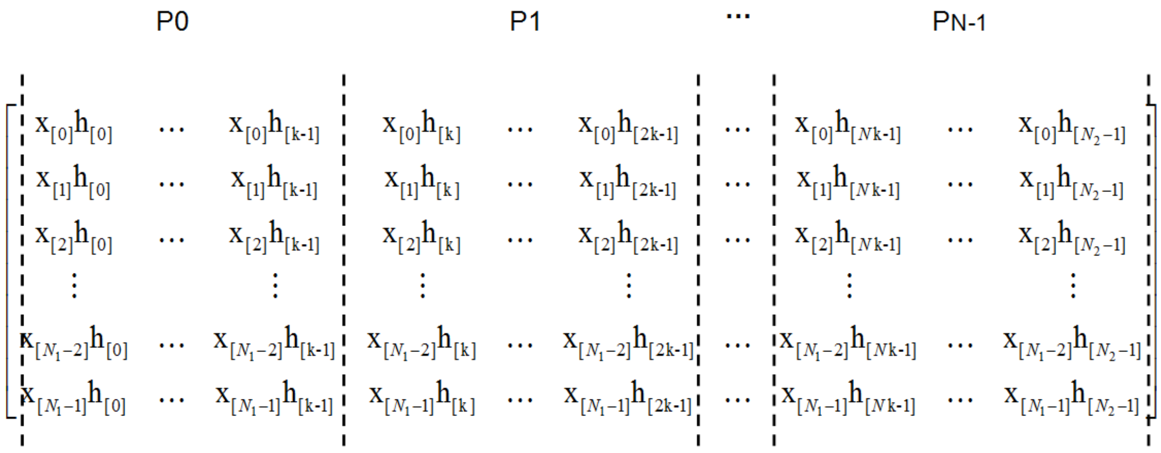


Fig. 4. Matrix partition.

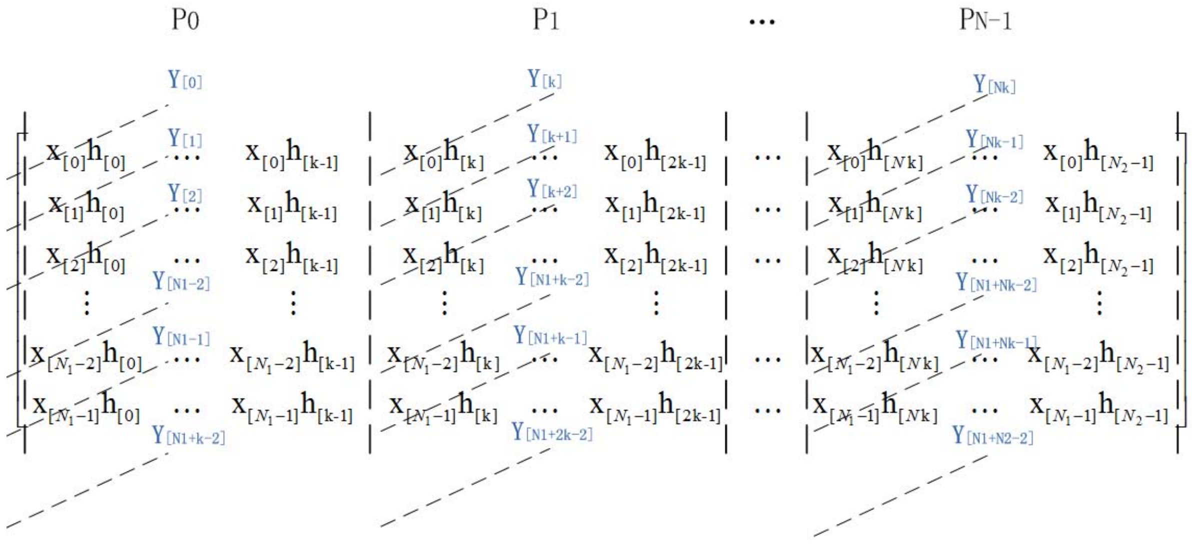


Fig. 5. Matrix partition.

IV. Experiment Analysis

To test the performance of the matrix partition based MPI model proposed in this paper, numerical experiments are conducted. And for comparison, general serial model and traditional MPI parallel model based on y(n) distribution are also tested. The experimental environment is CPU: Intel i7@3.4GHz; Memory: 4.00GB; Operation System: Linux Fedora 17; MPI library: MPICH2. The number of CPUs maintains 8. And various lengths of sequences are chosen to test the E-times of these three models. The results are shown in figure 6, where the horizontal axes refer to the lengths of the sequences, and vertical axes refer to E-times. And for observation, vertical axis in figure 6(b) uses logarithmic coordinates.

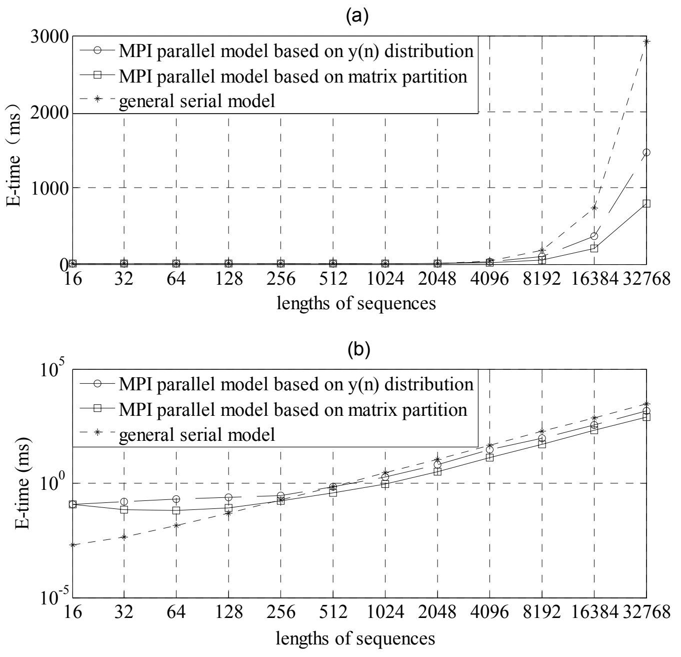


Fig. 6. Computational efficiency comparison of convolutions of equal lengths.

From the results, it can figure out that the method which adopts parallel convolution achieves obvious enhancement on speed when the length of sequence is over 256. And when it reached 1024, the matrix partition based MPI convolution algorithm will be 1.6 times as fast as the traditional method, and the speedup ratio may approximately be 3. So the feasibility and efficiency of the model are proved.

V. Conclusions

The experimental results of convolution computing of differed lengths proved that the matrix partition based MPI algorithm proposed in this paper has better executive efficiency. Since the advantages of parallel computing based on multi-core and distributed storage between nodes are better exerted, the E-time of the computing has been greatly decreased. And the performance of parallel computing of this model will be further improved in higher multi-core environment.

However, when the length of the sequence reaches quite long, the time-domain effect will not be so obvious. So further research on frequency-domain parallel algorithms and CUDA based heterogeneous algorithms are to be conducted.

References

[1] Michael J. Quin, “Parallel Programming in C with MPI and OpenMP,” McGraw-Hill Company, Inc. , 2005.

[2] Calvin Lin, Lawrence Snyder, “Principle of Parallel Programming,” Pearson Education, Inc. , 2009.

[3] ZHANG Longbing, WU Shaogang, CAI Fei, “Shared-Memory Versus Message-passing on PC Cluster,” Journal of Software 2010.

[4] Peter S. Pacheco, “An Introduction to Parallel Pragramming,” Singapore: Elserier Pte Ltd, 2011.

[5] Fayez Gebali. Mahafza, “Algorithms and Parallel Computing,” Wiley Publishing,Inc., 2012.

[6] Snir M, Otto S, HussLederman S, Walker D, Songarra J, “MPI: The Complete Reference,” London: MIT Press, 1996.

[7] Lu HH, Dwarkadas S, Zwaenepoel W, “Message passing versus distributed shared memory on networks of workstations,” Redelfs A ed. Proc. of the 1995 ACE/IEEE Supercomputing Conf. 1995.

[8] Aslot V, Domeika M, Eigenmann R, et al, “SPEComp: A new benchmark suite for measuring parallel computer performance,” Sloot PMA, Bubak M, 1999.

[9] Grama, Ananth, Anshul Gupta, George Karypis, and Vipin Kummar, “Introduction to Parallel Computing,” Addison-Wesley, 2003.

[10] Ungerer B, Rubic B, and Slic J, “Multithreaded processors,” Computer Journal, 45(3): 320-348.2002.