Multi-core Parallel Programming Method Based on Communication in Win32 Platform

Qing. Li, and Luna. Xu

Abstract—With the development of computer science and technology, more and more often does the term “multi-core parallel computing” appear in the field of software. Recently there are several parallel programming models with multi-threads. The thread-based parallel programming model includes row-thread library directive based models and tasking models. This paper proposes a communication model --- Multi-thread Interface (MTI) for Win32 which is similar to the MPI library, and implements the MTI parallel programming model based on that. Experiments are provided to show the utility of using MTI to do parallel programming. The results prove MTI to be an easy to use.

Keywords—multi-core, multi-thread, parallel

I. INTRODUCTION

A. Arrival of Multi-core era

In the effect and limitation of the Moore's law and the power consumption of chips, it's hard to promote the computation speed of CPU just by promote the clock frequency of CPU. On the other hand, the asynchrony of CPU speed increase and the memory frequency increase becomes a bottleneck problem to improve computer performance. These problems bring about the research of the multi-core systems. Parallel process becomes the major solution to improve computer performance[1]. Multi-core/Many-core CPUs have been published and gradually spread. Such as MPC8641D by Freescale; Pentium D8XX by Intel, etc.. It is because it's easier to improve processor's performance by increasing the number of cores rather than by increasing the frequency. Meanwhile, support for multi-core is offered both in hardware level and OS level. Multi-core/Many-core technology becomes the main trend to improve CPU performance[2].

B. Effect on HPC by multi-core technology

Multi-core technology brings fundamental changes to software development, especially on those general applications and applications running on PCs and low-level servers. Developers need to design threads model to utilize multiple cores; dispatch codes which need higher performance on different cores, and meanwhile make sure the design has good scalability to run on different kinds of computers of single-core; multi-core/many-core, or even higher level of computers.

In multi-core systems, rather than those parallel models based on communications of progresses, which represented by MPI, parallel models based on threads can avoid the cost of communications of progresses. Theoretically efficiency will be higher if running on Shared Memory Processor (SMP) structure system.

II. DESIGN OF MULTI-CORE PARALLEL MODEL BASED ON THREAD COMMUNICATION

According to the research of papers, there are many kinds of thread based parallel models. Mostly in three types: raw thread models like pthread; instructive models like OpenMP and task models like TBB[3]. These three models have their own characters on usability, performance and thread synchronization. Raw thread model provides the highest performance, and it's the most flexible model since users schedule the resources. But it's not easy to use and users carry big responsibility. The flexibility of instructive models is low, the mechanism based on lock will cause synchronization problems. Task models have low adaptability, hunger problem will occur and influence the efficiency.

Throughout a variety of typical parallel program development tools, those based on message passing are widely used, such as PVM(Parallel Virtual Machine), MPI(Massage Passing Interface). MPI is the most important tool. It has good scalability and very powerful. Almost all parallel computer manufacturers support it. MPI has become the de facto standard of parallel models. Although it's just a temporary solution, over the decades since it was published, software and researches based on MPI have a big growth[4].

The paper is based on the research of MPI model, we use the advantage of MPI to develop a thread communication model MTI.

A. Structure of MTI

We modified the framework of MPI, focus on characters of multi-thread and designed the structure of MTI.

Qing. Li is with Shanghai University, Shanghai, China. Professor and subdecanal of School of Computer Engineering and Science. Senior Member of China Computer Federation (CCF). Mainly study on High Performance Computing.

Luna. Xu is with Shanghai University, Shanghai, China. Master candidate studying on High Performance Computing.
As showed by figure 1 and 2, the main difference between MPI and MTI is in the structure of MTI, thread function should be defined. The computing part will be separated from the main function and performed by thread function. The basic role of the main function is to initialize, coordinate, and summarize the result.

As is known to all, there are 6 major functions in MPI, by analyzing the 6 functions, we designed 6 main functions in MTI which accord with the MPI standards, and considering the specialty of the SMP thread model and the higher demand for synchronization, we offered a function to synchronize threads additionally. The functions are showed in table 2-1.

### TABLE 2-1

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>int MTI_Init(int argc, char ***argv, MTI_ThreadFunc threadFunc, void * args)</td>
<td>Initialize the environment</td>
<td>Argv: Number of threads, number of cores by default threadFunc: point to the thread function</td>
</tr>
<tr>
<td>int MTI_Comm_size(MTI_Comm comm, int *size )</td>
<td>Get the size of the thread group</td>
<td>Size: number of threads in the group Commm: communication type: main thread or other threads.</td>
</tr>
<tr>
<td>int MTI_Comm_rank(MTI_Comm comm, int *rank )</td>
<td>Get the current thread id</td>
<td>Rank: the id of the thread</td>
</tr>
<tr>
<td>int MTI_Send(void* buf, int count, MTI_Datatype datatype, int dest, int tag, MTI_Comm comm)</td>
<td>Send message to another thread</td>
<td>Buf: data to be sent Count: size of data Datatype: type of data Dest: destination thread id Tag: signal Commm: communication type</td>
</tr>
<tr>
<td>int MTI_Recv(void* buf, int count, MTI_Datatype datatype, int source, int tag, MTI_Comm comm, MTI_Status *status)</td>
<td>Receive messages from another thread</td>
<td>Buf: data to receive Count: size of data Datatype: type of data source: source thread id Tag: signal Commm: communication type Status: receive status</td>
</tr>
<tr>
<td>int MTI_Finalize(void);</td>
<td>Receive messages from another thread</td>
<td>none</td>
</tr>
<tr>
<td>int MTI_Barrier(MTI_Comm comm);</td>
<td>Synchronize threads, all child threads wait in a barrier</td>
<td>Commm: communication type</td>
</tr>
</tbody>
</table>

### B. Program flow of using MTI

One can develop a parallel program using the major functions of MTI, the work flow is as figure 3.

---

**Sponsored by Shanghai science and technology commission key projects foundation (10510500600); Shanghai key discipline construction project funding (J50103)**
B. Synchronization

MTI provides MTI_Barrier function as an explicit synchronizing. The function is called by a thread to make all threads to run to the same barrier.

We know that there are many synchronizing objects to use in thread synchronization such as critical section, mutex, event and semaphore[5]. Those objects are used in MTI in different ways to implicitly provides synchronization.

For example, MTI_Send and MTI_Recv uses critical section to control the envelop queue; events to coordinate the send thread an receive thread in case data mess. MTI_Finalize uses WaitForMultipleObjects() method to wait all threads to return to free memory.

IV. EXPERIMENT

This paper tests MTI using 2 typical numerical calculation problems: numerical integration--π calculation; Jacobi iteration. The environment is intel core2 duo, 2GB memory, windows 7. The MPI version is mpich.nt.1.2.0.4.

A. Numerical Integration--π Calculation

π Calculation is a classical testing program in MPI. According to the mid-rectangle formula, we have:

\[
\pi = \int_{0}^{1} \frac{4}{1 + x^2} \, dx \approx \frac{1}{n} \sum_{i=0}^{n-1} f\left(\frac{i + 0.5}{n}\right)
\]

\[
f(x) = \frac{4}{1 + x^2}
\]

(4.1)

According to the formula above, have n=nIntervals, dispatch the n times computing to nThreads number of threads. Every thread execute n/nThreads times of accumulation. Put the result to a sum array orderly. Finally the main thread summarize the results, add them all and have the final answer.

There is a serial processing version and MPI version of the program to make comparison. The results are showed in table 4-1.

<table>
<thead>
<tr>
<th>Interval</th>
<th>Time of serial computing ts(s)</th>
<th>Time using MTI tp(s)</th>
<th>Seedup Efficiency(%)</th>
<th>Time using MPI t(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000,00 0</td>
<td>0.070</td>
<td>0.375</td>
<td>1.856</td>
<td>92.800</td>
</tr>
<tr>
<td>10,000,000</td>
<td>0.601</td>
<td>0.289</td>
<td>2.079</td>
<td>103.980</td>
</tr>
<tr>
<td>100,000,000</td>
<td>6.700</td>
<td>2.600</td>
<td>2.577</td>
<td>128.846</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>69.352</td>
<td>25.978</td>
<td>2.670</td>
<td>133.482</td>
</tr>
</tbody>
</table>

\[
Seedup = \frac{tp}{ts} \times 100\%
\]

(4.2)

\[
Efficiency = \frac{tp}{tn} \times 100\%
\]

(4.3)

A total of 4 groups data is tested. The calculated amount is with ten times increasing. It can be seen that the time using traditional serial method is about 2 times of the time using MTI. The speedup is above 2, and efficiency reaches 100%. As the interval increases, calculation time increases, the promotion of the performance becomes more obvious. Some efficiency exceed 100% due to the specialty of the problem, the threads doesn't need to communicate. Using MTI in this kind of problems can double the performance. On the other hand, time of using MPI is of cause less than the time using serial method, yet more than the time using MTI, it's because MPI uses 2 progresses to calculate, which involves communication, and costs more.

From CPU charts we can see that under serial processing, the utilization of CPU is low, only around 50%; yet under parallel processing, the CPU is fully used.
It can be seen from figure 6 that one of the two cores is doing the calculation and the other is doing other stuffs. But in figure 7 and 8, both of the cores are doing the calculation.

B. Jacobi Iteration

Jacobi iteration is a common iteration method. In a nutshell, the new value is the average of the neighbors of the old value. Jacobi iteration can be easily divided into parallel parts, and it's a common example of parallel computing [6].

This experiment will divide a 16*16 matrix by columns (figure 9). Two threads do the computing parallel. Since the new value is based on the neighbors, additional columns are needed on both sides to store the data. During computation, each thread will need to provide and get data to their neighbor thread (figure 10). The results are as table 4-2.

<table>
<thead>
<tr>
<th>Iteration steps</th>
<th>Time of using MPI (t₁) (s)</th>
<th>Time of using MTI (t₂) (s)</th>
<th>Seedup</th>
<th>Increase(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>0.350</td>
<td>0.242</td>
<td>1.446</td>
<td>30.857</td>
</tr>
<tr>
<td>100,000</td>
<td>2.802</td>
<td>1.900</td>
<td>1.475</td>
<td>32.191</td>
</tr>
<tr>
<td>100,000,0</td>
<td>28.188</td>
<td>18.475</td>
<td>1.526</td>
<td>34.458</td>
</tr>
</tbody>
</table>

\[ I = \frac{t₁}{t₂} + 1 \tag{4.4} \]

A total of 3 data groups are tested. Communication is performed in each iteration steps, so increase of iteration steps will also increase the number of communication. We can see from the results that using MTI can avoid the higher cost brought by progress communication and improve the program performance. As the steps increase, the improvement is more obvious.

V. Conclusion

This paper presents a communication parallel model similar to MPI on win32 platform, and implements MTI. Furthermore, two experiments are conducted in different ways to compare MTI, MPI and serial method in efficiency, speedup. Results prove that using MTI is more efficient than serial processing and MPI in both communication-needed and communication-free scenario.

References