Evaluation of WIEN2K Performance on MPICH2 vs. MPICH1

Hadi Khalilieh and Nidal Kafri

Department of Computer Science, Al-Quds University, Jerusalem, Palestine

hkhalilia1@science.alquds.edu, nkafri@science.alquds.edu

Rezek Mohammad

Palestinian Technical University/Khadoorie, Middle East Technical University/physics department

esteteh@hotmail.com

ABSTRACT

The emerging multi-core computer architecture attracts the researchers to utilize this architecture as an adequate and inexpensive solution to achieve high performance computation for many problems. Where, the multi-core architecture enables us to implement shared memory and/or message passing parallel processing paradigms. Therefore, we need appropriate standard software libraries in order to utilize the resources efficiently for a given computational problem.

In this work we evaluate the performance of two versions of the well known massage passing interface (MPI) library: MPICH1 vs. MPICH2. In our experiments we used two benchmarks. The first one is the WIEN2K application which is based on Density Function Theory, and the second is a Matrix multiplication. The results show that we achieve better performance when MPICH2 is used than MPICH1.

KEYWORDS

Parallel Processing, Message Passing Interface MPI, MPICH1, MPICH2, performance, multi-core systems, WIEN2K.

1. INTRODUCTION

In order to achieve high performance computing i.e., reducing computing elapsed time, parallel processing is widely used in scientific computing, engineering, multimedia application, industry, computer systems, statistical applications, and simulation. One of the important applications that need to speed up computation is WIEN2K application which is base on Density Functional theory.

Usually parallel processing can be implemented on shared memory computer systems or distributed memory systems using message passing paradigms. A hybrid approach using both paradigms also can be implemented. Parallel processing was usually carried out on expensive supercomputers and mainframes. After that, the emerging high performance computer network and protocols attracted the researcher to use the distributed memory parallel processing on clusters of on shelf computers and Grid computing.

In the past decade, the development of multi-core systems shifted the interest of many researchers towered parallel computing on such multi-core systems. Thus, we can achieve relatively cheap high performance using message passing, share memory, or hybrid techniques on single or a cluster of multi-core computers[2][3]. In order to facilitate realization of parallel programming on different platforms, there are several supporting libraries. For example, we can use PVM, JPVM and MPI for message passing on distributed memory. Also Posix and OpenMP are used for multithreading on shared memory [3]. It should be noted that these libraries

provide us with well defined standard interface to achieve portability and flexibility of usage. However, the developers of these libraries intend to improve the implementation to cope with the emerging platforms to increase the utilization efficiency. In this work we focus on evaluating the performance of different versions of MPI library namely MPICH1 and MPICH2. Since WIEN2K is currently using MPICH1.

The WIEN2K can simulate physical and chemical systems supposed to form a new material, this is very necessary to the laboratory person, who can produce the desired material such as drug and medicine [8]. The WIEN2K applied a parallel method to solve quantum mechanics equations based Density Functional Theory (DFT) to find the cohesive energy of any material.

In this work we evaluated the performance of MPICH1 and MPICH2 by running WIEN2K that originally uses MPICH1 and the new implementation of WIEN2K on MPICH2 as benchmark. Moreover, we implemented a matrix multiplication on both MPICH1 and MPICH2.

This paper is organized as follows: Section 2 reviews the main difference between MPICH1 and MPICH2**.** In section 3, literature review and background are introduced. Next section (4) discusses the experiment and the results. Finally, a conclusion and future work are provided in section 5.

2. PRELIMINARIES

Multi-core systems and clusters become an interesting and affordable platform for running parallel processing to achieve a high performance computing for many applications and experiments. For instance: internet service, database, scientific computing and simulation. This is due to their scalability performance/cost ratio [1].

On the other hand, there are many Libraries to support the shared and distributed memory. The message passing interface (MPI) is a set of API functions that enable programmers to write parallel programs based on message passing paradigm. One of the well known APIs MPICH1 which established based on MPI standard that founded in April 29-30, 1992 work shop in Williamsburg Virginia [4]. This library API supports FORTRAN and C programming languages. It has been issued with several modifications and extensions to support dynamic processes, one-sided communication, parallel I/O, etc [13][14]. MPICH2 standard is intended for use by all those who want to write portable message-passing programs in Fortran 77, FORTRAN 95, C and C++ [5]. The improvement of MPICH2 focused on many issues and functionalities such as dynamic processes, one-sided communication, parallel I/O, etc [13][14]. Of course, a number of changes about how you run them, dynamic spawning tasks and the nature of communication will be different. By new added features in MPICH2, we will get it more robust, efficient, and convenient to use [4]. Consequently, we will focus on the improvements in MPICH2 that we believe they have an impact on the performance:

1. MPICH1 focused mainly on point-to-point communications But MPICH2 included a number of collective communication routines and was thread-safe [4].
2. MPICH2 supports dynamic spawning of tasks. It provides primitives to spawn processes during the execution and to enable them to communicate together [11].
3. MPICH2 supports One-sided Communication. It provides three communication calls: MPI_PUT (remote write), MPI_GET (remote read) and MPI_ACCUMULATE (remote update). These operations are non-blocking [12] [14].

4. MPICH2 used generalized requests that aren't used by MPICH1**.** These requests allow users to create new non-blocking operations with an interface [14].
5. In MPICH2, significant optimizations required for efficiency (e.g., asynchronous I/O, grouping, collective buffering, and disk-directed I/O) are achieved by the parallel I/O system [14].
6. MPICH-1 defined collective communication for intra-communicators and two routines for creating new intercommunicators. But **MPICH-2** introduces extensions of many of the MPICH-1 collective routines to intercommunicators, additional routines for creating intercommunicators, and two new collective routines: a generalized all-to-all and an exclusive scan [14].
7. **MPICH2** supports MPI THREAD MULTIPLE by using a simple communication device, known as "ch3 device" (the third version of the "channel" interface) but MPICH1 doesn't support MPI THREAD MULTIPLE [5].
8. **MPICH1** doesn't concern with communication rather than process management. But **MPICH2** concerns with communication rather than process management. However, MPICH2 provides a separation of process management and communication. The default runtime environment consists of a set of daemons, called mpd's, that establish communication among the machines to be used before application process startup, thus providing a clearer picture of what is wrong when communication cannot be established and providing a fast and scalable startup mechanism when parallel jobs are started. But MPICH1 doesn't separate them and mpd's are built in [15].
9. **MPICH1** required access to command line arguments in all application programs before startup; including FORTRAN ones, so MPICH1's configure devoted some effort to finding the libraries such as libraries that contained the right versions of iargc and getarg. But **MPICH2** does not require access to command line arguments of applications before startup and MPICH2 does nothing special for configuration. If you need them in your applications, you will have to ensure that they are available in the environment you are using [15].

Various operating systems including Linux, Solaris, and Windows can be used for managing computer resources such as memory, I/O and CPU [6].

3. LITERATURE REVIEW AND BACKGROUND

Materials are build from atoms, atoms composed of a heavy positively charged nucleus and lighter particles called electrons. These particles interact with each other and also with their neighbors in the next atoms. In order to study the stability, structural, thermodynamic, mechanical, transport properties and electronic properties of these materials we have to solve many body second order deferential equation called equation of state, this equation obeys the laws of quantum mechanisms.

The equation of state composed of the kinetic energy operators for both the nucleus and electrons, potential energy resulted from interaction between electrons them self, nuclei's them self and nuclei's and electrons; these operators are measured by solving many-body Hamiltonian for the system, which is illustrated in equation (1) [7][10]

This equation can be solved numerically after transforming it to a one body problem after some approximations, this method called Density Functional Theory (DFT) [8][9].

$$H\Psi = E\,\Psi$$

$$\hat{H} = -\frac{h^2}{2}\sum_i \frac{\nabla^2_{\vec{R}}}{M_i} - \frac{h^2}{2}\sum_i \frac{\nabla^2_{\vec{r}}}{m_e} - \frac{1}{4\pi\epsilon_0}\sum_{i,j}\frac{e^2 Z_i}{\left|\vec{R}_i - \vec{r}_j\right|} - \frac{1}{8\pi\epsilon_0}\sum_{i\neq j}\frac{e^2}{\left|\vec{r}_i - \vec{r}_j\right|} + \frac{1}{8\pi\epsilon_0}\sum_{i\neq j}\frac{e^2 Z_i Z_j}{\left|\vec{R}_i - \vec{R}_j\right|} \quad (1)$$

In Our work here the Program packages like WIEN2K[7], using Full potential –Linear Augmented Plane Wave And Local Orbital's (FP-LAPW+Lo) technique is used, in such studies we have two main factors controlling the calculation, these two factors are vice versa, the first factor is the time of calculation and the second is the sample actuality, the sample actuality means here the number of atoms constituting the sample, the bigger the number is the more actual case we have, and more complexity, this will cost a lot of calculation time. WIEN2K package composed of five modules, each module solve one of the equations from (2) to (5) sequentially:

- The first module is called **LAPW0**, in this process the V_{xc} is calculated in the crystal from the initial density P_0 using poisons equation:

$$\nabla^2 V_{xc} = \rho(r) \quad (2)$$

- The second and third module is called **LAPW1**, **LAPW2** which are responsible for building and solving the Schrödinger equations (3) and (4), (setting up H and S matrix), and solves the generalized Eigen value problem for special point in the crystal. The number of these points is proportional to the reality of the study. The high number gives more accurate results and costs a lot of computational time, so Balanced is essential.

$$H_{ks}\Psi = \mathrm{E}\,\Psi \quad (3)$$

$$(-\nabla^2 + V_{xc})\,\Psi = \mathrm{E}\,\Psi \quad (4)$$

∇^2: is the second derivative with respect to space coordinates.
V_{xc}: is the effective attractive potential each electron feel.
E: is the energy of this electron in this crystal phase.
Ψ: is the wave function of this electron.

- The fourth module is called **LCORE**: from the density function, the electrons in the crystal are distributed on the lowest energy values, the density function for the core electrons is also calculated and in LCORE process as in equation (5):

$$\rho(r) = \int \Psi\,\Psi^* dr^3 \quad (5)$$

- The fifth module is called **MIXER**: the new total density is compared with the old density, if the values are the same or the difference is less than an assigned value; the self consistent (SC) is finished as shown in Figure 1. The total energy and wave functions of the electrons are found. Otherwise, the new density is mixed with old density with a percentage decided at the beginning of the calculation to reproduce a new density to run another cycle to get faster convergence and recalculate V_{xc} using equation (2).

The main scalable quantity for measuring the stability of any material is the cohesive energy; cohesive energy equals the difference between the total energy of the material in combined form and the sum of the free atom's energy in their free state as shown in equation (6)

$$\boldsymbol{E}_{\text{cohesive energy}} = \boldsymbol{E}_{\text{compound}} - \sum \boldsymbol{E}_{\text{free atoms}} \quad (6)$$

Each stable form of these atoms can produce positive value for the cohesive energy, the material normally can take more than one stable state, and the state with the highest cohesive energy is the most stable one [10].

The authors in [8] compared two parallel approaches that run on MPICH1 channel. The two methods are: Distributed k-point and Data distribution. However, the first one runs each of the two modules (LAPW1, LAPW2) in parallel way. But the other runs each of the first three

modules in parallel. In addition, a comparison between serial and parallel approaches for running Matrix Multiplication on MPICH1 was in [1].

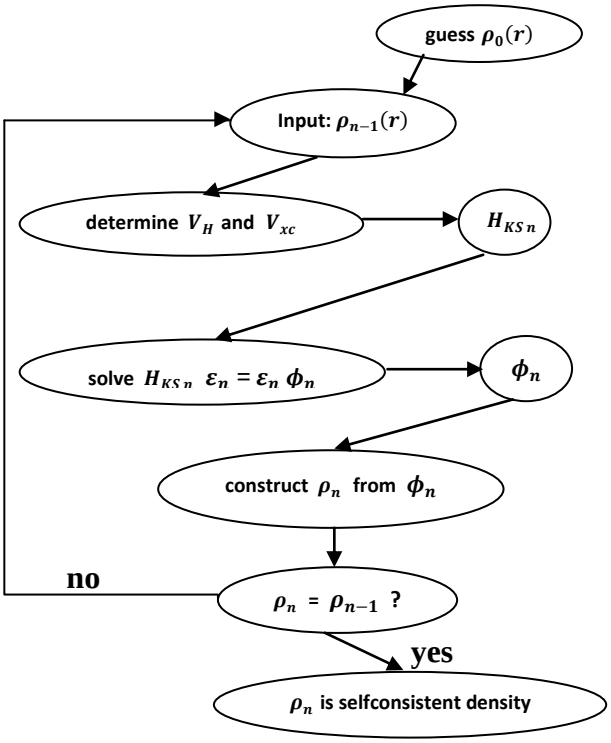

Figure 1: Physical problem solving steps

4. EXPERIMENT AND RESULTS DISCUSSION

In our study, we focused on distributing tasks of WIEN2K program using MPICH1 and MPICH2 on multi-core machine. Whereas, in [8] the experiments were carried out on a cluster using MPICH1 to distribute WIEN2K task. The main contribution in our work depends on the comparison between the results of these experiments.

Our experiments were running on Linux (Fedora 14) installed on multi-core (quad) machine (Intel Core i5 3GHz processor); the specification details of the experiments platform/machine are listed in Table 1.

Table 1: Machine Specifications		
No	**Specification**	**Multi-Core PC**
1	CPU speed	Quad 3 GHz
2	RAM size	8 GB
3	Cache	8 Mbyte
4	HD speed	7200 RPM

To accomplish the calculations, a set of programs were installed on Fedora Linux version 14 and optimized with appropriate options together with WIEN2K. These programs are listed in Table 2.

Table 2: Software Requirements		
Program name	Version	Source
WIEN2K	13.1	www.WIEN2K.at
MPI Channel	MPICH1.3 & MPICH2-1.0.5p3	www.mpich.org
Intel Fortran 90 Compiler	11.072	Intel
Intel C Compiler	10.074	Intel
Mathematical Kernel Library (MKL)	11.0	Intel
Fastest Fourier Transform in the west (FFTW)	FFTW-2.1.5	Intel

Recall that we continue the work of [8], where they installed and used MPICH1 to run WIEN2K program. For this work we installed MPICH2 channel then installed WIEN2K MPICH2 version and run "LAPW0" which is a basic module of WIEN2K. This is done via determined parallel commands. These Commands were written on the terminal of the operating system.

The experiment was carried out by running the programs (LAPW0 and Matrix Multiplication) using MPICH1 and MPICH2 on one, two, three, and four processors of the quad multi-core machine. Where, each processor has a unique id from 0 to 3. Each experiment was repeated several times and the average of the elapsed time were recorded. The experiments in divided into two cases: the first one is running LAPW0 for one cycle, and in the second case is the running of Matrix multiplication.

It should be noted that for running the experiments on MPICH1 we use "mpirun" command and “mpiexec” for running it on MPICH2. For example, the steps of the LAPW0 execution on MPICH2 are shown in figure (2).

```
[rezek@rezek-dell15~]$ cd/home/
rezek /mpich2 /examples
[rezek@rezek-dell15 examples]$
mpicc -c lapw0_mpi.c
[rezek@rezek-dell15 examples]$
mpicc -o lapw0_mpi lapw0_mpi.o
[rezek@rezek-dell15 examples]$ mpd &
[1] 3929
[rezek@rezek-dell15 examples]$
mpiexec -n 1 lapw0_mpi
lapw0_mpi has started with 1 tasks.
Initializing arrays...

Running Time = 62.005132

Done.
[rezek@rezek-dell15 examples]$
mpiexec -n 2 lapw0_mpi
lapw0_mpi has started with 2 tasks.
Initializing arrays...

Running Time = 34.002134

Done.
[rezek@rezek-dell15 examples]$
mpiexec -n 3 lapw0_mpi
lapw0_mpi has started with 3 tasks.
Initializing arrays...

Running Time = 25.141348

Done.
```

Fig 2 : Screen Shot of Running LAPW0 on MPICH2

The results of the average running time for case 1 (LAPW0) are summarized in table 3. This table shows the execution time on MPICH1 and MPICH2 and the improvement factor (*if*) by the number of processors. Where the improvement factor (*if*) is measured as the ratio of the difference between the execution time on MPICH1 and MPICH2 to the Execution time on MPICH1 i.e., $(T_{MPICH1}-T_{MPICH2})/ T_{MPICH1}$.

$$if = \frac{T_{MPICH1} - T_{MPICH2}}{T_{MPICH1}}$$

It is clear that the performance of MPICH2 is better than MPICH1 by approximately 3%. Also, Figure 3 shows the difference between the execution time on MPICH1 and MPICH2.

Table 3: Execution Time of LAPW0 on MPICH1 and MPICH2 on Different # of Processors.

# of Proc	*Exec. time on mpich1 (min)*	*Exec. time on mpich2 (min)*	*If*
1	64.25	62.54	0.026615
2	35.05	34.38	0.019116
3	26.03	25.37	0.025355
4	20.5	19.52	0.047805

Recall that in case 2 matrix multiplication program for matrices of size (5120 x 5120) were running using MPICH1 and MPICH2 on one, two, three, and four processors. The results of the average running time are summarized in table 4 and depicted in Figure 4. Again it is clear that the performance of MPICH2 is better than MPICH1.

The results of the experiments in case 1 and case 2 assess the improvement of MPICH2 over MPICH1 which has significant results on the performance and efficient utilization of resources. Note that the time units in case 1 are in minutes, whereas it is in seconds in case 2.

Consequently, in all cases MPICH2 is better than MPICH1. Therefore, we believe that the nine added features have positive impact on the performance. The most important added features in MPICH2 are the collective communications, the support of one sided communication, MPI Thread Multiple, and its concern on communication rather than process management.

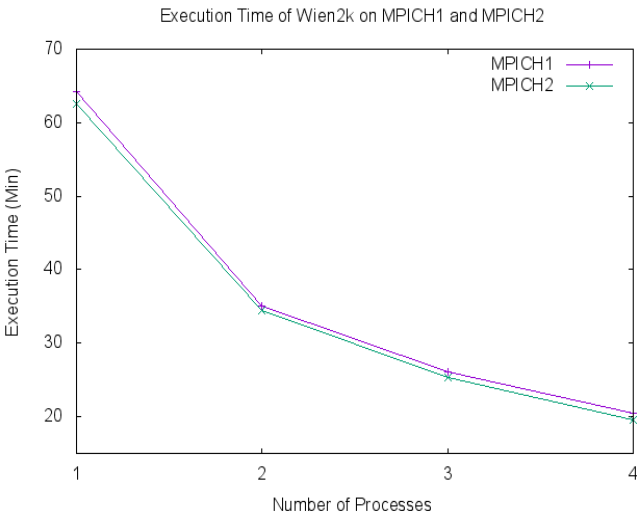

Fig 3: the WIEN2K execution time of MPICH2 vs. the execution time of MPICH1.

Table 4: Execution Time of Matrix Multiplication on MPICH1 and MPICH2 on Different # of Processors.

# of Proc	*Exec. time on mpich1 (sec)*	*Exec. time on mpich2 (sec)*	*If*
1	92.357	89.562	0.030263
2	63.109	61.776	0.021122
3	60.910	59.113	0.029503
4	57.965	55.935	0.035021

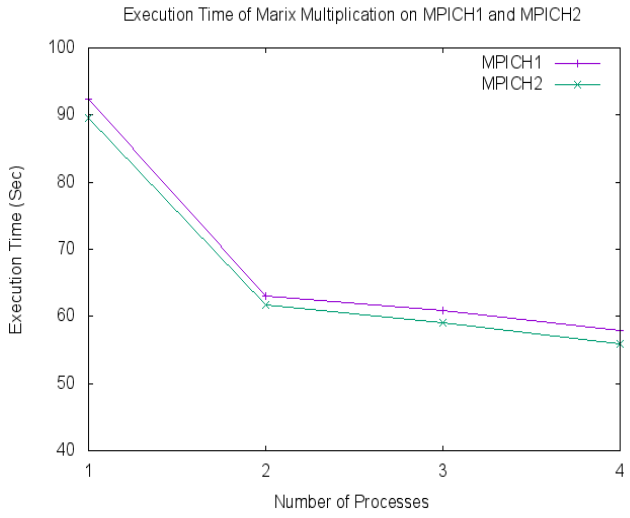

Fig 4: Execution Time of Matrix Multiplication Using MPICH1 vs. MPICH2

CONCLUSION AND FUTURE WORKS

The goal of this work is to evaluate and compare the performance of MPICH1 and MPICH2 using different cases running on one, two, three, and four processors. As a result we can conclude that MPICH2 perform better than MPICH1. This is due to the collective improvement and added features in MPICH2.

Finally, as a future work we intend to extend our experiment to test the performance of newly issued MPICH3 using different tasks.

REFERENCES:

[1] Sherihan Abu ElEnin, Mohamed Abu ElSoud,*"* Evaluation of Matrix Multiplication on an MPI Cluster*"* Faculty of computers and Information, Mansourauniversity, Egypt. 2011

[2] Dami´an A. Mall´on, Guillermo L. Taboada, Carlos Teijeiro, Juan Touri˜no, Basilio B. Fraguela, Andr´es G´omez1, Ram´on Doallo, and J. Carlos Mouri˜no1,*"* *Performance Evaluation of MPI, UPC and OpenMP on Multicore Architectures".*Galicia Supercomputing Center (CESGA), Santiago de Compostela, Spain. Computer Architecture Group, University of A Coru˜na, A Coru˜na, Spain. 2009

[3] David Culler. Jaswinder Pal Singh, Anoop Gupta.*"* *Parallel Computer Architecture A Hardware / Software Approach ".* University of California, Berkeley, Princeton University, Stanford University, Aug 28, 1997, Pages 40 -127.

[4] *"*MPI: A Message-Passing Interface Standard, Message Passing Interface Forum*"*. ARPA and NSF under grant ASC-9310330, the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615, by the Commission of the European Community through Esprit project P6643. Nov 15, 2003

[5] *"*MPI: A Message-Passing Interface Standard, Version 2.1, and Message Passing Interface Forum*"*. June 23, 2008

[6] EDOUARD BUGNION, SCOTT DEVINE, KINSHUK GOVIL, and MENDEL ROSENBLUM, "*Disco: Running Commodity Operating Systems on Scalable Multiprocessors*", Stanford University, November 1997, Vol. 15, No. 4, Pages 412–447.

[7] S. Cottenier, "Density Functional Theorythe Family of (L)APW-methods: a step-by-step introduction", August 6, 2004, ISBN 90-807215-1-4.

[8] Rezek Mohammad, Areej Jabir, and Rashid Jayousi, *"Optimum Execution For WIEN2K using Parallel Programming Models (Comparison Study)"*. Department of physics, Palestinian Technical University/Khadoorie, Middle East Technical University, and department of Computer Science, Al-Quds University, Jerusalem, Palestine. 2011.

[9] Schrodinger¸ E. *"An Adulatory Theory of the Mechanics of Atoms and Molecules"*. Physical Review 28 (26): 1049-1070. 1926.

[10] Hellmann¸ Hans, *"A new Approximation Method in the Problem of Many Electrons"*. Journal of Chemical Physics (Karpow-Institute for Physical Chemistry,Moscow), 1935.

[11] M´arcia C. Cera1¸ Guilherme P. Pezzi¸ Maur´ıcio L. Pilla, Nicolas B. Maillard1, and Philippe O. A. Navaux, , *"Scheduling Dynamically Spawned Processes in MPI-2"*. Universidade Federal do Rio Grande do Sul, Porto Alegre Brazil and Universidade Cat´olica de Pelotas, Pelotas, Brazil).

[12] C.M. Maynard, *"Comparing One-Sided Communication with MPI, UPC and SHMEM"*. EPCC, School of Physics and Astronomy, University of Edinburgh, JCMB, Kings Buildings, Mayfield Road, Edinburgh, EH9 3JZ, UK.

[13] Weihang Jiang, Jiuxing Liu, Hyun-Wook Jin, Dhabaleswar K. Panda, William Gropp and Rajeev Thakur, *"High Performance MPI-2 One-Sided Communication over InfiniBand"*. Computer and Information Science The Ohio State University Columbus, OH 43210 Mathematics and Computer Science Division Argonne National Laboratory Argonne, IL 60439.

[14] *"*MPI: A Message-Passing Interface Standard, Version 2.2, and Message Passing Interface Forum*"*. **Sept 4, 2009**

[15] William Gropp, Ewing Lusk, David Ashton, Pavan Balaji, Darius Buntinas, Ralph Butler, Anthony Chan, Jayesh Krishna, Guillaume Mercier, Rob Ross, Rajeev Thakur, and Brian Toonen,*" MPICH2 User's Guide,* Version 1.0.6, Mathematics and Computer Science Division Argonne National Laboratory*"*. **September 14, 2007**