КИБЕРНЕТИКА

МРНТИ 28.29.53

A.S.Kussainov[1], A.K.Beisekov[1], G.B.Turmaganbet[1]

[1]al-Farabi Kazakh National University, Almaty, Kazakhstan

DEPLOYMENT AND TESTING OF THE PARALLEL ENVIRONMENT FOR THE QUANTUM VARIATIONAL MONTE CARLO METHOD

Abstract. It is formulated a brief introduction for the beginner programmer scientists in the parallel and multi-threaded programming. The example of fast installation of minimal set of tools for the parallel programming, the creation of own multi-threaded program, its compilation, debugging and starting was given. Open MPI software package provides the realization of programming interface of MPI for the exchange of messages between processes, also it is considered as the example and the tool of parallel programming for the increase of performance of multi-core and distributed computing systems. It is presented the example of program, which is realizing variational method of Monte Carlo for the salvation of Schrodinger equation in case of quantum harmonic oscillator, written for the multi-core and distributed computing systems. It is given the explanation with the comments of transition from written single-threaded sequential program to C++ parallel program.

Key words: Parallel programming, OpenMPI, variational method of Monte Carlo, Schrodinger equation, harmonic oscillator

Аннотация. Сформулировано краткое введение для начинающего ученого-программиста в параллельное и многопоточное программирование. Приведен пример быстрой установки минимального набора средств для параллельного программирования, написания собственной многопоточной программы, ее компиляции, отладки и запуска. Программный пакет OpenMPI, представляющий собой реализацию программного интерфейса MPI для обмена сообщениями между процессами, рассмотрен как пример и инструмент параллельного программирования с целью увеличения производительности многоядерных и распределённых вычислительных систем. Представлен пример программы, реализующей вариационный метод Монте-Карло для решения уравнения Шредингера в случае квантового гармонического осциллятора, написанной для многоядерных и рас-

пределённых вычислительных систем. Дано описание с комментариями перехода от написанной однопоточной последовательной программы на C++ к параллельной программе.
Ключевые слова: параллельное программирование, OpenMPI, вариационный метод, метод Монте-Карло, уравнение Шредингера, гармонический осциллятор.

Түйіндеме. Бастаушы ғылыми-программисттің параллель және көпағынды программалауға арналған қысқаша кіріспесі қалыптастырылған. Параллель программалауға арналған минималды қ ұралдар жиынтығын жылдам қондыру, өзіндік көпағынды программалардың жазылуы, оның компилляциялары, ретке келтіру мен іске қосу мысалдары келтірілген. Процесстер арасындағы хабар алмастырушы MPI программалық интерфейсінің жүзеге асуын көрсетуші OpenMPI программалық пакеті көпядролы және кең таралған есептеу жүйелерінің өнімділігін арттыру үшін параллель программалаудың мысалы мен құралы ретінде қарастырылған. Көпядролы және кең таралған есептеу жүйелеріне жазылған кванттық гармоникалық осциллятор кезіндегі Шредингер теңдеуін шешуге арналған вариациялық Монте-Карло әдісін іске асырушы программа мысалы келтірілген. Параллель программаға C++-те бірағынды ретті жазылған программаға сипаттама мен түсіндірмелер берілген.
Түйінді сөздер: параллель программалау, OpenMPI, вариациялық Монте-Карло әдісі, Шредингер теңдеуі, гармоникалық осциллятор.

Introduction

High performance personal desktops or workstations, as well as clusters and high performance computing systems which could be accessed remotely, became widely available nowadays. Many of these are state-of-the-art, expensive machines maintained by the numerous staff and capable to address the fundamental pure and applied science problems of our days [1-2]. In case of the later ones, the researcher could directly proceed to the coding of the parallel program, delegating the tedious task of network and parallel environment configuration to a network administrator. This type of IT professionals, though possessing the vast knowledge and experience in networking protocols, most likely will make you to work with the closed box solution, even though the creation and configuration of your own computational cluster is relatively easy task for today's scientist.

To fill this gap in the professional IT training the multiple studies and help resources have been printed and circulated among the interested scientists and researches [3, 4]. Research universities around the world including Kazakhstan are in possession or include in their strategic development plans supercomputers and HPC (high-performance computing) systems [5].

There is another important moment to address. Many beginner programmers though have in their possession the state-of-the-art multicore computing system are able to utilize only a small portion of its computational power. All modern computers have the multicore central processing units (CPUs). Unless you have a sophisticated compiler which could parse your, let us say C++, code to run simultaneously on all the cores, regularly you will have only one of eight cores, for octet CPU, to handle your task. You need to multithread your application by yourself because each thread of execution can only saturate one core [6].

These two moments are addressed at once by using the OpenMPI library [7] which can handle multithread coding and feed it to a multicore CPU or distribute the tasks across the network of computers connected into a computational cluster. Unlike the similar, OpenMP, development of the message parsing protocols [8] it is mainly and extensively documented in electronic resources and much easier to deploy for the beginner.

Additionally, one could make himself comfortable with parallel programing before attempting to learn and configure the complex networks and investing in buying equipment or machine time on high-performance computing systems.

We have selected variational Monte Carlo method for the Schrodinger equation [9] of the quantum harmonic oscillator to implement numerically as an excellent example of multithreading and performance optimization in scientific computing [10].

Methods

In our work the Debian distribution of the Linux class operational systems was made an operational system of choice mainly because of its flexibility, freeware nature, and more than modest requirements for an existing hardware. You will have multiple versions available for download free of charge at [11]. To successfully run a fully functional

version with graphical desktop, parallel computation library, ssh client/ server etc we need the modest 256 megabytes of RAM and several gigabytes of hard disk space. The Windows OS users may install an Oracle virtualbox software [12] and populate it with any Linux installation of their choice or install a second OS with ability to select it at the boot time.

One should take caution to install software and OS of the same register size, that is 32-bit or 64-bit (amd64 or i386 packages), across your complex computational system and virtual environment.

Please be aware, that if your computer is old enough you may came across the problem of it not supporting 64-bit software and virtualization technology. From now on, we assume that your computer's CPU has more than one core.

Using OpenMPI is the easiest and quickest way to learn parallel programming and maximize the performance of your desktop system. This package is always included in full installation DVDs, CDs or online depositories and may be installed by the following command:

$$\$ \text{ apt-get install openmpi-bin openmpi-common libopenmpi 1.6 libopenmpi-dev} \quad (1)$$

Depending on the state of your system, some of these components may already be installed or unavailable and you will be offered with an alternative.

For a numerical problem to calculate we chose the variational Monte Carlo method to solve the Schrodinger equation for the harmonic oscillator. We start with the following, one dimensional, time independent Schrodinger equation

$$-\frac{\hbar^2}{2m}\frac{d^2\varphi(x)}{dx^2}+\frac{1}{2}mk^2x^2\varphi(x)=E\varphi(x) \quad (2)$$

where k is the force constant. If we know the complete set of N eigenfunctions in the following form

$$\Phi(x)=\sum_{i=0}^{N-1}c_i\varphi_i(x) \quad (3)$$

the average value of energy <E> will be given by the these expressions

$$\langle E\rangle = \frac{\int_{-\infty}^{+\infty} dx\Phi(x)^{*}H\Phi(x)}{\int_{-\infty}^{+\infty} dx\Phi(x)^{*}\Phi(x)} = E_0 + \frac{\sum_{i=0}^{N-1}|c_i|^2(E_i - E_0)}{\sum_{i=0}^{N-1}|c_i|^2} \quad (4)$$

here H is the Hamiltonian given by the left part of equation (2) and E_0 is the ground state energy. Variational Monte Carlo method uses equation (4) as a starting point. Replacing the unknown set $\Phi(x)$ of eigenfunctions by the trial wavefunction $\Phi_{T,a}(x)$ we are then varying he parameter a, see equation (5). If we are lucky, the calculated average energy at the local minimum will give as a ground state value and corresponding characteristic wave function.

$$\langle E\rangle = \int_{-\infty}^{+\infty} dx\omega(x)E_L(x), \text{ where } \omega(x) = \frac{|\Phi_{T,a}(x)|^2}{\int_{-\infty}^{+\infty} dx'|\Phi_{T,a}(x')|^2} \text{ and } E_L(x) = \frac{H\Phi_{T,a}(x)}{\Phi_{T,a}(x)} \quad (5)$$

One of the main features of this method is that we do not sample the whole configuration space from minus to plus infinity indiscriminately rather than traversing it in the manner described as Metropolis-Hastings algorithm. Target wave function's tails go to zero pretty fast even not far away from the origin and its overall shape is similar to the normal distribution. The Metropolis-Hastings algorithm [13], which samples the space according to the weight function $\omega(x)$ in equation (5), is a Markov chain Monte Carlo (MCMC) algorithm.

We place n of the so called walkers randomly and uniformly across the selected region at coordinates $(x_n)_t$ (index t stands for the current state of walkers' coordinates). The next set of coordinates $(x_n)_{t+1}$ of walkers, who are sampling the integral in equation (4) independently, is determined by the following set of rules:

$$x_{t+1} = x_t + \delta\,,$$
$$\text{if } \frac{\omega(x_{t+1})}{\omega(x_t)} \geq 1 \text{ the step is accepted}\,, \quad (6)$$
$$\text{if } \frac{\omega(x_{t+1})}{\omega(x_t)} < 1 \text{ accept it only if it's larger than random number on } (0,1)$$

where δ is the size of step which is determined based on the problem's conditions. If this step is accepted the local energy $E_L(x)$ calculated at this walker's coordinate contributes to the integral in equation (5).

If the trial function chosen to be $\Phi T_a(x) = exp(-ax^2)$, the value of $\langle E \rangle$ is than calculated according to the formula

$$\langle E \rangle = \frac{1}{counts} \Sigma_{i=1}^{counts} E_L(x_i), \text{ where } E_L(x) = a + x^2(0.5 - 2a^2) \quad (7)$$

where *counts* are the number of accepted steps across all walkers' trajectories.

Here we used expression for E_L derived from equation (5). We also assume that in Hamiltonian, see equation (2), we choose the values of $h=m=k=1$. It is very neat numerical problem to implement in a single thread and in parallel.

Results and Discussions

The problem itself is pretty straightforward to formulate using C++ programming language, see Table 1. We have structured the code in such a way that if one decides to comment out the code lines printed in boldface font he will end up with the regular single thread program. The parallel version is compiled and run in 8 threads through the following set of commands

```
$ mpicxx \-o Jan06_vmc_paral Jan06_vmc_paral.cpp
$ mpirun -np 8 Jan06_vmc_paral
```
(8)

while the single thread version is compiled and run in a more conventional fashion

```
$ g++ -o arman_vmc arman_vmc.cpp
$./arman_vmc
```
(9)

If you are running a parallel program on a cluster of individual machines the second line in equation (8) is replaced with

```
$ mpirun --hostfile my_hostfile -np 8 Jan06_vmc_paral
```
(10)

where the text file my_hostfile specifies the configuration of your network and the number of cores per individual CPU.

Table 1

C++ code used to calculate the range of ground state energy values as a function of parameter a. In bold font are given the pieces of code which convert a single thread program to a multithread

```
1  #include <mpi.h>                          //header file to provide parallel programing environment
2  #include <cstdlib>
3  #include <iostream>
4  #include <cmath>
5  using namespace std;
6  double a,e,sum_e,*x;
7  int c_ounter;
8  int recv_data[2]={2400, 20000}, *send_data;   //walkers' number and Monte Carlo steps values
9  double senback_data[2], *collect_data;        // arrays to collect energy and accepted steps data from individual parallel process
10 int id, ntasks, len;
21 double w(double xt, double x) {
22    return exp(-2*a*(xt*xt - x*x));}           //the ratio of the weight function computed for consecutive x values
23 double e_nergy(double x) {
24    return a+x*x*(0.5-2*a*a);}                 // local energy function
25 void Assign_Positions(){                      //function to initialize starting point for all walkers
26 srand(time(NULL)*id+1);
27 x = new double [recv_data[0]];
28 for (int i = 0; i < recv_data[0]; i++){
29          x[i] = rand()/(RAND_MAX + 1.0) - 0.5;}}      //walkers uniformly distributed within (-0.5:+0.5) range
30 void Stir_All_Walkers(){                                //function to move all walkers according to Metropolis algorithm
31 for(int j=0;j < recv_data[0]; j++){
32          double xt=x[j]+pow(-2.0*log(rand()/(RAND_MAX+1.0)),0.5)* // Box-Muller transform to generate normal distribution
33                    cos(2.0*3.141592*rand()/(RAND_MAX+1.0));
34          if(w(xt, x[j])>1){
35                    x[j] = xt;++c_ounter;e = e_nergy(x[j]);sum_e += e;}
36          else{
37                    if (w(xt,x[j])>rand()/(RAND_MAX+1.0)){
38                              x[j] = xt;++c_ounter; e= e_nergy(x[j]);sum_e += e;}}
39 }}
40 int main(int argc, char *argv[]){
41 MPI_Init(&argc, &argv);
42          MPI_Comm_size(MPI_COMM_WORLD, &ntasks);
43          MPI_Comm_rank(MPI_COMM_WORLD, &id);
44          if(id==0){                              //subroutine to evenly distribute walkers between the claimed number of processes
45                    int WperTask = floor(recv_data[0]/ntasks);
46                    send_data = new int [2*ntasks];
47                    for (int i = 0; i < ntasks; i++)
48                              {send_data[2*i]=WperTask;
49                               if(i<=recv_data[0]%ntasks-1){send_data[2*i]++;}
50                               send_data[2*i+1]=recv_data[1];}
51                    collect_data = new double [2*ntasks];}
52          MPI_Scatter(send_data,2,MPI_INT,          //sending assigned number of walkers and individual number of step
53          recv_data,2,MPI_INT,0,MPI_COMM_WORLD);    //to each process
54          Assign_Positions();
55          for(a=0.1;a<=1.5;a+=0.05){
56                    for(int k=0;k<=floor(0.2*recv_data[1]);k++){Stir_All_Walkers();}
57                    double avg_e=sum_e=0;c_ounter=0;
58                    for(int k=0;k<=recv_data[1];k++){
59                              Stir_All_Walkers();}
60                    avg_e = sum_e/c_ounter;
61                    senback_data[0]=sum_e;senback_data[1]=c_ounter;   //collecting data back from all processes
62          MPI_Gather(senback_data,2,MPI_DOUBLE,                       //and calculating alpha and average energy values
63          collect_data,2,MPI_DOUBLE,0,MPI_COMM_WORLD);
64                    if(id==0){double total_e=0;double total_count=0;
65                              for(int i=0;i<ntasks;i++){
66                                        total_e+=collect_data[2*i];
67                                        total_count+=collect_data[2*i+1];}
68                              cout<<a<<"\t"<<total_e/total_count<<"\n";}
69                    c_ounter=0;}
70          MPI_Finalize();
71          exit(0);}
```

The configuration of our computer is listed as follows: Intel Core i7 4790K, 4.0GHz/LGA-1150/22nm/Haswell/8Mb L3 Cache/ IntelHD4600/EM64T, DDR-3 DIMM 16Gb/1866MHz PC14900 Kingston HyperX Fury Black, 2x8Gb Kit, CL10.

The main features of our code are the following: we split, see the lines 41-53, the total number of walkers uniformly between the claimed number of threads, which is eight, see equation (8); then we used the Box-Muller transform [14] to generate the pairs of independent, standard, normally distributed with zero expectation and unit variance random numbers, given a source of uniformly distributed random numbers from standard C++ rand() generator, see the lines 32-33; the *MPI_Scatter* and *MPI_Gather* were the *MPI* functions to facilitated our data exchange between the threads, see the lines 52 and 62. We have not made an additional effort to optimize the step's size or the standard random generator's quality.

The data obtained from our simulations are plotted on Figure 1. The left part represents the plot of the average energy versus parameter a. As expected, we can clearly see the minimum at a=0.5 where the local energy function has no dependence on a. This value of a corresponds to the zero variance value on the right plot which pinpoints the ground state in our problem.

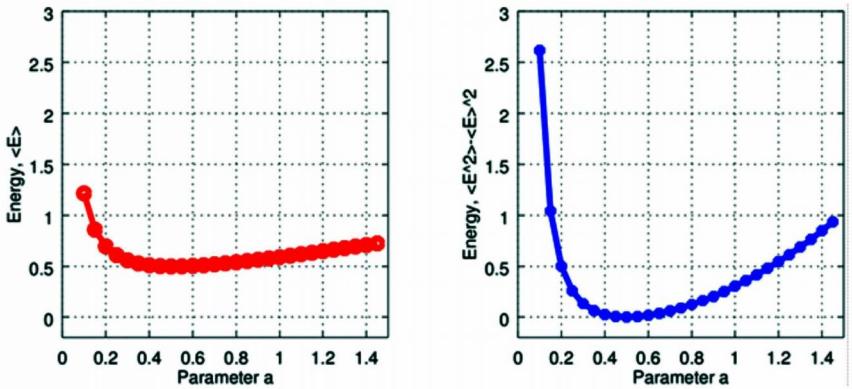

Figure 1. Numerical simulation data for the quantum harmonic oscillator. The average energy value <E> on the left and the quantity $<E^2>-<E>^2$ on the right are both plotted as the functions of parameter *a*

Table 2

Performance comparison between the parallel and single threaded implementations of the variational Monte Carlo algorithm

	Wall time	CPU time
Parallel algorithm timing (averaged between 8 threads). Process name is Jan06_vmc_paral	54.8723	53.709
Single threaded one. Process name is arman_vmc	275.519	275.361

For the reader's info, CPU time is the time which is actually spent by CPU to work on the process, while the Wall time additionally includes the time spent by the process in the line awaiting to be handled plus some other delays. How the work load is now uniformly spread across the processes is clearly seen from the Figures 2 and 3. The single instance of arman_vmc process is managed singlehandedly by one core #2, see Figure 2. While in case of the parallel program all eight cores of our CPU are loaded with their own fraction of work, running 8 instances of the Jan06_vmc_parall simultaneously each with their own parameter, see Figure 3.

Conclusions

As we can see, the relatively straightforward modification of our system allows us to run multiple parallel algorithms and increase productivity of our code in many times. The coding and experience gain in such an exercise is a good start in transition to the distributed and high performance computations. Implemented variational Monte Carlo method is a key tool for many computationally extensive and effective numerical methods in science.

Acknowledgments

This research was supported by grant №3824/ГФ4 provided by the Science Committee at the Ministry of Science and Education of Republic of Kazakhstan to the principal investigator at the National Nanotechnology Laboratory of Open Type, Physics and Technology Department, al-Farabi Kazakh National University.

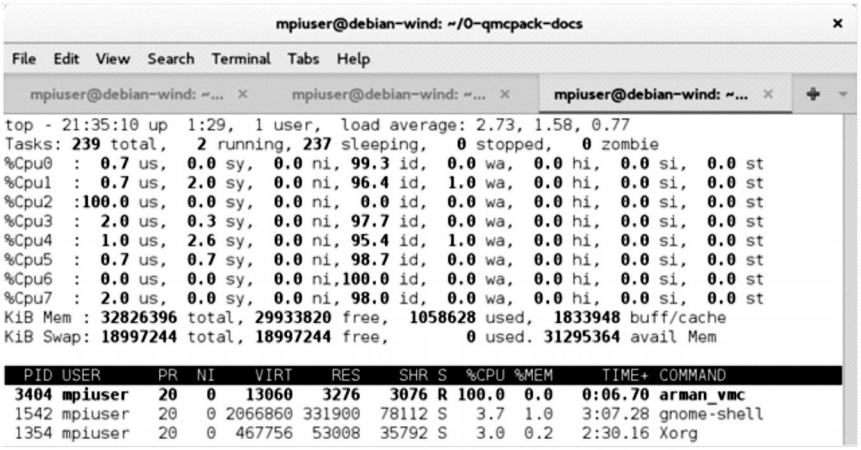

Figure 2. Output of the top command displaying info about the state of the individual cores of our CPU for a single thread C++ program of variational Monte Carlo simulation

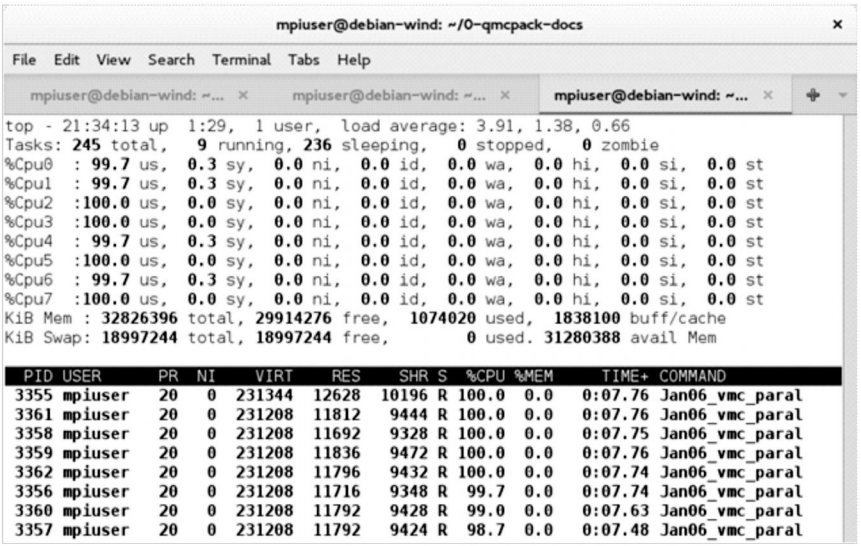

Figure 3. Output of the top command displaying info about the state of the individual cores of our CPU for the multithreaded C++ program of variational Monte Carlo simulation

References

1 Supercomputer Sites [web data base]. Top 500. – The list of the 500 most powerful computer systems. [Электрон. ресурс]: 2015. – http://www.top500.org/lists/2015/11/

2 *Sadaf R., Alam* et al. An Evaluation of the Oak Ridge National Laboratory Cray XT3 // International Journal of High Performance Computing Applications. – 2008. – V.22. – № 1. – P. 52-80.

3 *Gergel' V.P.* Vysokoproizvoditel'nye vychisleniya dl mnogoyadernyh mnogoprocessornyh sistem. Uchebnoe posobie //Nizhnij Novgorod: Izd-vo NNGU im.N.I.Lobachevskogo, 2010. – 421 s.

4 *Scott L.R., Clark T.* and *Bagheri B.* Scientific Parallel Computing. Princeton, NJ, USA // Princeton University Press, 2005. – 392 p.

5 *Abdrahmanov R.* Superkomp'yuter ot partnyora: superkomp'yuter stoimost'yu 10 mln. dollarov SSHA poluchit KazNU im. al'-Farabi po grantu pravitel'stva KNR // Vechernij Almaty. – 2015. – 12 sentyabrya.

6 *Akhter S., Roberts J.* Multi-core Programming: Increasing Performance Through Software Multi-threading. Intel Press, 2006. – 336 p.

7 OpenMPI project [web data base]. – Information and resources. [Электрон. ресурс]: – http://www.open-mpi.org/

8 *Chapman B.* Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation). – The MIT Press: Scientific and Engineering Computation edition, 2007. – 384 p.

9 *Kashurnikov V.A., Krasavina V.*. Vychislitel'nye metody v kvantovoj fizike: Uchebnoe posobie. – M.: MIFI, 2005. – 412 s.

10 *Voevodin V.V., Voevodin Vl.V.* Parallel'nye vychisleniya // SPb.:BHV-Peterburg,2002. – 608 s.

11 Debian OS [web data base]. – Information and resources. [Электрон. ресурс]: – https://www.debian.org/

12 Oracle software and applications [web data base]. – Free

access information and resources. [Электрон. ресурс]: – https:// www.virtualbox.org/

13 *Metropolis N., Rosenbluth A.W., Rosenbluth M.N., Teller A.H., Teller E.* Equations of State Calculations by Fast Computing Machines // Journal of Chemical Physics. – 1953. – № 21 (6). – P. 1087-1092.

14 *Box G.E.P., Muller M.E.* A Note on the Generation of Random Normal Deviates // The Annals of Mathematical Statistics. – 1958. – Vol. 29. – № 2. – P. 610-611.

***Кусаинов Арман Саинович**, кандидат технических наук, PhD, e-mail: arman.kussainov@gmail.com*

***Бейсеков Алтынбек Кудиярбекович,** магистрант, e-mail: arman.kussainov@gmail.com*

***Тұрмағанбет Гүлнұр Бүркітқызы,** магистрант, e-mail: turmaganbet.gulnur@mail.ru*