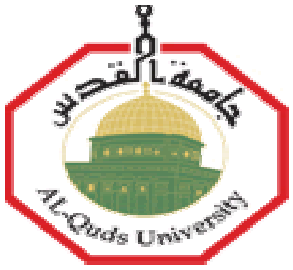
Dean of Graduate studies

AL-Quds university

Computer Science



Towards Optimum Execution For Wien2k Using Parallel Computing Models (Comparative Study)

Areej Mustafa Mahmoud Jabir

M.Sc. Thesis

Jerusalem-Palestine

1432/2011

Towards Optimum Execution For Wien2k Using Parallel Computing Models (Comparative Study)

Prepared by

Areej Mustafa Mahmoud Jabir

A thesis submitted in partial fulfillment of requirements for the degree of master of Computer Science Department

B.Sc Alquds Open University Palestine

Supervisor: Dr. Rashid Jayousi

Co-supervisor: Dr. Rezek Mohammad

1432/2011

Al Quds University

Deanship of Graduate Studies

Computer Science

Thesis approval

Towards Optimum Execution for Wien2k Using Parallel Computing Models
(Comparative Study)

Prepared by: Areej Mustafa Mahmoud Jabir

Registration number: 20810099

Supervisor: Dr. Rashid Jayousi

Co-supervisor : Dr. Rezek Mohammad

Master thesis submitted and accepted, Date……

The names and signature of the examining members are as follows:

1-Head of committee: Dr. Rashid Jayousi Signature……

2- Internal Examiner: Dr. Nidal Kafri Signature……

3- External Examiner Dr. Nael Salman Signature……

4-Committee Member: Dr. Rezek Mohammad Signature……

Jerusalem- Palestine

1430/2011

Dedication

To my beloved Country Palestine

To my parents and my family

To all those who believe in the richness of learning

Areej Jabir

Declaration

I certify that this thesis submitted for the degree of master, is the result of my own research, except where otherwise acknowledged, and that this study (or any part of the same) has not been submitted for higher degree to any other university or institution.

Signed …….

Areej Mustafa Mahmoud Jabir

Date :…….

Acknowledgment

I would like to express my sincere feelings to my supervisors, Dr. Rezek Mohammad and Dr. Rashid Jayousi. I am grateful for their painstaking care in the course of this project, for their meticulous effort in teaching me very precious, numerous concepts. I could have done nothing without both of them.

I would like to thank also all the people in the Computer Science department for their endless help during my study.

I would also very much like to express my gratitude to my family, my mother and father who always supported me in whatever direction I decided to go in life

A thank you to all my friends, colleagues at works especially Nida' Hamdan for the interest you have shown in my work, and all their encouragement.

Areej Jabir

Abstract

In the present work, a comparison between two parallel methods have been done using message passing with OpenMPI API. Both Distribute k-point method and Data distribution method have been used to run physical package which is used to study the physical and chemical properties of the materials which is called Wien2k package. Two data set size are used to be as a benchmark of this study, execution time of running with respect to RAM size either in shared or distributed case has been studied, two different size of RAM per CPU has been tested in Distribute k-point for the two benchmark, respectively. Network speed and its effect on calculation time has been studied, the network speed used in the study was 100 Mbps and 1 Gbps, the network effect has been tested on Distribute k-point method for two classes of CPUs and different RAM size per CPUs . The effect of CPUs speed on execution time has been studied by distribute k-point and Data distribution methods; different speed of CPUs has been used to study homogenous and heterogeneous effects on execution time. In all tests, RAM size considerably affect the time of calculation effectively, it's found that increasing size of RAM available per CPU will cause considerable decrease in the calculation time, the study showed a small effect for the network speed on the calculation

time, the effect of network can be neglected with respect to RAM size effect, The execution time showed that Data distribution gives better reduction in the time of calculation and higher speed up factor with increasing number of CPU's, two scalable quantity has been used to compare and analyze the results, speed up factor and the power factor of decaying formula for time of execution.

ملخص

في هذه الدراسة , تم عمل دراسة مقارنة بين نموذجين للبرمجة المتوازية باستخدام message passing with OpenMPI API, كلا النموذجين وهما نموذج توزيع قيمة متجه الموجة ونموذج توزيع البيانات قد تمت تجربتهما على حزمي برمجية تهتم بدراسة الخصائص الفيزيائية و الكيميائية للمواد و التي تسمى حزمة WIEN2k البرمجية, لقد تم اختبار نقطتا مرجع تمثل حالتين مختلفتين لتشغيل برنامج Wien2k في كل الاختبارات, لقد تم دراسة تأثير حجم الذاكرة المخصصة لكل مسرع علي زمن التشغيل سواء كانوا مجتمعين بنظام المشاركة أو بنظام المنفصل , إن سرعة الشبكة التي تم استخدامها في الفحص هي 100 Mbps و 1Gbps , فحص تأثير الشبكة علي زمن التشغيل باستخدام طريقة القيمة المتجهة وبتشغيل المرجعين و لسرعات مختلفة للمسرعات و لسعات ذاكرة مختلفة, إن تأثير سرعة المسرعات علي زمن التشغيل قد تم فحصه باستخدام الطريقتين وهما القيمة المتجهة و توزيع البيانات و لسرعات مختلفة و التي من خلالها تم دراسة تأثير التركيب المتجانس و الغير متجانس للمسرعات.

لقد وجد أن حجم الذاكرة له التأثير الأقوى من بين العوامل الأخرى كسرعة الشبكة و سرعة المسرعات, و نتيجة لذلك وجد أن زيادة حجم الذاكرة المخصص لكل مسرع يعطي نقصا كبيرا في زمن التشغيل, ووجد أيضا أن سرعة الشبكة قليل جدا بالمقارنة مع تأثير حجم الذاكرة المخصص لكل مسرع.

لقد أثبتت الدراسة أن طريقة توزيع البيانات بأنه الأفضل في توفير زمن التشغيل عند ثبات الذاكرة و عدد المسرعات.

Table of Contents

List of tables

List of figures

Introduction

Since the Von Neumann computer architecture [1] and the Integrated circuits one can observe a rapid development in computer performance in order to satisfy the increasing needs to speed up accomplishing the tasks that it serves. One of the ways to speed up solving these task is parallel computing.

Traditionally software has been written to be executed in a serial way, which means that instructions will be executed one after another, and only one instruction may be executed in any moment of time, some complex application will take months to execute. There are many constraints that prevent to build very fast serial computers; transmission speed, limits to miniaturization, and economic limitations. For last two decades, trends was going to parallel computing, the program is broken down into discrete block, and every block will be executed simultaneously. Such techniques proved to be beneficial in many areas, increasing the efficiency in terms of speeding up the computation time, i.e. processing huge amount of information in shorter time, and economically much cheaper as no need to build special parallel computers[2,3,4] . Recycled computers can be used as well current computer networks can be used to implement parallel algorithms. Parallel computing nowadays is popularly used in different fields of science and technology.

Designing parallel model consists of preparing both hardware (architecture) and software (programming model)[**5**]. The hardware can be built in one computer using multi processor, or in arbitrary computers connected via networks, or a combination between the

two. There are several programming models in common use; we will mention the most common of them:

1- Shared memory model where tasks share a common address space, which they read and write asynchronously, it's assumed that all data structures allocated in a common space that is accessible from every processor.
2- The message passing model, in which each processor is assumed to have its own private data space and data, must be explicitly moved between spaces as needed.

One of the famous models and the most effectively used is OpenMPI model [6]. It is an Application Program Interface (API), jointly defined by a group of major computer hardware and software vendors. OpenMPI provides a portable, scalable model for developers of non-shared memory parallel applications. The API supports C/C++ and Fortran on multiple architectures, including UNIX & Windows NT. OpenMPI is an Application Program Interface (API) also can be used to explicitly direct multi-threaded and non-shared memory parallelism.

The new generations of personal computers are multi thread processors, the aim of this study to search for optimum use of the recent new personal computers (PC), the three main factors that could affect programs executions are:

1- Memory size.

2- Processor Speed.

3- Network speed and internet Processor

Scientific computing is one of the most important fields that use the computational power of computers since the development of electrical digital computers presented by the Hangarian Von Neuman Architecture. However, the demand for higher performance computing power, urged the hardware and software engineers with the help of chemical, physical and mathematical scientist to develop and improve the technology that enable us to achieve this demand. Therefore, different improvement were added to Von Newman architecture. Even nowadays computers does not satisfy the requirement of a lot of applications. One of the ways that can be used toward the achievement of suitable performance is by using parallel computing. Therefore, the researchers developed different techniques to solve particular problems. Some of these problems are pure computations to solve problems or to simulate systems reality.

The importance of simulation can be described by the viewpoint that there are three pillars in scientific and technology research: analytic methods, experimental methods and numerical simulation [7], if we compare simulation with analytical models, we will find simulation could include more details and could be more flexible, which are impossible with simple analytical model. In comparison with real system, the researcher can control all the variables with their logical sequential in the experiment, in addition to the above mentioned , simulation experiments can examine non existing systems.

Recently simulation is a very helpful and valuable work tool in all areas of manufacturing. It can be used in industry allowing the system's behavior to be studied and tested. It can also provide a very low cost, secure and a very fast analytical tool. Testing the deployment of new systems is a very difficult task for any organization, whatever the size and specialty of this organization. For such systems simulation is the best solution for minimization the error in decision making and maximizing the economical benefit. Many alternative and possible solutions to the problem and its effects on production line can be tested. A comparison between these alternative solutions also can be made, such comparison can help in finding an optimal or near optimal solution. Researchers use simulation to study performance very often, simulation studies go back to the to the early 1960's [8].

One of the applications that needs a high performance computing in physical and chemical computation regarding material science is the problem of computing the cohesive energy. This energy is the main scalable quantity for measuring the stability of any material. There are different approaches and methods to compute the cohesive energy which developed through the past decades.

In the present work, we will study a package program named WIEN2K [9] that can simulate physical and chemical systems. This physical model is composed of different atoms supposed to combine and form a new material, this new material needs to study the physical, chemical, electrical, electronic and structural properties, the package studies all the possible combined phases of the material, these phases are governed by the laws of physics, the conditions necessary to yield such phases also can be predicted, these feedback information's is very necessary to the laboratory person, who can produce the

desired material in the suitable phase. Off course, one can imagine the result if we don't know the suitable conditions and the cost of such studies.

The aim of this work is to compare different approaches using Message Passing interface (MPI). In this work we tackle the computing of cohesive energy by comparing two of the most effective important approaches using parallel computing namely the message passing approach (MPI) using more than one networked multicore computers . The methods that we compared are the Distribute *k*-points and Data distribution

The thesis is organized as follows. The next chapter introduces the cohesive energy and the used approaches to solve it. Chapter 3 give a brief description on high performance computing development. After that, we illustrate the our parallel approach to solve this problem. In chapter 5, we introduce the experimental environment and implementation. Chapter 6. Discusses the results . Finally we conclude our work.

1. Chapter two:

Physical model Literature Review

Condense matter physics nowadays looks different than 50 or 15 years ago. Physicists know that solids obey the laws of quantum mechanics**,** by solving these equations all of properties of solids like structural, mechanical, thermodynamic, transport properties and electronic properties. Any material composed of many atoms combined together, according to the chemical bonding between these atoms, these atoms can take many positions while keeping the same total number of atoms of the material. Each stable form of these combinations gives different properties. [10, 11,12] .

In material science, 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 the combined form and the sum of the free atom's energy in their free state as shown in Fig. (1.1)

$$E_{\text{cohesive energy}} = E_{\text{compound}} - \sum E_{\text{Free atoms}} \qquad (2.1)$$

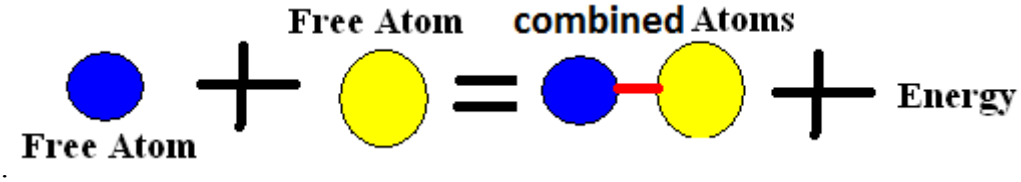


Figure 1.1 Atoms interaction

Each stable order of these atoms can produce positive value for the cohesive energy .The material normally can take more than one phase, the phase with the highest cohesive energy is the most stable one, see Fig.(1.2) and Fig.(1.3) which are drown using Wien2k package [11].

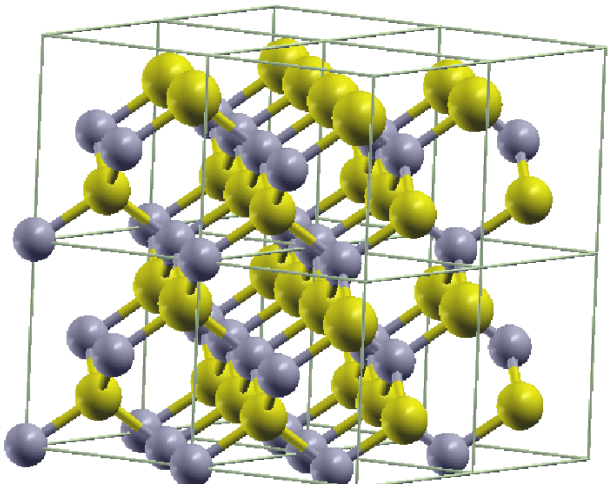


Figure 1.2: schematic diagram of Zincblind phase

.

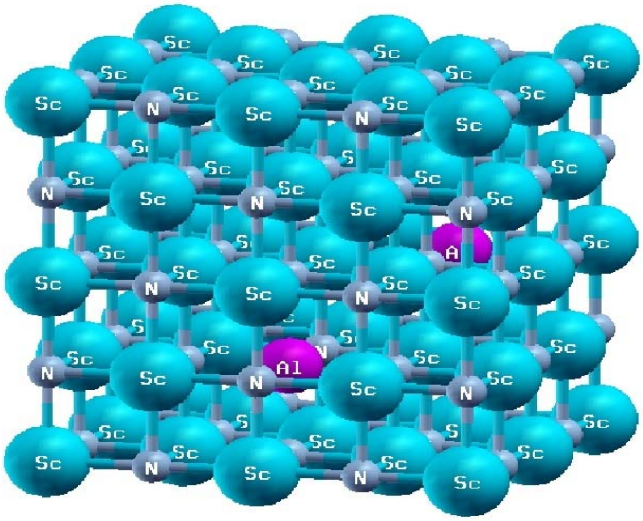


Figure 1.3 schematic diagram of simple cubic phase

Although solving quantum mechanics is easily said, In practice this turns to be a very hard numerical task, that take a long time even for a very idealized cases, that's why parametric calculations was used until 2000. In these calculations all the atomic and electronic interactions replaced by a scalar value taken from experimental results, which mean all the models used were to explain properties of materials already exist in the laboratory, some of famous methods were used to solve like this problems :

1- Pseudo potential method (PP) which first introduced by Hans Helmann (1930) [13].
2- Tight binding method (TB) which introduced in 1960.[14]

In addition to many other methods, In PP method an attempt to replace the complicated effect of core electrons on the atomic potential, so an effective potential is used to fit the experimental data about the material, in many cases many forms of potential can be used, for each form of the material different potential can be used to give the experimental data.

In Tight binding method, the value of the interaction between the valence electrons is replaced by a numeric value, the value of this number is predicted from already known experimental data, as in the previous method(pseudo potential). The value of the same interaction is different from form to form for the same material.

Until the development of theoretical schemes like Density Functional Theory (DFT) [15] by Hohenberg and Kohn, with the help of the fast and rapid cheap computers has began to change the situation. Another name for such calculations is called ab-initio calculation; the

only input for such calculation is only the basic information like the name of the atoms and form of the material.

A lot of packages nowadays using DFT such as VASP [16], Wien2k[9], Gaussian[17],… etc. In these studies we have two factors controlling the calculation:

1- The sample actuality.
2- The time of calculation.

The sample actuality means here the number of atoms constituting the sample, the bigger the number of atoms in the study case the more actual the case we have, this will cost a lot of calculation time, so the relation between the two factors are vice versa. In the present work, Wien2k program is studied, we will focus on the ordered structure of atoms which are named in solid state physics "Crystal" as shown in Fig. (1.2) and Fig.(1.3), the crystal is composed of a definite number of atoms has a definite position in space, the rest of the crystal is an empty space, the space between the atoms in the crystal called Interstitial region.

Experiments have proved that the outer shell electrons of the atoms are responsible to define the physical and chemical properties of the atoms and its compounds [18]. The net interactions between the repulsive and attractive forces between the different atoms electrons and their nuclei decide which phase these atoms will take. Each atom composed of a big number of electrons, each electron is interacting with all the other electrons and with each positive nuclei. These interactions can only be treated and analyzed using quantum mechanics treatment. This many body problems can only be solved by making

use of the translational symmetry, which cause the electronic wave functions to be of Bloch-type, labeled by a k-vector in reciprocal space and the quantum number of the electron. Thus the periodicity in real space is defined by the k-vector in reciprocal space, whose unit cell is called Brillouin Zone (BZ). The latter becomes smaller the larger the real space unit cell gets[19]. The interaction between the electrons and nucleus can be presented through the one electron Schrödinger Equation [20]:

$$H_{KS}\,\Psi = E\,\Psi \tag{2.2}$$

$$(-\nabla^2 + \mathrm{V_{xc}})\Psi = \mathrm{E}\,\Psi \tag{2.3}$$

Where, ∇^2 is the second derivative with respect to space coordinates, and Vxc is the effective attractive potential each electron feel, E is the energy of this electron in this crystal phase, and Ψ is the wave function of this electron. And this Ψ when squared and summed over all the crystal space we get density function of this electron as a function of position:

$$\rho(r) = \int \Psi\Psi^*\ dr^3 \tag{2.4}$$

Adding this density function for all the electrons, the answer of this sum are logically equal the total number of electrons in the interaction. The problem is that we don't know the actual V_{xc} and Ψ, this problem treated in DFT by giving initial wave function Ψ , this

wave function is very close to atomic wave function, later we solve the Schrödinger equation and finding the V_{xc} from the equation :

$$\nabla^2 V_{xc} = \rho(r) \tag{2.5}$$

This new V_{xc} is again entered to the Schrödinger equation and again we solve for the new Ψ and so on, this cycle is kept repeated until the total energy reaches a minimum value, this minimum energy value is chosen at the begging of the calculation, it should be suitable and comparable to the size of the problem, the value of this energy is directly related to the time of calculation through the number of cycles needed, at this optimum energy the wave function Ψ and exchange correlation potential V_{xc} is the best representative for all the electrons see Fig.(1.4),

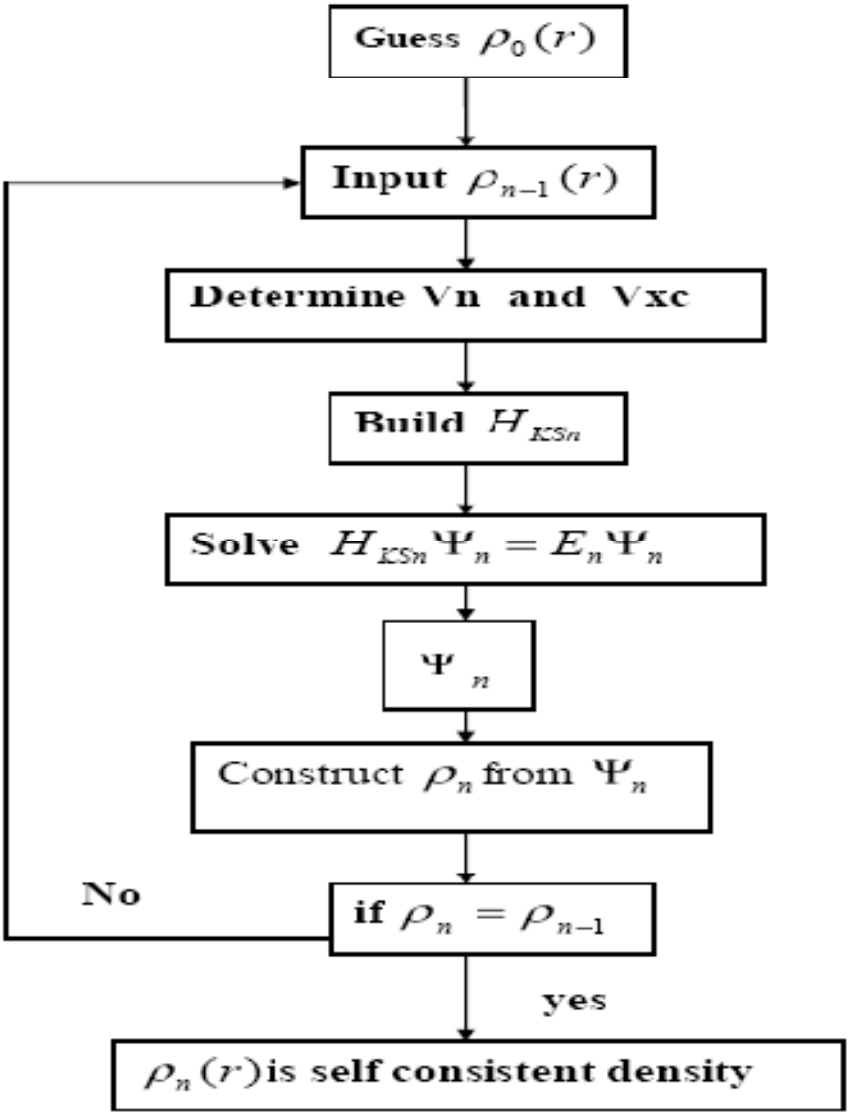


Figure 1.4 physical problem solving steps

These sequential operations divided into five main modules, the first module is called in the program LAPW0, in this process the V_{xc} is calculated in the crystal from the initial . The second module called LAPW1 is responsible for building the Schrödinger equation (setting up H and S matrix) and solves the generalized Eigen value problem, for a special point in the BZ, these points are called K points, the number of these points is proportional

to the reality of the study, the high number give more accurate results and coasts a lot of computational time, so balance should be done. After solving the Eigen value problem, the Eigen vectors ΨI is calculated for each Eigen value and the new density is calculated according to Equation 2.4, this process called LAPW2, from this density function the electrons in the crystal is distributed on the lowest energy values, the density function for the core electrons also calculated and in LCORE step, the new total density is compared with the old density, if the values is the same the self consistent (SC) is finished, the total energy and wave functions of the electrons is found, otherwise the new density is mixed with the old density with a percentage decided at the beginning of the calculation to reproduce a new density to run another cycle, this process is MIXER. The cycle is repeated until we get a difference between the total energy and the new total energy less that a value already decided it. The weighted time needed for each step in a single processor calculation is shown in Fig.(1.5)

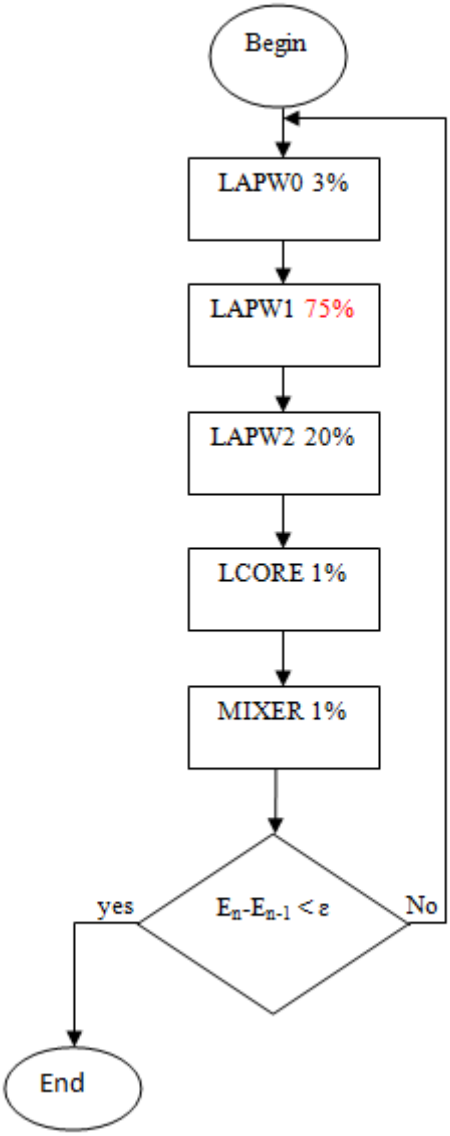


Figure 1.5 weighted time for each module which was calculated from the Dayfile of Wien2k package

2. Chapter Three:

Parallel Computation Background

Inventing a super computer is a very old dream, this dream begin to become reality for the first time in 1953, IBM company invented a computer with power as 100 PC's using 8085 microprocessor**,** the aim of this super computer is to perform many calculations simultaneously assuming that the problem can be divided into a smaller ones**,** which can be solved concurrently in parallel.

There are many architectures for parallel computing, one of these architecture is proposed by Michael J. Flynn in 1966[21] which known as Flynn's taxonomy, it is classified the architecture on the presence of single or multiple streams of instructions or data. This yields to these taxonomy:

1. Single instruction single data (SISD): one instruction will be executed on one processor and this defines the serial computers.
2. Multiple instruction single data (MISD) multiple processors will execute multiple instruction on a single data
3. Single instruction multiple data (SIMD): multiple processors will execute the same instructions on different data
4. Multiple instruction multiple data (MIMD) : multiple processors will execute multiple instruction on multiple data

Parallel computing takes different forms according to the working style, these forms are**:**

1-Bit level: it is based on increasing processor word size, this will reduce the number of instructions the processor must execute in order to perform an operation on variables whose sizes are greater than the length of the word. [22,23,24].

2-Instruction-level parallelism (ILP)[25]: is a measure of how many of the operations in a computer program can be performed simultaneously, only independent operations are distributed where the input of it is available, this is very applicable in scientific and graphical programs.

3-Data parallelism [26]: in this form of parallelization data is distributed on multiple processors environment, in a multiprocessor system executing a single set of instructions (SIMD), data parallelism is achieved when each processor performs the same task on different pieces of distributed data. In some situations, a single execution thread controls operations on all pieces of data. In others, different threads control the operation, but they execute the same code.

In the present work, we will concentrate on the last form of parallelism, data parallelism which emphasizes the distributed (parallelized) nature of the data, as opposed to the processing (task parallelism). Most real programs fall somewhere on a continuum between task parallelism and data parallelism. In this form of parallelism all CPUs will execute the

code and having access to the data at the same time. Obviously, this will be faster than doing it on a single CPU. This concept can be generalized to any number of processors. However, when the number of processors increases.

Parallel computer programs are more difficult to write than sequential ones, because concurrency introduces several new classes of potential software bugs, of which race conditions are the most common. Communication and synchronization between the different subtasks are typically one of the greatest obstacles to getting good parallel program performance.

One of the method to show the performance of the parallel algorithm is to calculate the ratio between the serial and the parallelism of the program which called speed up. The maximum possible speed-up of a program as a result of parallelization is observed as Amdahl's law. Originally formulated by Gene Amdahl in the 1960s.[27] It states that a small portion of the program which cannot be parallelized will limit the overall speed-up available from parallelization. A program solving a large mathematical or engineering problem will typically consist of several parallelizable parts and several sequential parts. If α is the fraction of running time a sequential program spends on non-parallelizable parts, then

$$S \leq \frac{1}{\alpha} \tag{3.1}$$

is the maximum speed-up with parallelization of the program. If the sequential portion of a program accounts for 10% of the runtime, we can get no more than a 10× speed-up, regardless of how many processors are added. This puts an upper limit on the usefulness of adding more parallel execution units.

2.1 Memory Architecture

Main memory in a parallel computer is either shared memory Fig.(2.1)(shared between all processing elements in a single address space), or distributed memory Fig.(2.2) (in which each processing element has its own local address space).

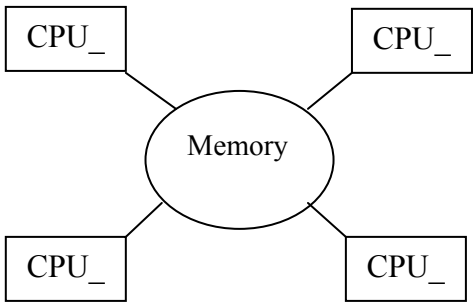


Figure 2.1 shared memory architecture [28]

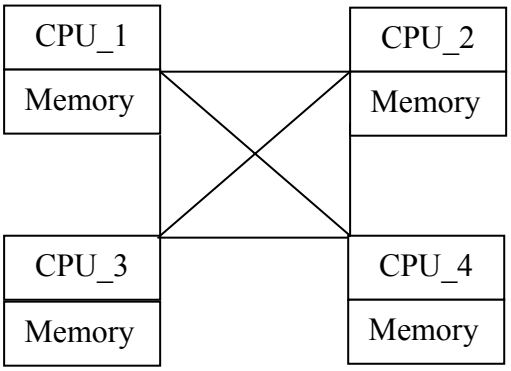


Figure 2.2 distributed memory architecture [28]

Distributed memory refers to the fact that the memory is logically distributed, but often implies that it is physically distributed as well. Distributed shared memory and memory virtualization combine the two approaches, where the processing element has its own local memory and access to the memory on non-local processors. Accesses to local memory are typically faster than accesses to non-local memory.

Computer architectures in which each element of main memory can be accessed with equal latency and bandwidth are known as Uniform Memory Access (UMA) systems. Typically, that can be achieved only by a shared memory system, in which the memory is not physically distributed. A system that does not have this property is known as a Non-Uniform Memory Access (NUMA) architecture. Distributed memory systems have non-uniform memory access [28]

2.2 Classes of parallel computers

Parallel computers can be roughly classified according to the level at which the hardware supports parallelism. This classification is broadly analogous to the distance between basic computing nodes. These are not mutually exclusive; for example, clusters of symmetric multiprocessors are relatively common.

Some of these classes are :

1- Multicore computing.

2- Symmetric multiprocessing.

3- Distributed computing.

3. Chapter Four

Parallel Algorithms

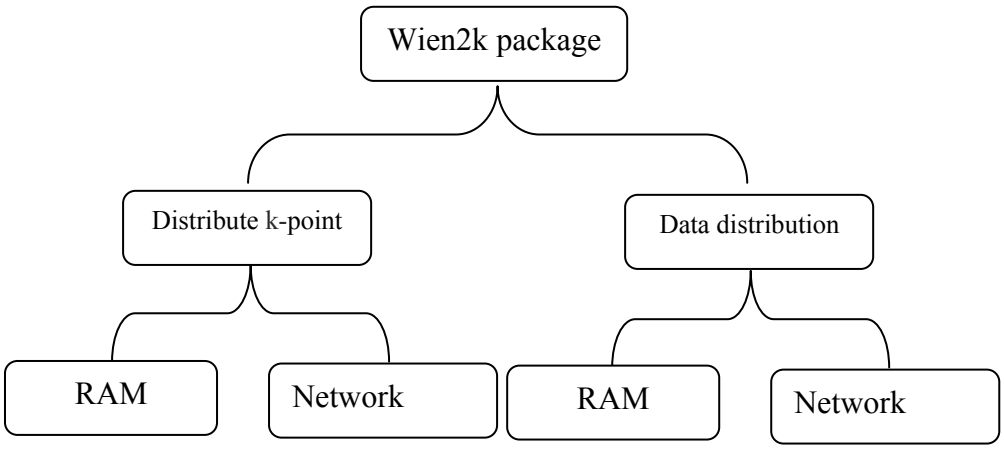


Figure 3.1 possible running for Wien2k package

In the present work, two cases have been tested, in each case different crystal has been used, each crystal has different number of atoms, the number of atoms is regarded as a benchmark for the calculation, its directly related to the difficulty of the calculations, for the first case 48 atoms per unite cell were used, this is equivalent to solving an Eigen value problem of 2880 X2880 matrix in LAPW1 module, this matrix size came from; each atom has more than 30 electrons and each electron wave function has two possibilities (spin up, spin down) and each electron is interacting with all the other electrons and this yield a matrix of this size (48*30*2=2880) , the second case is composed of 64 atom per unit cell, in a computational language its solving a matrix of size 3840 X 3840. One of the computers has been used as a server, the server has a definite IP and a name assigned with this IP, rezekq24t machine has been used as server and the other two machines as nodes, their names are rezekq24 and rezek3k, the server and each node has the same programs installed with the same names assigned to the directories, the directory where we do

initialization, calculation and saving output files of the study is named the **/common** directory with read write attributes, the same directly name also created on each node, after the initialization process on the server we mount the server **/common** directory on each node, with this step each node is doing the calculation using its CPU's and writing the output in this **/common** directory which is originally on the server, for that reason network speed may affect the calculation time, the file which is responsible to define the number of CPUs and which CPUs to perform each module is called machines file, so the number of CPU's and which machines will be used are defined at the beginning of calculation, according to the form of this machines file the package will decide which parallel method is executed.

In the present work, two styles of machines file has been used, one for the distribute k-point method and one for data distribution method. Both of these method are based on MPI using OpenMPI API

a- The following file is a sample of machines file used in distribute k-point method;

```
#machines file for Distribute k-point method
lapw0:rezekq24t:2
1:rezekq24t
1:rezekq24t
1:rezekq24
1:rezekq24
1:rezek3k
1:rezek3k
```

Figure 3.2 Sample of machine file shows CPUs distribution for modules

It declares that module LAPW0 will be executed on child0 and child1 on rezekq24t, while module LAPW1 will be divided into 6 CPUs, on child0, child1 on rezekq24t, child0, child1 on rezeq24 and child0, child1 on rezek3k. Using this form of machines file will also divide LAPW2 by the same way as in LAPW1, each CPU on each node will write the output of the calculation in a file named according to the order of the machine name in the machine file followed with underscore and a number of the order of the child in that machine, for the previous example we will have casename-vector1_1,.., casename-vector2_1, ..casename-victor3_2. So the first file casename-vector1_1 comes from the CPU named child0 on machine rezekq24t, and the second file casename-vector2_1 comes out from child0 on machine rezekq24 and the file casename-victor3_2comes out from child1 on machine rezek3k.

3.1 The paralleled modules executed in Distribute k-point method:

1- LAPW0 parallelism: this module is responsible for solving Equation (2.5), this is a three dimensional second order deferential equation, the domain of the equation is the volume of the unit cell, the way how LAPW0 executed is defined in the file named machines file, LAPW0 can be executed on one CPU or two … as many as we have, In this K-point method LAPW0 only solved by one CPU. The input for this module is the physical dimension of the crystal and the angles, the module depends on FFTW package to do parallelism.

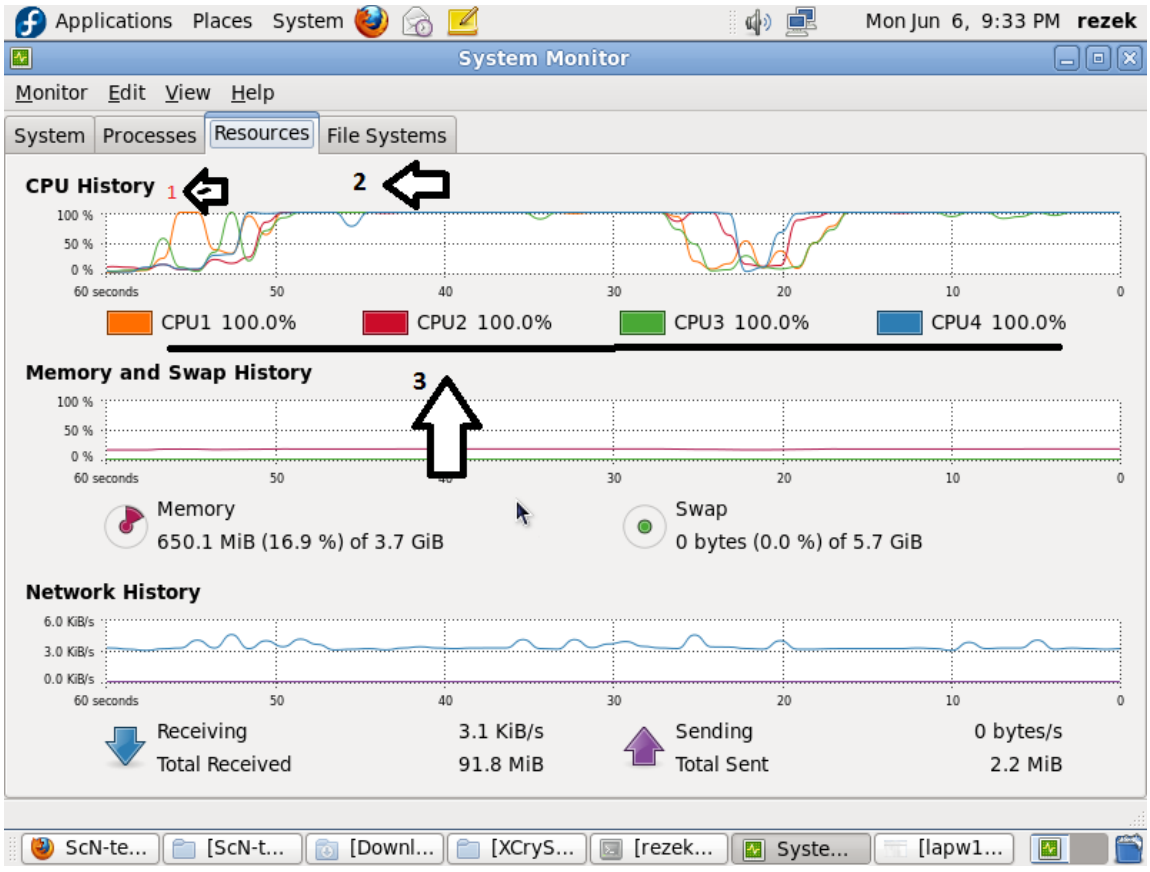


Figure 3.3 system monitor during Distribute k-point method

The black arrow next to number 1 in Fig.(3.3) directed at interval where lapw0 executed, from Fig.(3.3) lapw0 has been executed on one CPU, also initialization for the next step which is lapw1 doesn't happen simultaneously, that's shown from arrow two in Fig(3.3) not all the CPUs begin at the same time.

2- Lapw1 and Lapw2 parallelism: in these modules the number of CPUs and the number of k-points is known from the initialization, the total number of K points should be an inter multiplier to the number of CPUs used in the calculation, otherwise there will be unbalance distribution of tasks which resulted in a waste of time of running for the CPUs. Fig.(3.4) shows how Lapw1 and Lapw2 distributed on machines,

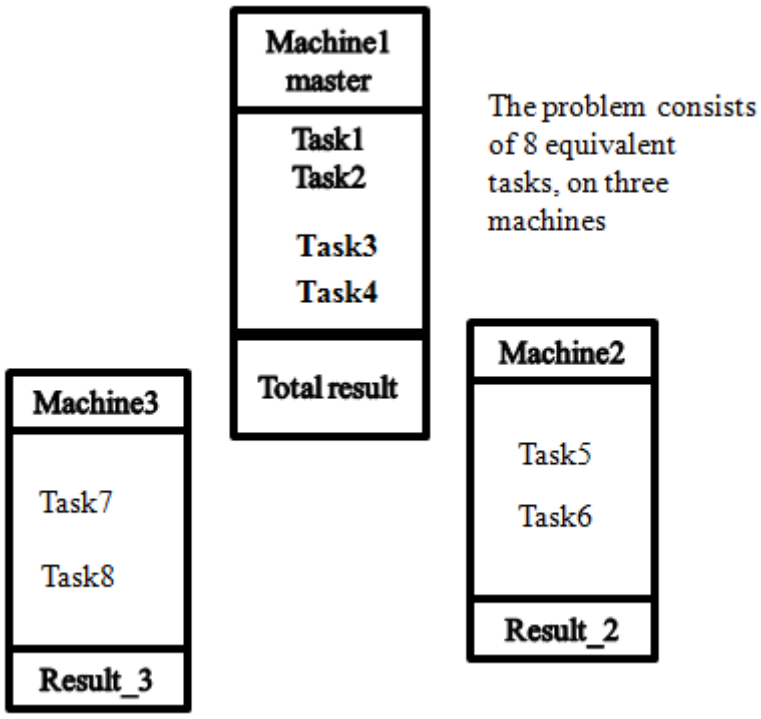


Figure 3.4 sample distribution of tasks in Distribute k-point method

For each K point different matrix's are build, so each CPU will solve LAPW1 and LAPW2 for 1, 2, ..M times, so that

$$M \times NPE = \text{Number of K points} \tag{4.1}$$

Where, M is the number of times that each CPU will solve cycle, NPE is the total number of CPUs in the machines file, From Fig.(3.4) each CPU solves the Eigen value problem for each K value independently from the other CPUs, this will minimize the effect the network overhead, in addition will use the whole computational power of the CPUs.

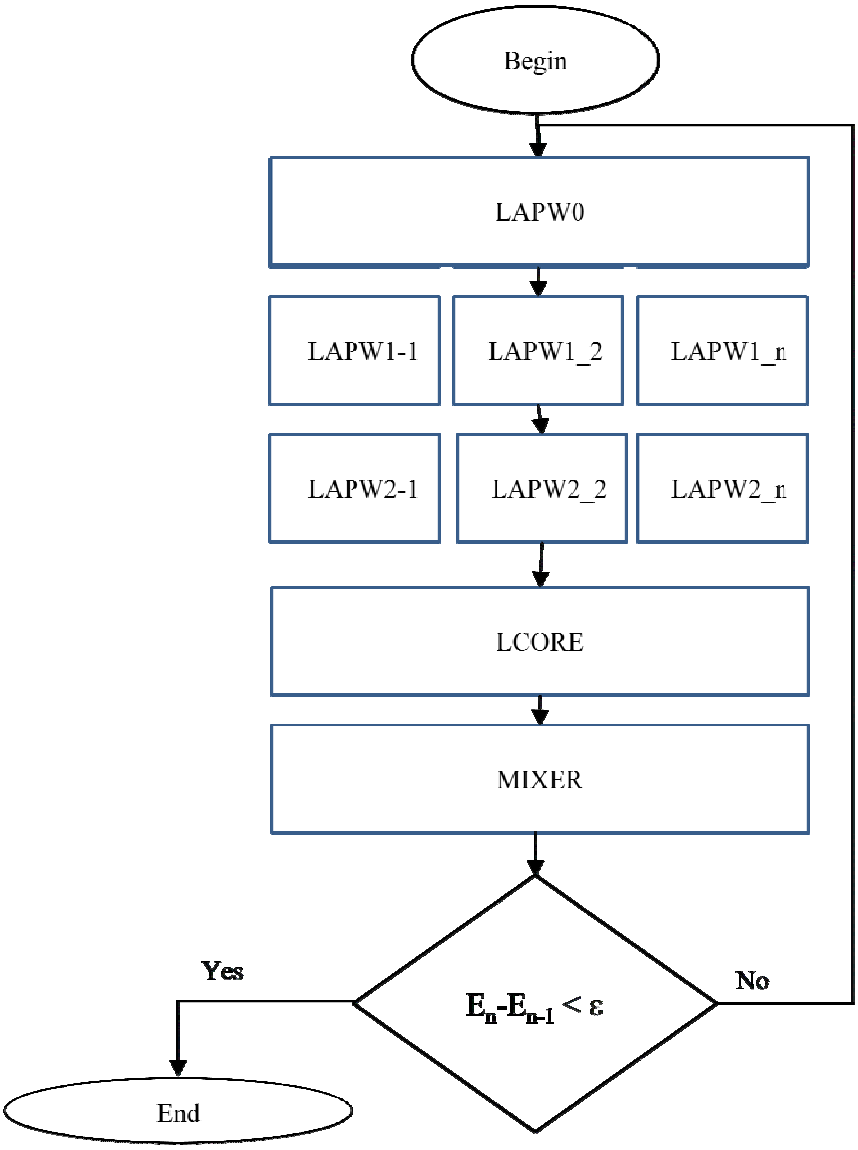


Figure 3.5 parallelism level in Distribute k-point method

In this Distribute k-point method, as expressed also in Fig. (3.5) the output files of parallelism of Lapw1 and Lapw2 is written into files as stated before, a program doing a combination of the output from all the CPUs into a new single file, this is done because the rest of the modules (LCORE and MIXER) is carried on a single CPU.

As a result, the package program will divide the LAPW1 and LAPW2 modules equally according to the number of CPU's, keeping in mind LAPW1 and LAPW2 modules consumed 95% of the calculation time.

3.2 The parallelized modules executed in Data distribution method:

For the Data distribution method, the distribution of modules is different, for each k point the same task distributed on the CPUs available in the machines file, each module is divided separately as shown in Fig.(3.6), the machine one represent the server, the other two machine are the nodes used, the server send the sub-task to the CPUs on machine 2 and machine 3.

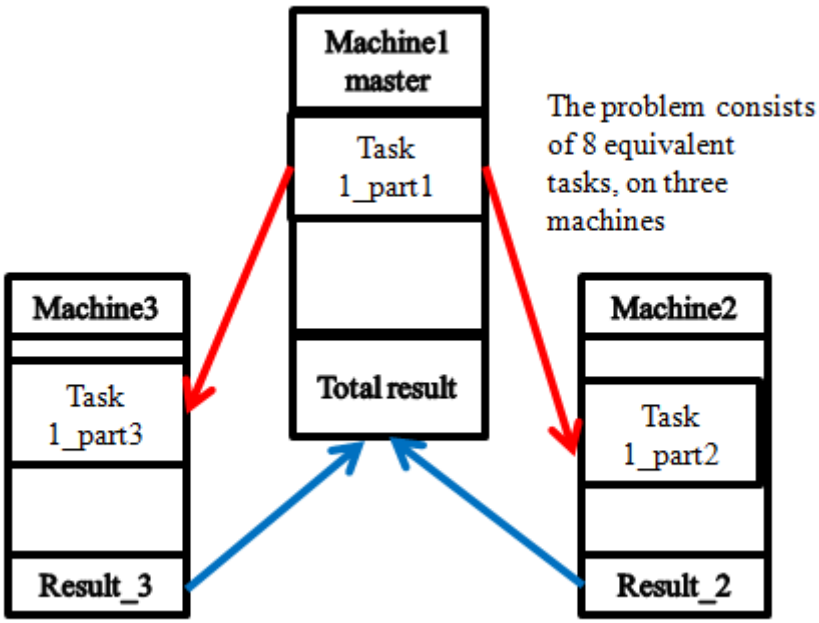


Figure 3.6 schematic distribution of tasks in Data distribution method

And the way of division is declared in the machines file as follow:

```
#machines file for Data distribution method
lapw0:rezekq24t:2
1:rezekq24t:3 rezekq24:2
Lapw2_vector_split:4
```

Figure 3.7: sample file for CPUs distribution for the modules

1- LAPW0 parallelism: From Fig.(3.7) LAPW0 will be executed on two CPUs, as a result the unit cell volume is split into two identical volumes, this is done by dividing the length sides of the unit cell and the angles between sides of the unit cell between the two CPUs, each CPU solves Equation (2.5) using the same program (FFTW), the output of this process is to find ρ(r) for each sub volume at a lot of points, after the two CPUs finish the calculated ρ(r) for the hole volume is found, a sub program will combine the two solutions called SUMPARA.

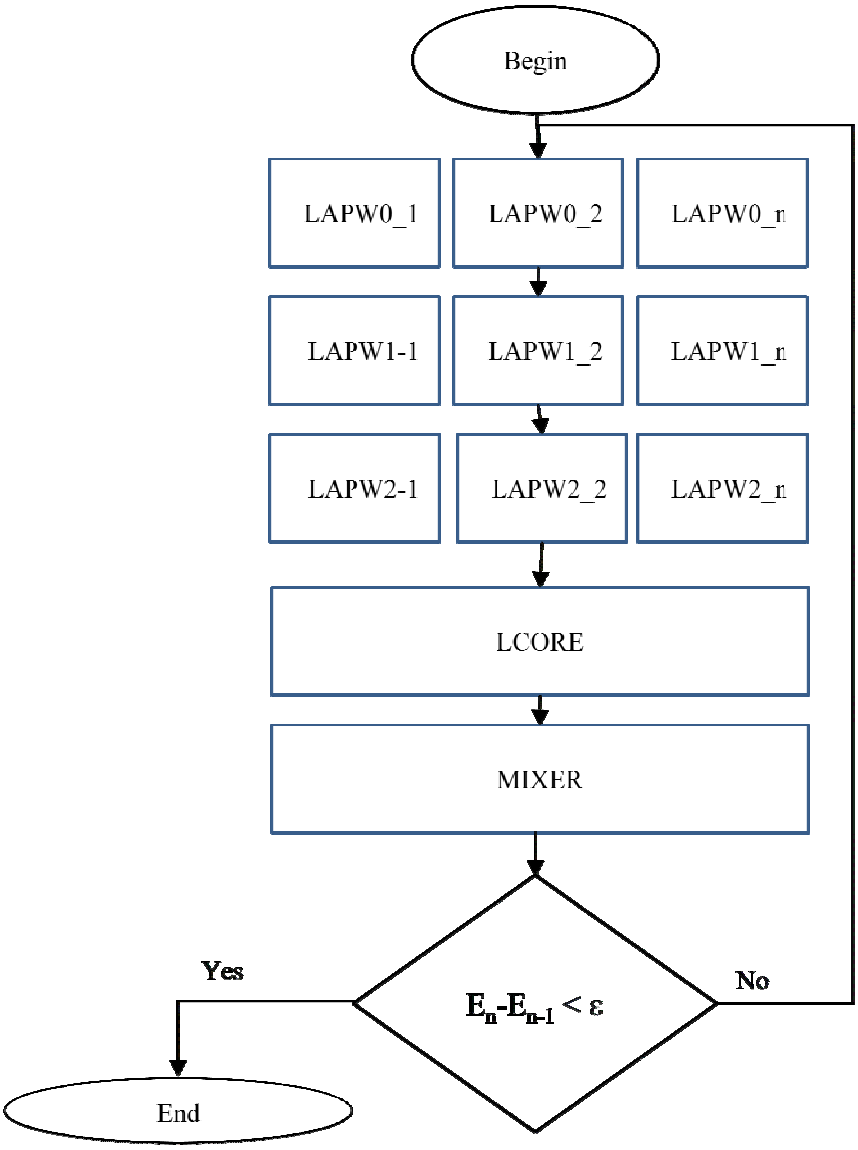


Figure 3.8 parallelism schematic in Data distribution method

2- LAPW1 parallelism: from the file in Fig.(3.7) LAPW1 will be solved by 5 CPUs, three from machine rezekq24t and two from rezekq24, so the Number of processors (NPE) =5, the Eigen value problem dimension will be divided into a smaller one's, this done by dividing the big matrix H into a sub matrixes (sparse matrixes) , the dimension of these sub matrixes is calculated as follows:

$$NPCOL=SQRT(real(NPE)) \quad (4.2)$$

$$NPROW=NPE/NPCOL \quad (4.3)$$

Where NPCOL is the number of columns in the sub matrixes, NPROW is the number of rows in the sub matrixes, for our case here it will be 2X2 size. Each CPU is solving for this small matrix and returns the result, by calling the Scalapack libraries.

3.3 Network effect

In order to study the network effect on the calculation time, different cases has been executed and the time of running has been recorded, the data for the calculation time has been taken from a file named DAYFILE, as shown in Fig. (3.9), at the beginning of the file it states that this is the calculations carried in cycle 2, the DAYFILE contains all the cycles time, the first module registered is lapw0 with the options used, the numbers in bracket in red indicates the starting time for lapw0, the second number in bracket also in red indicates the starting time for the second module, the time difference between the two times represent the time of execution in addition to the time of network communication, the time of execution is under the sentence '**.machine0 : 4 processors'** which states that this module executed using 4 CPUs, the time of executing this module is 1:50.06 which means 1 minute and 50 seconds and 0.06 of a second. While the difference between the two starting is (13:27:47 – 13:25:57) = 00:01:50 which indicates that the effect network in this case is zero, the same procedure has been perform for the other modules. The file also shows the calculation time for every CPU and we can see in module lapw1 that 4 tasks has been distributed and the time of calculation for one of the CPUs which is in bold is 38(min):42(sec).00(parts of second)

```
cycle 2      (Mon Nov 15 13:25:57 IST 2010)     (95/98 to go

>   lapw0 -grr -p  (13:25:57) starting parallel lapw0 at Mon Nov
15 13:25:57 IST 2010
-------- .machine0 : 4 processors
404.104u 21.281s 1:50.06 386.4%    0+0k 0+812136io 12pf+0w
>   lapw0 -p      (13:27:47) starting parallel lapw0 at Mon Nov 15
13:27:48 IST 2010
-------- .machine0 : 4 processors
531.029u 21.853s 2:21.50 390.7%    0+0k 0+812352io 9pf+0w
>    lapw1  -c -up -p        (13:30:09) starting parallel lapw1
at Mon Nov 15 13:30:09 IST 2010
->  starting parallel LAPW1 jobs at Mon Nov 15 13:30:09 IST 2010
running LAPW1 in parallel mode (using .machines)
4 number_of_parallel_jobs
     rezekq24t(2) 2233.674u 7.062s 37:34.76 99.38%      0+0k 0+0io
0pf+0w
     rezekq24t(2) 2301.546u 5.633s 38:42.00 99.36%      0+0k
0+0io
     rezekq24t(2) 2276.276u 5.463s 38:17.05 99.33%      0+0k 0+0io
0pf+0w
     rezekq24t(2) 2246.514u 5.547s 37:45.02 99.43%      0+0k 0+0io
0pf+0w
   Summary of lapw1para:
   rezekq24t       k=8   user=9058.01     wallclock=9138.83
2.584u 4.215s 38:44.41 0.2%  0+0k 240+856io 6pf+0w
>   lapw1  -c -dn -p         (14:08:54) starting parallel lapw1 at Mon
Nov 15 14:08:54 IST 2010
->  starting parallel LAPW1 jobs at Mon Nov 15 14:08:54 IST 2010
running LAPW1 in parallel mode (using .machines.help)
4 number_of_parallel_jobs
     rezekq24t(2) 2247.405u 7.085s 37:45.38 99.52%      0+0k 0+0io
0pf+0w
     rezekq24t(2) 2253.862u 5.206s 37:52.18 99.42%      0+0k 0+0io
0pf+0w
     rezekq24t(2) 2164.932u 5.278s 36:22.94 99.42%      0+0k 0+0io
0pf+0w
     rezekq24t(2) 2216.934u 5.235s 37:14.25 99.46%      0+0k 0+0io
0pf+0w
   Summary of lapw1para:
   rezekq24t       k=8   user=8883.13     wallclock=8954.75
2.517u 4.152s 37:54.60 0.2%  0+0k 976+824io 17pf+0w
>   lapw2 -c -up  -p -vresp  (14:46:48) running LAPW2 in parallel mode
     rezekq24t 82.335u 10.841s 1:33.88 99.24% 0+0k 0+0io 0pf+0w
     rezekq24t 83.680u 10.526s 1:34.65 99.53% 0+0k 0+0io 0pf+0w
     rezekq24t 82.234u 10.552s 1:32.84 99.94% 0+0k 0+0io 0pf+0w
     rezekq24t 82.891u 10.590s 1:33.63 99.84% 0+0k 0+0io 0pf+0w
   Summary of lapw2para:
   rezekq24t       user=331.14      wallclock=375
13.914u 6.777s 1:58.38 17.4% 0+0k 584+249576io 3pf+0w
>   lapw2 -c -dn  -p -vresp  (14:48:47) running LAPW2 in parallel mode
     rezekq24t 84.070u 10.837s 1:35.90 98.96% 0+0k 0+0io 0pf+0w
     rezekq24t 83.516u 10.869s 1:34.74 99.62% 0+0k 0+0io 0pf+0w
     rezekq24t 83.149u 10.666s 1:34.83 98.93% 0+0k 0+0io 0pf+0w
     rezekq24t 83.396u 10.458s 1:34.75 99.05% 0+0k 0+0io 0pf+0w
   Summary of lapw2para:
   rezekq24t       user=334.131     wallclock=380.22
```

Figure 3.9 a sample of DAYFILE displays registration of operations time

3.4 RAM Effect

The effect of RAM has been measured for different cases, different modules and machines, two sizes of RAM has been used, they are 1GB and 2 GB per CPU,

The amount of RAM used for each module has been followed by a top command in linux, Fig.(3.10) shows a sample print of this screen. The first underlined numbers in the upper row indicates the total RAM available, consumed RAM in calculating this module and the unused RAM in the calculation of this module, the module name is in the table below near to the arrow, in this figure the module is LAPW1, we can see here four tasks on the four CPUs on this machine, the numbers before the module name is the percentage usage of the CPU capacity, in this case it shows 85.7% for the first CPU.

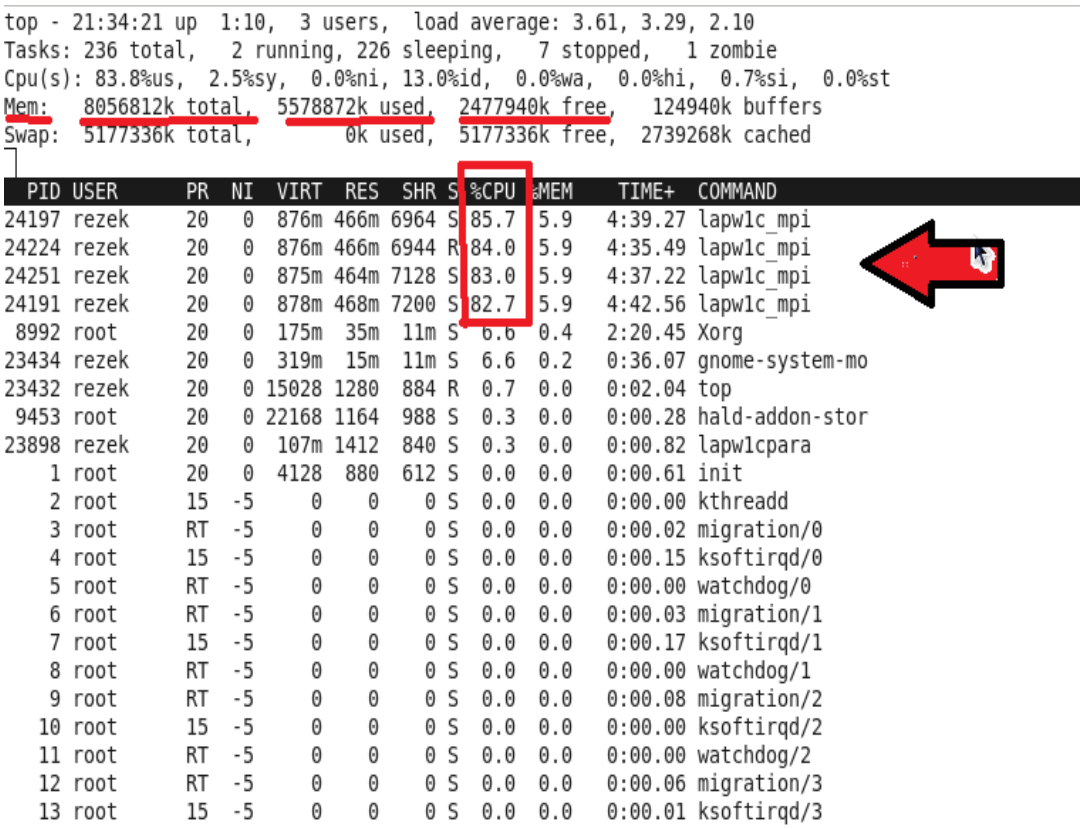


Figure 3.10 : a sample of screen print for a top command

4. Chapter Five

Experiments Design

In the present work, three machines were used, the properties of these machines are mentioned in Table (4.1), as can been seen two of these machines are identical and the third one is different. The machines are connected by a switch of speed 100MHz, with special subnet mask (255.255.0.0) to assure fast connection.

Table 4.1 Machine specifications

	rezekq24t	rezekq24	rezek3k
CPU speed	Quad 2.4 GHz	Quad 2.4 GHz	Dual core 3KHz
RAM size	4GB/8GB	4GB/8GB	4GB
Network speed	1Gbps	1Gbps	1Gbps
Cashe	8Mbyte	8Mbyte	8Mbyte
HD speed	7200 RPM	7200 RPM	7200 RPM
IP address	10.96.190.40	10.96.190.41	10.96.190.42
Network	255.255.0.0	255.255.0.0	255.255.0.0

In the first run each machine is used with 4GB of RAM. In the second run, the first two machines (rezekq24, rezekq24t) is used with 8GB of RAM.

To accomplish the calculations a set of programs were installed and optimized with the suitable options together with Wien2k. These programs are listed in Table(4.2), with the version of each program and the source of the program , the operating system used is Fedora Linux version 10, first we install the program secure shell (SSH) from open SSH website (www.openssh.org), after installing this program in a common directory, then we set the options so that we can access each machine from any of the other two machines, in order to achieve this goal we install the program as a root and we choose password less option, we used SSH-version 5-p5, the program SSH guaranty that only the authorized user can read and write to any specific directory, the id public keys generated on each machine is copied to the authorized-keys file of the other machines, each machine name is copied to the file known host of the other two machine.

Table 4.2 Software requirements

program name	version	source
wien2k	10.1	www.wien2k.at
OpenMPI	openmpi_1.42	www.open-mpi.org
open ssh	5.5p1	www.openssh.org
Intel Fortran 90 compiler	11.072	Intel
Intel c compiler	10.074	Intel
mathematical kernel library (MKL) including Scalapack	10.072	Intel
Fastest Fourier Transform in the west (FFTW)	FFTW-2.1.5	Intel

The program Wien2k is written in Fortran 90 and C shell , for that purpose we have installed the newest Intel Fortran and C++ compilers for Linux operating systems with mathematical kernel library, these compilers also needed to compile and used by OpenMPI and FFTW programs.

Both OpenMPI and FFTW are used for parallel MPI running, these programs are installed on the same directory name on each machine. In the installation process the needed compilers are defined in the installation options, so that MPI can use them to compile the Wien2k program.

In the last step we install Wien2k package, Once the package installed there is no need to keep the compilers active and can be called for any user, the package is installed and extracted in the same directory name on each machine, the package asked for the two main directories, the first where cases are initiated for running, the other where temporary files of running are kept, for example /common directory for cases, and /common/work as temporary directory, these directories should be the same on each machine, these directories attributes are adjusted to be read and write, also these directories should be mountable from each machine.

The package asks for MPI Fortran 90 compiler and MPI C compiler and their libraries and where installed together with the MKL and SCALAPACK.

5. Chapter Six:

Results and Discussion

In the present chapter we will discuss the effect of the three main factors which are mentioned in the introduction and affect the calculation time in the two methods of parallel calculations.

1- Network speed.
2- RAM available per CPU.
3- Speed of the CPU.

5.1 Network speed:

The network cards installed on each machines is 1Gbps, while the switch which is used to connect the machines together is only 100Mbps speed.

From Fig. (5.1) is a sample of using four CPU's together on the same machine (rezekq24t), the network switch in this case has not been used, The operation showed is LAPW1 which takes 75% of the cycle time, the first red line states that one of the CPU's which is performing two tasks out of the eight tasks, the time consumed to accomplish this task is different from one CPU to another, this difference is due to the time taken to distribute these tasks on each CPU, the other line in the same Fig.(5.1) shows the total time to accomplish the LAPW1 run which is 38(min):44.41(sec) and the time consumed from the last CPU is 38(min):42.00(sec), From Fig.(5.2) shows the running of LAPW1

but on the three machines together, the time consumed on each CPU depends also on the speed and memory available on each CPU, that's why on rezek3k we noticed its fast compared with the other two machines rezekq24t and rezekq24.

```
    cycle 2       (Mon Nov 15 13:25:57 IST 2010)      (95/98 to go)

>   lapw0 -grr -p (13:25:57) starting parallel lapw0 at Mon Nov 15 13:25:57
IST 2010
-------- .machine0 : 4 processors
404.104u 21.281s 1:50.06 386.4%     0+0k 0+812136io 12pf+0w
>   lapw0 -p      (13:27:47) starting parallel lapw0 at Mon Nov 15 13:27:48
IST 2010
-------- .machine0 : 4 processors
531.029u 21.853s 2:21.50 390.7%     0+0k 0+812352io 9pf+0w
>   lapw1  -c -up -p    (13:30:09) starting parallel lapw1 at Mon Nov 15
13:30:09 IST 2010
->  starting parallel LAPW1 jobs at Mon Nov 15 13:30:09 IST 2010
running LAPW1 in parallel mode (using .machines)
4 number_of_parallel_jobs
     rezekq24t(2) 2233.674u 7.062s 37:34.76 99.38%      0+0k 0+0io 0pf+0w
     rezekq24t(2) 2301.546u 5.633s 38:42.00 99.36%      0+0k 0+0io 0pf+0w
     rezekq24t(2) 2276.276u 5.463s 38:17.05 99.33%      0+0k 0+0io 0pf+0w
     rezekq24t(2) 2246.514u 5.547s 37:45.02 99.43%      0+0k 0+0io 0pf+0w
   Summary of lapw1para:
   rezekq24t       k=8   user=9058.01      wallclock=9138.83
2.584u 4.215s 38:44.41 0.2%   0+0k 240+856io 6pf+0w
>   lapw1  -c -dn -p    (14:08:54) starting parallel lapw1 at Mon Nov 15
14:08:54 IST 2010
->  starting parallel LAPW1 jobs at Mon Nov 15 14:08:54 IST 2010
running LAPW1 in parallel mode (using .machines.help)
4 number_of_parallel_jobs
     rezekq24t(2) 2247.405u 7.085s 37:45.38 99.52%      0+0k 0+0io 0pf+0w
     rezekq24t(2) 2253.862u 5.206s 37:52.18 99.42%      0+0k 0+0io 0pf+0w
     rezekq24t(2) 2164.932u 5.278s 36:22.94 99.42%      0+0k 0+0io 0pf+0w
     rezekq24t(2) 2216.934u 5.235s 37:14.25 99.46%      0+0k 0+0io 0pf+0w
```

Figure 5.1 effect of network speed

The time of finishing the process here is 1(hr):06(min):42.26(sec) and the time consumed by the latest CPU is 1(hr):06(min):35.92(sec),

which shows that the difference is 6 seconds by, comparing the result with the case of no network 2.41 second , we can conclude that the effect of network effect on the calculation

time can be neglected compared with Memory and CPU speed. This is can be easily seen if we divide this time over the total calculation time.

```
lapw1  -c -dn -p        (03:33:30) starting parallel lapw1 at Sat Sep  4
03:33:30 EEST 2010
->  starting parallel LAPW1 jobs at Sat Sep  4 03:33:30 EEST 2010
running LAPW1 in parallel mode (using .machines.help)
8 number_of_parallel_jobs
     rezekq24t(1) 1938.304u 25.341s 1:04:13.39 50.96%      0+0k 0+0io 0pf+0w
     rezekq24t(1) 1991.639u 19.942s 1:04:13.20 52.21%      0+0k 0+0io 0pf+0w
     rezekq24(1) 1931.133u 22.067s 1:06:35.92 48.88%      0+0k 0+0io 0pf+0w
     rezekq24(1) 1956.264u 22.023s 1:05:43.88 50.16%      0+0k 0+0io 0pf+0w
     rezekq24t(1) 1980.513u 18.575s 59:09.71 56.32%      0+0k 0+0io 0pf+0w
     rezekq24(1) 1947.177u 21.351s 1:46.26 53.99%      0+0k 0+0io 0pf+0w
     rezek3k(1)  1497.231u 5.857s 26:55.46 93.04%      0+0k 0+0io 0pf+0w
     rezek3k(1)  1475.953u 5.891s 26:36.96 92.79%      0+0k 0+0io 0pf+0w
   Summary of lapw1para:
   rezekq24t       k=3   user=5910.46      wallclock=3677.71
   rezekq24  k=3   user=5834.57      wallclock=237.26
   rezekq24t       k=3   user=5910.46      wallclock=3677.71
   rezekq24  k=3   user=5834.57      wallclock=237.26
   rezek3k   k=2   user=2973.18      wallclock=3212.42
   3.041u 3.186s 1:06:42.26 0.1%    0+0k 40176+1488io 564pf+0w
```

Figure 5.2 a sample of dayfile shows effect of network speed

5.2 RAM available per CPU

5.2.1 One Gigabyte per CPU

In order to study the RAM effect on the calculation time, different cases executed on the same machines with the same network switch, in the first setting 4GB is installed on machine rezekq24 and rezeq24t with 1GB per CPU and 3GB on rezek3k with 1.5GB on each CPU as shown in Fig. (5.3). which represent a print screen for a top command while module LAPW1 has been executed on 4 CPUs, the method used in this figure is Openmpi

.

```
top - 20:14:12 up 48 min,  3 users,  load average: 0.67, 0.25, 0.25
Tasks: 213 total,   3 running, 209 sleeping,   0 stopped,   1 zombie
Cpu(s): 16.5%us, 10.0%sy,  0.0%ni, 69.7%id,  0.8%wa,  0.0%hi,  3.0%si,  0.0%st
Mem:   3928080k total,  1491552k used,  2436528k free,    47556k buffers
Swap:  6029304k total,        0k used,  6029304k free,   955292k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
10769 rezek     20   0 99496  15m 7040 S 29.6  0.4   0:12.87 lapw1c_mpi
10823 rezek     20   0 99600  15m 7088 S 28.6  0.4   0:11.66 lapw1c_mpi
10796 rezek     20   0 98660  14m 6924 R 28.2  0.4   0:11.59 lapw1c_mpi
10763 rezek     20   0 99140  15m 7352 S 26.9  0.4   0:10.75 lapw1c_mpi
 8739 rezek     20   0  655m  74m  24m S  2.7  1.9   0:30.58 firefox
 1401 root      20   0  170m  33m  10m S  0.7  0.9   1:08.43 Xorg
 8758 rezek     20   0  303m  13m 9652 S  0.3  0.4   0:00.43 gnome-terminal
10882 root      20   0 15016 1224  864 R  0.3  0.0   0:00.07 top
    1 root      20   0  4144  868  600 S  0.0  0.0   0:00.51 init
    2 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      15  -5     0    0    0 S  0.0  0.0   0:00.02 ksoftirqd/0
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      RT  -5     0    0    0 S  0.0  0.0   0:00.02 migration/1
    7 root      15  -5     0    0    0 S  0.0  0.0   0:00.01 ksoftirqd/1
    8 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/1
    9 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/2
   10 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/2
   11 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/2
   12 root      RT  -5     0    0    0 S  0.0  0.0   0:00.02 migration/3
   13 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/3
   14 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/3
```

Figure 5.3 a print screen for a top command shows a machine with 4 GB RAM

In this test the cases carried from one CPU to 10 CPU's.

Table 5.1 Time of calculation for 48 atoms per unit cell with 1GB RAM in Distribute k-point

Number of CPU	CPU speed (GHz)	Cal. Time (Hr)
1	2.4	1439.6
2	4.8	763.2
3	7.2	577.8
4	9.6	496.8
6	15.6	283.4
1	3.0	817.0
2	6.0	461.9
4	10.8	340.5

Table 5.2 Time of calculation for 64 atoms per unit cell with 1GB RAM Distribute k-point

Number of CPU	CPU speed (GHz)	Cal. Time (Hr)
1	2.4	3081.7
2	4.8	1708.7
4	9.6	1304.0
4	9.6	975.4
8	20.4	448.6

As seen from the Table(5.1) and Table(5.2), the general trend that the time is exponentially decreased as the speed increased as shown in Fig.(5.4) and Fig.(5.5)

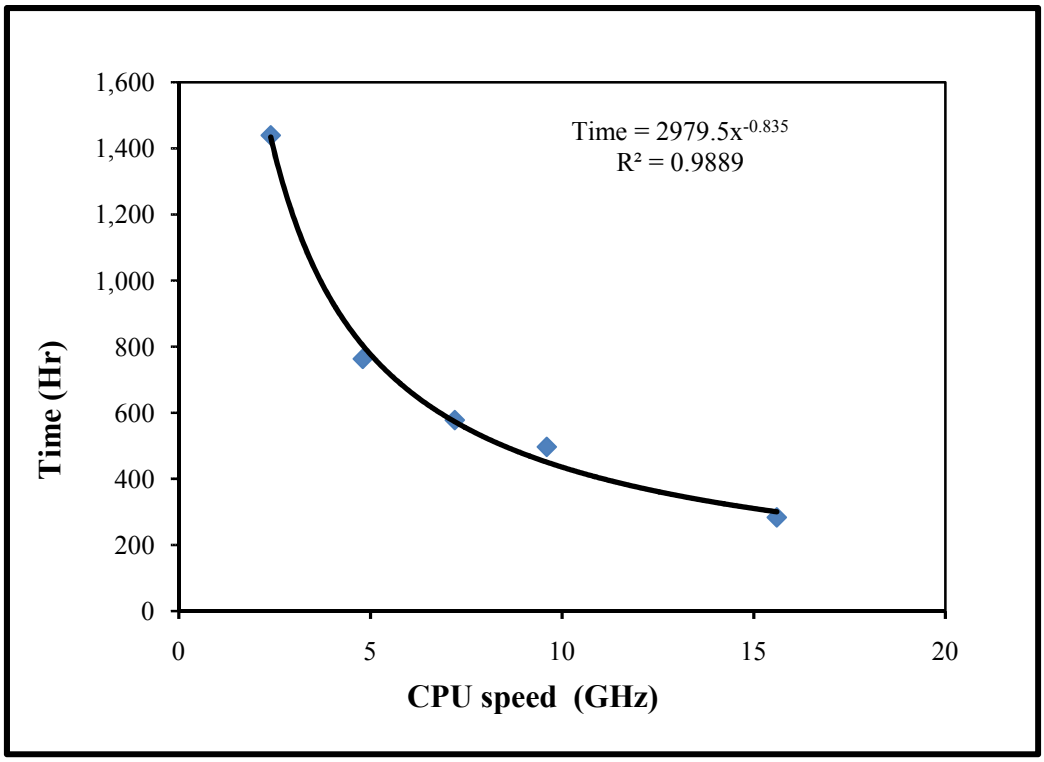


Figure 5.4 Time of calculation for 48 atoms per unit cell and 1GBe per CPU on Distribute k-point method

Shows that the time of calculation is exponentially fitted to the equation Time(x)=constant* X^(-power), this power for the 48 atoms and 64 atoms found 0.835and o.855 respectively and X represent the CPU speed in units of GHz. The high power indicates the fast decay of the calculation time, the constant multiplied with the CPU speed gives the starting time calculation for single CPU.

From the figures we notice that adding many CPU's to the calculation after a certain number does not reduce the time very much. Because we reach the saturation that means the parallelism is completely distributed and the serial part can't be distributed.

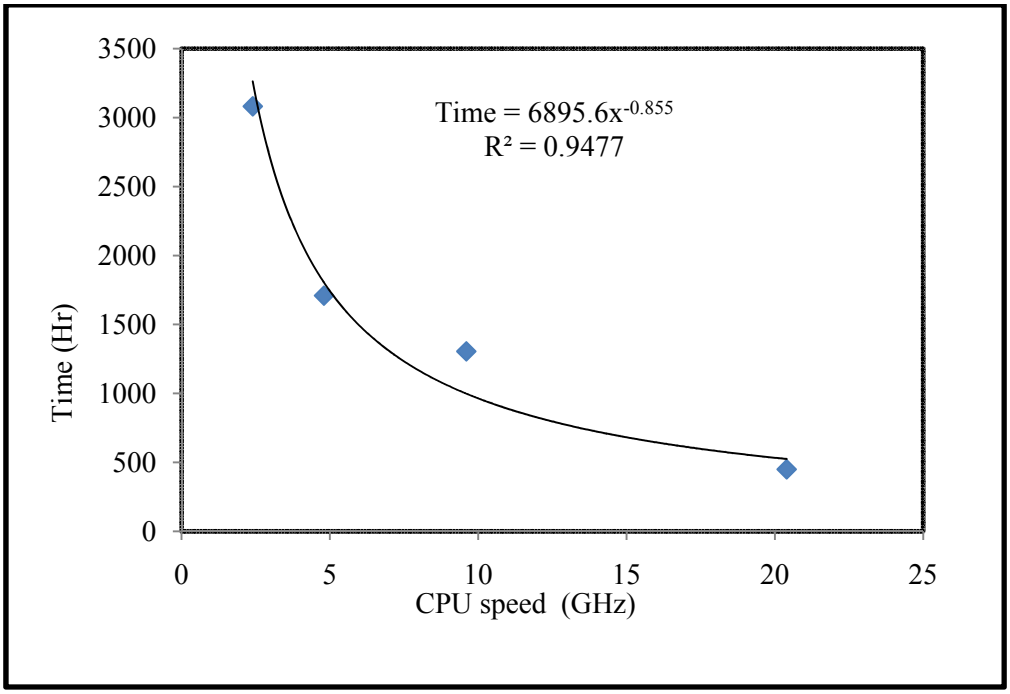


Figure 5.5 Time of calculation for 64 atoms per unit cell and 1GB per CPU on Distribute k-point method

From Table(5.1) and Table (5.2) if we look at the points where 4 CPU's is used, for the first point these four CPU's is used on one machine, and the time of calculation is 496.8 Hrs, while for the second point the four CPU's comes from two machines and keeping the other two CPU's on each machine unfunctional, this gave all the available RAM to the two operating CPU's, the time of calculation in this case found to be 340.5 Hr. This difference comes from the increase of RAM per CPU.

5.2.2 Two Gigabyte per CPU:

As continuation for the effect of RAM per CPU on the parallel computations either in Distribute k-point or Data distribution method, RAM per CPU has been increased to 2 GB per CPU, Fig.(5.6) shows a print screen for a top command while a module LAPW1 is executed using Distribute k-point method.

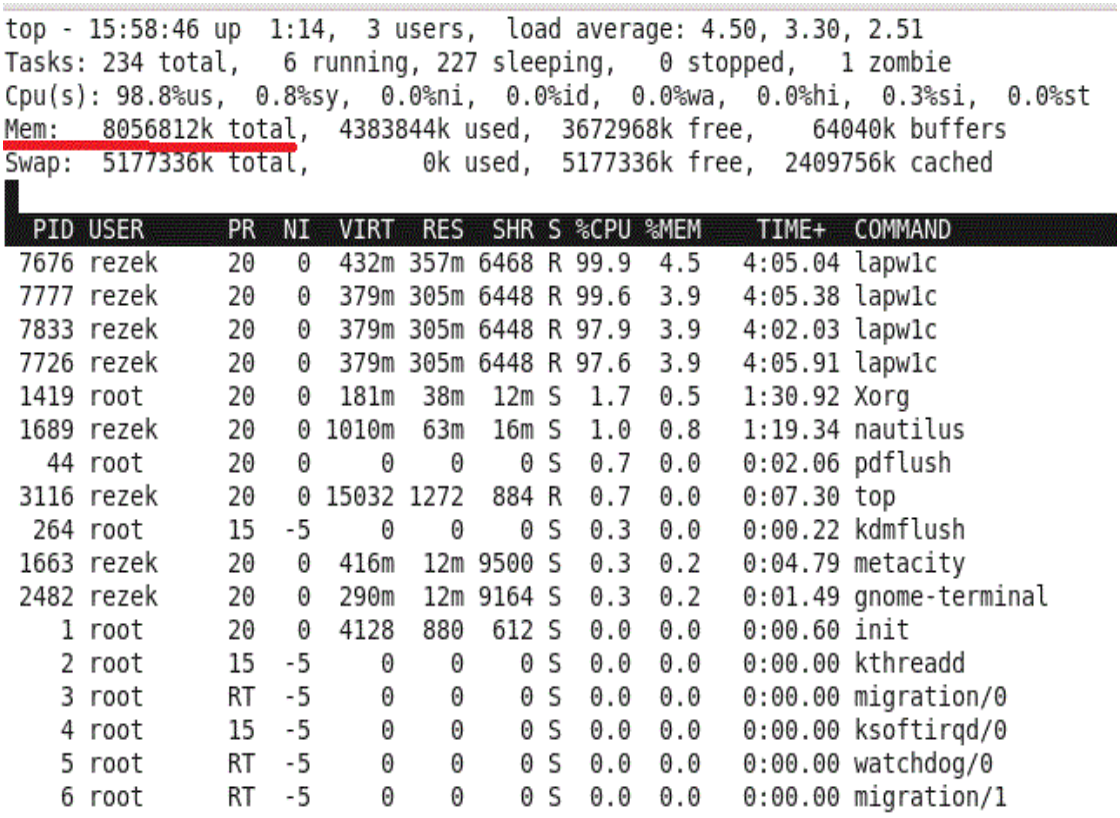


Figure 5.6 shows a print screen of a top command while lapw1 is running on 4 CPUs with RAM 8 GB

some cases has been carried in single and parallel mode until we reach saturation, the following two Tables (5.3), Table(5.4) show the time of calculation for 48 atoms and 64 atoms with the CPU speed and different number of CPU's using Distribute k-point method

.

Table 5.3 Time of calculation for 48 atoms per unit cell with 2GB RAM on Distribute k-point

Number of CPU	CPU speed (GHz)	Cal. Time (Hr)
1	2.4	473.229
2	4.8	260.257
3	7.2	214.688
4	9.6	201.938
6	15.6	117.882

Table 5.4 Time of calculation for 64 atoms per unit cell with 2GB RAM on Distribute k-point

Number of CPU	CPU speed (GHz)	Cal. Time (Hr)
1	2.40	1554.438
2	4.80	852.904
4	9.60	682.479
4	9.60	416.996
8	19.20	343.754

Table (5.5), Table(5.6) shows the time calculation for 48 atoms and 64 atoms per unit cell using Data distribution method.

Table 5.5 Time of calculation for 48 atoms per unit cell with 2GB RAM on Data distribution method

Number of CPU	CPU speed (GHz)	Cal. Time (Hr)
1	2.40	472.986
2	4.80	253.493
3	7.20	205.671
4	9.60	191.795
6	14.40	106.613

Table 5.6: Time of calculation for 64 atoms per unit cell with 2GB RAM on Data distribution method

Number of CPU	CPU speed (GHz)	Cal. Time (Hr)
1	2.40	1560.173
2	4.80	831.441
4	9.60	649.589
4	9.60	386.561
8	19.20	306.141

Comparing the current time of calculation with the corresponding process carried with the 1 GB RAM per CPU we notice a drastic drop of Time for single CPU, which indicates the important effect of RAM in such calculation, regardless of the speed of CPU's. The same network speed and same machines were used.

These data has been drawn and fitted to the same equation used in the previous section, Fig.(5.7) shows the graph of 48 atoms per unit cell and the exponential was found to be 0.695 and Fig.(5.8) shows the graph of 64 atoms per unit cell and the exponential found to be 0.68 which is smaller than the 1 GB RAM per CPU. This indicates slower decay in this case with respect to the 1 GB RAM per CPU. The same trend has been noticed with the Data distribution method, but the exponentials found to be 0.72 and 0.7 for 48 , 64 atoms per unit cell respectively, this confirms faster decay than the Distribute k-point method as shown in Fig.(5.9) and Fig.(5.10).

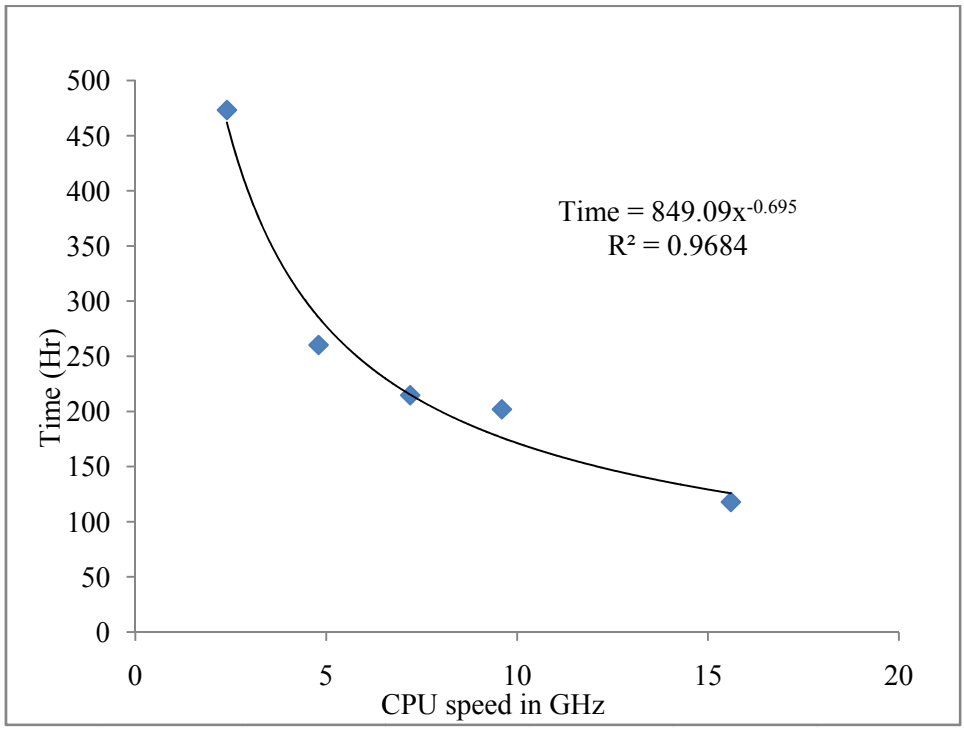


Figure 5.7 calculation for 48 atoms per unit cell and 2GB per CPU on Distribute k-point method

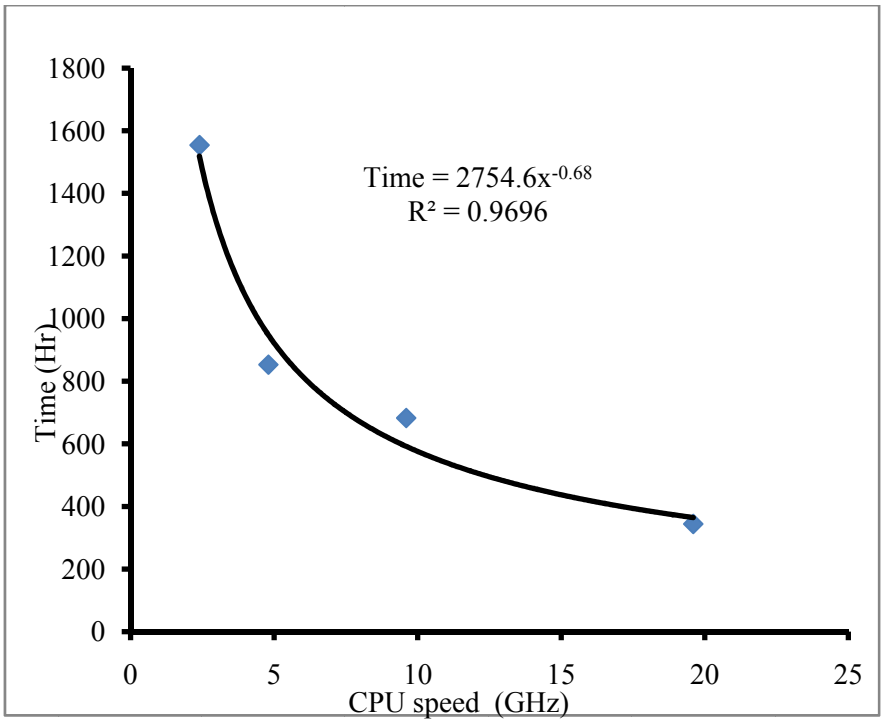


Figure 5.8 Time of calculation for 64 atoms per unit cell and 2GB per CPU on Distribute k-point method

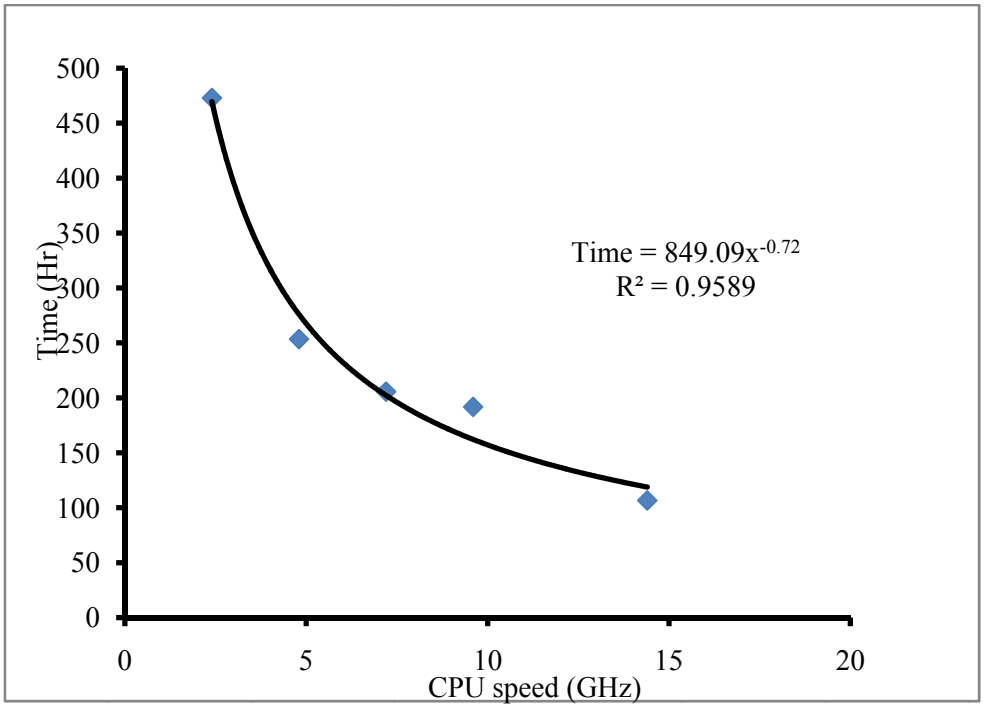


Figure 5.9 Time of calculation for 48 atoms per unit cell and 2GB per CPU using Data distribution

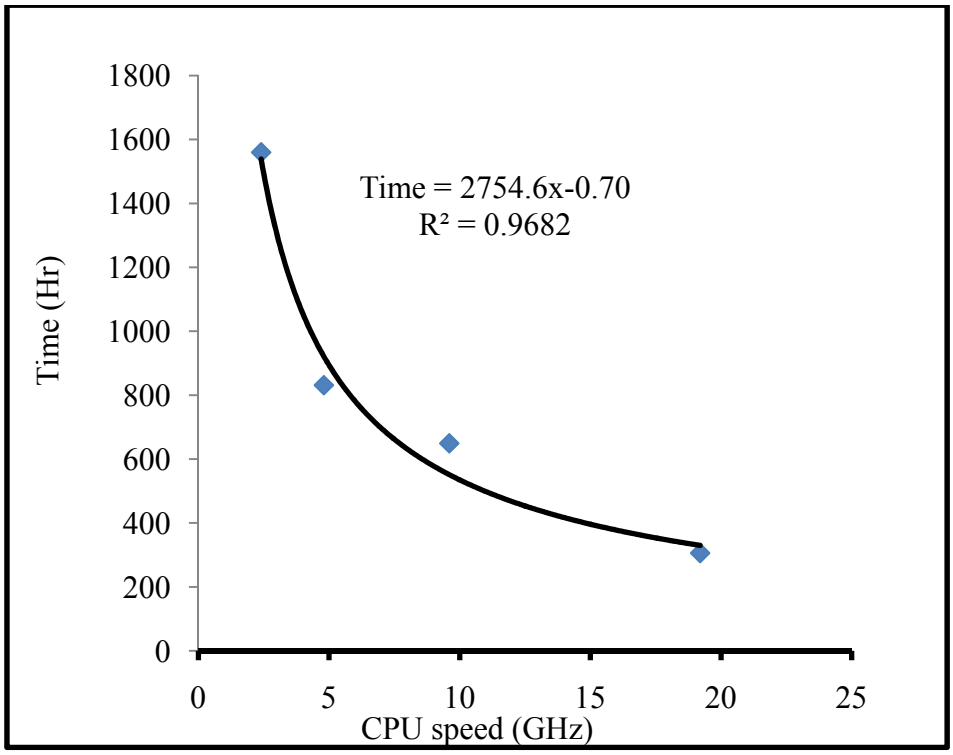


Figure 5.10 Time of calculation for 64 atoms per unit cell and 2GB per CPU using Data distribution

From Table(5.4) we have a gain two cases carried once on one machine with four CPU's and another run on two machines with the same total speed, we gain more RAM per CPU in the second run, the time of calculation in the first trial found to be 649.589 Hr, using the same conditions but distributed on rezekq24 and rezekq24t machines we get 386.561 Hr, it displays the effect of RAM here which is 4 GB per CPU, The need of RAM can be watched from the top command in Linux, Fig.(5.11) shows the used RAM and the unused RAM , if the amount of unused RAM is less than 500MB for the hole machine it's clear that there is a lack of memory, increasing RAM is not easy and not always possible.

```
top - 09:33:39 up 2 days, 23:21,  4 users,  load average: 2.00, 2.01, 2.00
Tasks: 174 total,   4 running, 168 sleeping,   2 stopped,   0 zombie
Cpu(s): 99.8%us,  0.2%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3060424k total,  2727224k used,   333200k free,      664k buffers
Swap:  2031608k total,   145788k used,  1885820k free,   250088k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
20830 root      20   0 1151m 1.1g 1956 R  100 37.7  54:44.89 lapw1c
20878 root      20   0 1177m 1.1g 1956 R  100 39.1  54:42.99 lapw1c
24250 root      20   0  2408 1040  780 R    0  0.0   0:00.07 top
    1 root      20   0  2112  524  504 S    0  0.0   0:00.97 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.30 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   0:00.00 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root      RT  -5     0    0    0 S    0  0.0   0:00.12 migration/1
    7 root      15  -5     0    0    0 S    0  0.0   0:00.20 ksoftirqd/1
    8 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/1
    9 root      15  -5     0    0    0 S    0  0.0   0:00.01 events/0
   10 root      15  -5     0    0    0 S    0  0.0   0:00.01 events/1
   11 root      15  -5     0    0    0 S    0  0.0   0:00.00 khelper
   63 root      15  -5     0    0    0 S    0  0.0   0:00.11 kblockd/0
   64 root      15  -5     0    0    0 S    0  0.0   0:00.01 kblockd/1
   67 root      15  -5     0    0    0 S    0  0.0   0:00.00 kacpid
   68 root      15  -5     0    0    0 S    0  0.0   0:00.00 kacpi_notify
  196 root      15  -5     0    0    0 S    0  0.0   0:00.00 cqueue/0
  197 root      15  -5     0    0    0 S    0  0.0   0:00.00 cqueue/1
  199 root      15  -5     0    0    0 S    0  0.0   0:00.00 ksuspend_usbd
  204 root      15  -5     0    0    0 S    0  0.0   0:00.00 khubd
  207 root      15  -5     0    0    0 S    0  0.0   0:00.01 kseriod
  246 root      20   0     0    0    0 S    0  0.0   0:09.56 pdflush
  247 root      20   0     0    0    0 S    0  0.0   0:09.72 pdflush
  248 root      15  -5     0    0    0 S    0  0.0   0:25.77 kswapd0
  300 root      15  -5     0    0    0 S    0  0.0   0:00.00 aio/0
  301 root      15  -5     0    0    0 S    0  0.0   0:00.00 aio/1
  395 root      15  -5     0    0    0 S    0  0.0   0:00.00 khvcd
```

Figure 5.11 a print screen for a top command while LAPW1 executed

This indicates that using parallel machines is a very good method to mimic calculations time.

6. Chapter Seven

Conclusion and Future Work:

In the present study, two parallel techniques have been used to examine the effect of RAM size, CPU speed and network speed on calculation time. For that reason two benchmarks have been used, each benchmark represent an input for the wien2k package program, each factor of the above mentioned that affect the calculation time has been tested for different initial conditions, network speed has been tested using distribute k-point method, different speeds of CPUs either on heterogeneous or homogenous systems, the calculations time for the five modules constituting one cycle is very big compared to the network delay, the network effect found to be independent from the other two factors.

Two measurable quantities have been implemented to test the effect of CPU speed and RAM size, speedup factor and an exponential formula, both showed that increasing CPUs number will decrease the calculation time until saturation is reached, clear difference between the two methods has been found, the same trend also has been found for the two benchmarks used, two machines with identical CPUs speed regarded as homogenous system and another machine with different speed regarded as heterogeneous system when combined with the other machines, the time of serial modules is very small compared with parallel modules and this is very clear through the speed up factor.

The same tests has been carried for two different size of RAM per CPU, the two methods distribute k-point and data distribution showed sharp drop in execution time of calculation when RAM size was increased.

It hasn't been noticed that Data distribution method display faster decay for larger number of CPU and mimic the time of calculation with respect to Distribute k-point method, the larger the number of atoms or the complexity of the case it's better to use Data distribution method, we notice this for the 64 atoms per unit cell where we save 21 hour of running the quad machine with four CPU's (10 GHz speed), while only 6 Hour saved in the case of 48 atoms per unit Cell.

Future computers are made of multi threads, this motivate programmers to work hard on parallel programming, as a result of this study RAM size available per CPU is the most effective factor for speedup, each computer PC can only handle maximum RAM size, manufactures should take care of this problem and put it into consideration.

Accuracy and number of distribution hasn't been studied, I wish I can continue my future work on this issue, parallel programming is a very challenging, interesting, and attractive for me, even though mind only work in serial.

References

[1] Burks W., Herman H. Goldstine ,von Neumann J., Preliminary discussion of the logical design of an electronic computing instrument,1946

[2] Carroll, A. B. and Wetherald, R. T. Application of Parallel Processing to Numerical Weather Prediction. Journal of the ACM (JACM), Vol. 14 Issue 3,(1967).

[3] Andrews, G. R., "Foundations of Multithreaded, Parallel and Distributed Programming "Addison-Wesley, 2000. ISBN 0-201-35752-6.

[4] Barros, S. and Kauranne, T.: On the parallelization of global spectral Weather models. Parallel computing 20 (1994) 1335 - 1356.

[5] Grama A, Gupta A., Karypis G. , Kumar V., Introduction to Parallel Computing, Second Edition, Addison Wesley, (2003)

[6] Peter S. Pacheco, "A User's Guide to MPI", 1998, www.open-mpi.org.

[7] Strandhagen, J. O.. "Operative Simulation in Production Management" Avhandling. (1994)

[8] Banks, J., Carson J. S., Nelson B. L., Discrete event system simulation, 1996, 2d ed, Upper Saddle River, New Jersey, Prentice-Hall, 2- Law, A. M., McComas, M. G., Secrets of successful simulation studies, Industrial Engineering 22: 47-48, 51-53, 72, 3-Scriber T. J. 'An introduction to simulation using GPSS/H', 1991,NewYork,Wiley.

[9] P. Blaha, K. Schwarz, G.K.H. Madsen, D. Kvasnicka, J. Luitz,WIEN2k, An Augmented-Plane-Wave + Local Orbitals Program for Calculatin Cryst Properties, Karlheinz Schwarz, Techn. Universitat, Wien, Austria, 2001.

[10] Goldenfeld, N., Lectures on Phase Transitions and the Renormalization Group, Perseus Publishing (1992).

[11] Anderson, P.W., *Basic Notions of Condensed Matter Physics*, Perseus Publishing (1997).

[12] Landau, L.D. and Lifshitz, E.M., *Statistical Physics Part 1*, vol. 5 of Course of Theoretical Physics, Pergamon, 3rd Ed. (1994).

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

[14] Walter Ashley Harrison (1989). Electronic Structure and the Properties of Solids.

[15] Hohenberg, P. and Walter K. (1964). "Inhomogeneous electron gas". Physical Review 136 (3B): B864–B871.

[16] Kresse, G. and Furthmüller, J., *Phys. Rev. B* 54, 11169 (1996); Comp. Mat. Sci. 6, 15 (1996).

[17] http://www.gaussian.com.

[18] Ashcroft N, N. W. and Mermin, N. D., Solid State Physics, (Holt, Rinehart, and Winston, New York, 1976).

[19] Charle K., "Introduction to solid state physics" 8th edition. Willey 2005.

[20] Schrödinger, E. "An Adulatory Theory of the Mechanics of Atoms and Molecules". Physical Review 28 (6): 1049–1070. (1926).

[21] Duncan, Ralph , "A Survey of Parallel Computer Architectures". *IEEE Computer*: 1990..

[22] Aho, E. Vanne, J. Hamalainen, T.D. Kuusilinna, K.Inst "Block-level parallel processing for scaling evenly divisible images", Circuits and Systems I: Regular Papers, IEEE Transactions on **Volume:** 52 **page(s):** 2717 - 2725 2005

[23] Marijn J.H., Maaren H., Parallel SAT Solving using Bit-level Operations,2008, Journal on Satisfiability, Boolean Modeling and Computation,99-116.

[24] Culler D., Singh J., Gupta A. Parallel Computer Architecture – A Hardware/Software Approach. Morgan Kaufmann Publishers, 1999

[25] Hennessy J. , Patterson D. , computer architecture a quantitative approach, 4th edition. Morgan Kaufmann Publishers, 1990.

[26] http://www.ateji.com/px/patterns.html#data Data Parallelism using Ateji PX, an extension of Java, 2-Hillis, W. Daniel and Steele, Guy L., Data Parallel Algorithms Communications of the ACM December 1986, 3-Blelloch, Guy E, Vector Models for Data-Parallel Computing MIT Press 1990. ISBN 0-262-02313-X

[27] Amdahl G. ,"The validity of the single processor approach to achieving large-scale computing capabilities".(April 1967) In *Proceedings of AFIPS Spring Joint Computer Conference*, Atlantic City, N.J., AFIPS Press, pp. 483–85.

[28] https://computing.llnl.gov/tutorials/parallel_comp/