ACCELERATING STENCIL COMPUTATION IN MULTI-CORE ARCHITECTURE

AMAR RAEED KHORSHID AL-HILALI

JANUARY 2015

ACCELERATING STENCIL COMPUTATION IN MULTI-CORE ARCHITECTURE

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES OF
ÇANKAYA UNIVERSITY

BY

AMAR RAEED KHORSHID AL-HILALI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF

COMPUTER ENGINEERING

JANUARY 2015

Title of the Thesis: **Accelerating Stencil Computations in Multi-Core Architectures.**

Submitted by **Amar Raeed Khorshid AL-HILALI**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.

Prof. Dr. Taner ALTUNOK
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Müslim BOZYİĞİT
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Emre SERMUTLU
Supervisor

Examination Date: 16.01.2015

Examining Committee Members

Assist. Prof. Dr. Emre SERMUTLU	(Çankaya Univ.)
Assist. Prof. Dr. Bülent EMİROĞLU	(Kırıkkale Univ.)
Assist. Prof. Dr. Abdül Kadir GÖRÜR	(Çankaya Univ.)

STATEMENT OF NON-PLAGIARISM PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Amar, ALHILALI
Signature :
Date : 16.01.2015

ABSTRACT

ACCELERATING STENCIL COMPUTATION IN MULTI-CORE ARCHITECTURES

AL-HILALI, Amar
M.Sc., Department of Computer Engineering
Supervisor: Assist. Prof. Dr. Emre SERMUTLU

January 2015, 60 pages

Stencil computations are common in linear systems of equations, numerical solutions of partial differential equations, molecular dynamics and many other scientific problems. For large structures, long computation times are an important problem. Increasingly higher number of cores are used in parallel for such computations, but still, the speedups are not sufficiently satisfactory. The main aim of this thesis is increasing the cache reuse and minimizing number of memory accesses by optimizing loop structures. We present and test several algorithms and improvements on them to get an optimal runtime.

Keywords: Stencil Code, Multicore, Jacobi, Cache Reuse, Convergence, Tiling, Iteration.

ÖZ

ÇOK ÇEKİRDEKLİ MİMARİLERDE ŞABLON HESAPLAMALARININ HIZLANDIRILMASI

AL-HILALI, Amar
Yüksek Lisans, Bilgisayar Mühendisliği Anabilim Dalı
Tez Yöneticisi: Yrd. Doç. Dr. Emre SERMUTLU
Ocak 2015, 60 sayfa

Şablon hesaplamaları doğrusal denklem sistemlerinde, kısmi diferansiyel denklemlerin sayısal çözümlerinde, moleküler dinamikte ve daha başka pek çok bilimsel problemde yaygın olarak karşımıza çıkarlar. Büyük yapılar için uzun hesaplama süreleri önemli bir problemdir. Bu tür parallel hesaplamalar için giderek artan sayıda çekirdek kullanılmaktadır ancak hala hızlanmalar yeterince tatmin edici değildir.

Bu tezin ana amacı döngü yapılarını iyileştirerek işlemci belleğinin tekrar kullanımını arttırmak ve sistem belleğine erişim sayısını en aza indirmektir. En iyi işlem zamanını elde etmek amacıyla birden fazla algoritma ve onların iyileştirilmiş halleri sunulmuş ve test edilmiştir.

Anahtar Kelimeler: Şablon Kodu, Çok Çekirdek, Jacobi, Bellek Yeniden Kullanımı, Yakınsama, Hücreleme, Döngü.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Assist. Prof. Dr. Emre SERMUTLU for his supervision, special guidance, suggestions and encouragements through the development of this thesis.

It is pleasure to express my special thanks to my family who stood by my side during difficult times and then to all my friends.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

FIGURES

LIST OF TABLES

TABLES

LIST OF ABBREVIATIONS

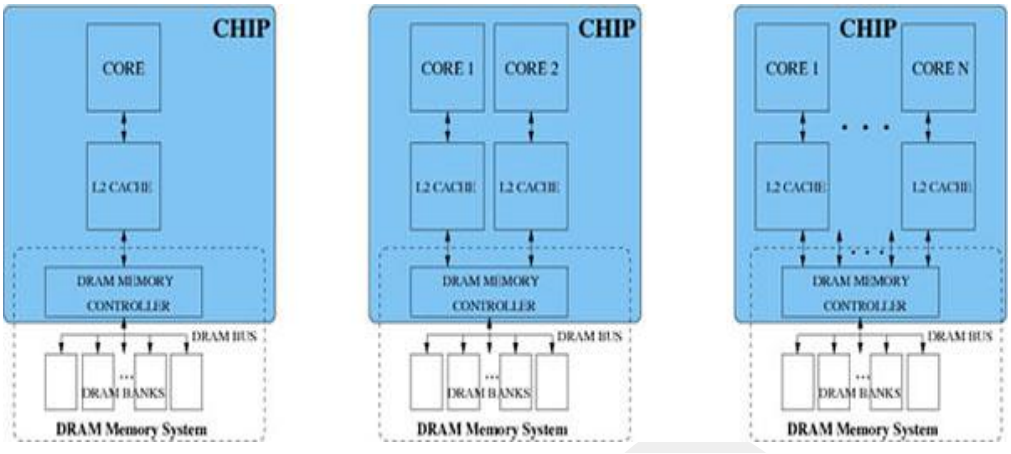
PDE	Partial Differential Equation
SOR	Successive Over – Relaxation
TLB	Translation Look aside Buffer
CMP	Chip Multiprocessor
API	Application Programming Interface
CPU	Central Processing Units
PDE	Partial Differential Equation
SOR	Successive Over – Relaxation
TLB	Translation Look aside Buffer
CMP	Chip Multiprocessor
API	Application Programming Interface
CPU	Central Processing Units
ccNUMA	Cache Coherent Non-Uniform Memory Architecture
ARB	Architecture Review Board
LRU	Least Recentally Used
FIFO	First In First Out

CHAPTER 1

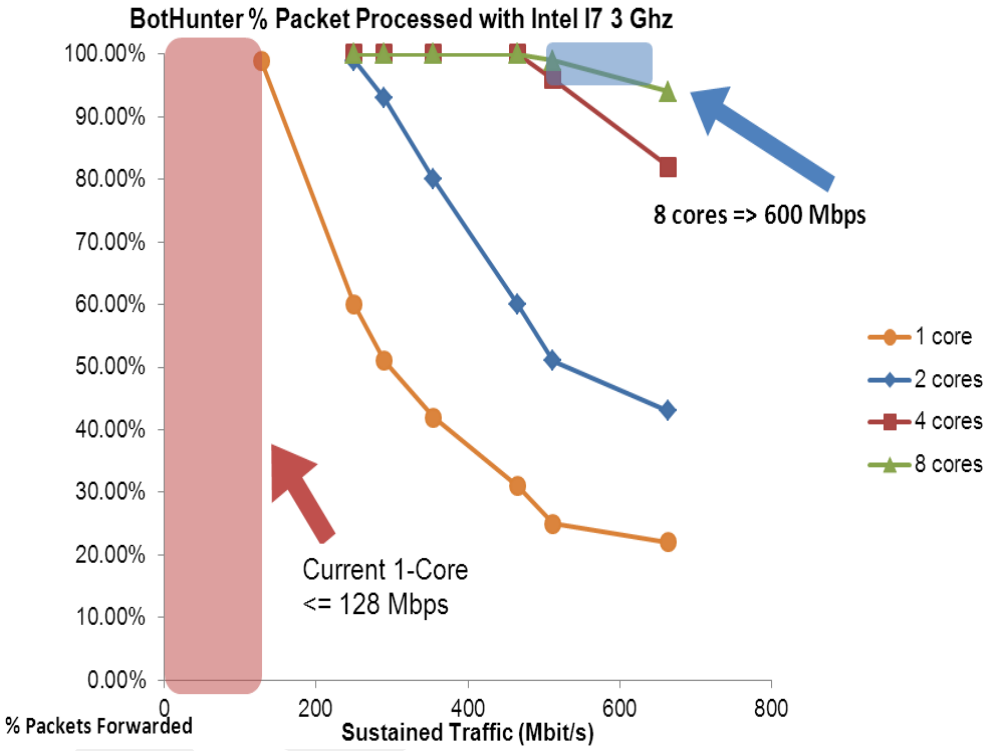
INTRODUCTION

1.1 Background

The ability of performing multiple tasks at same time is calling parallel computing or parallel processing. This term is used in context of human cognition, which dialed with the ability of brain to solve several problems simultaneously. For example the brain is divided what sees into many contents: color, motion, shapes and depth. These content which brain divided are individually analyzed and then compared to stored inside memory, in order to used to identify what you viewing. The second using of parallelism will be in the parallel computing which simultaneous using more than one CPU or processor core in order to execute multiple program or solving huge number of equations. Optimally, parallel processing makes result appearance faster, because there is more engine (CPUs, processing COREs) executing it see fig. 1(A). Actually, it is often difficulty to divide a program in such a way that split up (CPUs and COREs) execute different portion without middling some each other. In the past most computers has single – CPU single CORE , but there was possibility to perform parallel processing by connecting two or more computer each other by setting up network. Today the computer has more than one core, so we can use the parallelism easily. In this thesis I prefer to notice that parallelism is differ from concurrency , because concurrency is a term used in operating systems and database communications logically which indicate to the property of systems in which multiple tasks are still logically active, and progress at same time by interfering system implementation task order, thus creating illusion of implementing instructions simultaneously. While, parallelism is used by computer community to identify executing simultaneously, and it is goal is solving problems in an optimal time or solving huge number of equations at less time, see fig. 1(B) .



(A)



(B)

Figure 1 (A and B) Difference between single-core and multi-core

1.2 Overview

This thesis try to present the classical methods of stencil code, which deals with solving partial differential equation in less time. Actually these methods have been very classic, old, simple understanding and implementation as mentioned in section 2.2. At the same time it present some topics which deals with these methods and the related works with these methods. The structure of arrays calibrate stencil codes as apart from other methods like finite element method, finite difference codes which invest on consistent grids can be subedit a stencil codes. Thesis content figures, tables, schemes, clarification, parts of codes, and appendices. Thesis dropping light on stencil codes which found commonly in codes of computer simulations, the Jacobi kernels, the Gauss-Siedle method, image processing and cellular automata. The main idea of this thesis is to improve on algorithm to be able to competitive with existed methods, by using optimal time for solving the problems by best of implementing of cache and reusing data in array. The methodology which used in this thesis is varied from the others in reusing data, modality of implementing of cache, separating of values in an array and solving these values in array. The critical case for stencil performance is exploiting locality in the time dimension .The reason of this criticality is data grid size in real application exceed the capacity of L1,L2 cache on current. The problem of performance are largely, because of enough cache requiring, and can reducing this by tiling method. This thesis consider the most common classical iterative techniques for the linear systems: the Jacobi, and Gauss-Seidel methods. These method are constantly denoted as stencil code, and the reason is updating of array elements according some stable patterns.

1.3 Motivation

The main idea of this study was to develop an algorithm which could competitive and overcome to another algorithm to improving executing of large equation runtime. This motivation leads to working and completion this experiment and do the best to reach to an enhancement which could to competitive with new algorithms and overcome on it.

1.4 Organization of Thesis

Thesis is comprises 4 chapter organized to be Reference to who deal with this study, first chapter is starting with introduction of this work then followed by overview and motivation, in the second chapter it started talking about related work, dealing with stencil code and the explaining basic iterative method in this section study explain three main method of iterative, then the study turns to talking about multi-processor systems and the type of multi-processor systems, continuing in other section subject of caching memory, giving way to enter in Jacobi method by cross-cutting methods by detail, then talking about tiling Jacobi to open minds to up-to-date methods of blocking by detailing and giving an example or figure illustration for each one. Then start with three dimension Jacobi method and blocking of three dimension method, to finish this chapter by explaining the Gauss Seidel method and compare it with Successive Over – Relaxation method. In the third chapter, the study starting to clarification of important tool to performing this experiment to the fullest, so it starting from clarification of open-MP, then transmitted to the usage of this (API), then starts to talking about thesis project to divides the work into two part, first is theory part and the second is software part to getting the result and produce the result to reader. The last chapter simply is conclude the new improvement from this work and recommend some suggestions for feature working.

CHAPTER 2

STENCIL CODE

2.1 Stencil Code

Stencil computations appeared in an extensive field of applications of computational sciences. This is an important part of run time in many scientific simulation codes. A primary application of stencil-based computations are numerical Partial Differential Equation solvers that implement a finite difference or multi-grid method. Where Stencil computation is used to solve in Partial Differential Equation solving it is also using in image manipulation. These kernels are used in outermost time loop to make a huge number of sweeps on multi-dimensional grid so that the valuations of any grid point are modified according the valuation of neighboring points.
Partial Differential Equations can be simplified numerically by first discretizing the computational domain, e.g., a regular Cartesian grid together with a stencil depending on f and then using a Newton –Raphson – type algorithm, this will require evaluating the sparse Jacobian of f on the discretized domain, which is a very computation intensive operation for complicated Partial Differential Equations [1].

High degree of temporal locality are exhibited in most of stencils, because any update operation needs to access neighboring values. The critical case for stencil performance is exploiting locality in the time dimension. The reason of this criticality is that, data grid size in real applications usually exceed the capacity of L1 and L2 cache on current systems[2].

Systems of linear equations for which numerical solutions are needed are often very large, making the computational effort of direct methods, such as Gaussian elimination, prohibitively expensive. For systems that have coefficient matrices with appropriates structure- especially large, spare systems (i.e., systems with many coefficient whose value is zero) – iterative techniques may be preferable. Iterative

solvers such us Jacobi and Gauss-Seidel are very important because they are the building blocks of many other methods. Their computational properties are similar.

This thesis considers the most common classical iterative techniques for the linear systems: the Jacobi method. This method are constantly denoted as stencil code, and the reason is updating of array elements according some stable patterns. The performance of each technique is illustrated for several small systems and for linear system in example from Poisson's equation. Some theoretical results are presented to give guidance in determining when the method may be useful. In Open-MP API specification for parallel programming is provided for each technique. In stencil-based computations, each node in a multi- dimensional grid is updated with weighted values contributed by neighboring nodes [3].

2.2 Basic Iterative Method

Iterative scheme according to formula of stencil computations in three spatial domains are applied a lot of math practice e.g. linear problems solving and multi-grid techniques. The optimal prototype for Poisson problems is Jacobi method, while Gauss-Seidel tries to solve Laplace problems. The big size of data sets probability may be the reason for using these methods which are known as data- dense and accessible main memory bandwidth forces upper limit of execution.

Relaxation of coordinate was the first method which used in solving of large linear systems starting with a gained approximated solution. The methods which I deal with in this thesis modify the component of the approximation a few at time a certain order, until convergence. Any step in this operation is called relaxation steps.

Iterative methods formally produce the solution of linear system after ultimate number of steps. At each step it demand calculation the remaining of the system. In full matrix case, the computation (calculation) cost is based on the order of n^2 operation for every iteration, in order to comparing it with an overall costs of their order of $\frac{2}{3}n^2$ operation which required by direct methods. Iterative method can be competitively with direct methods as long as the number of iteration that needed to convergence is either independent of n or scales sub linearly [4]. In the scattered matrices case direct method may be inappropriate because the content which filled

the matrices. Although the direct solvers extremely could be innovated on scattered matrices characteristics. The main idea of iterative method is to build square of the vector $x^{(k)}$ that deal with the property of convergence:

$$x= \lim_{k\to\infty} x^{(k)}$$

Practically iterative process is layover at minimum value of (n) such us

$$\|x^{(n)} - x\|<\varepsilon,$$

While ε is a constant tolerance ||.|| is any appropriate vector norm. Yet, since the same solution is clearly not available, it is indispensable to identify appropriate stopping criterion to monitor the convergence of iteration. There are many options in iterative solutions such as **pure iteration** which compute each $x_{(k-1)}$ from x_k- $p^{-1}(Ax_k - b)$. This is called stationary, because the application at every step is the same. Convergence to $x_\infty = A^{-1}b$, shown below, acceleration when all eigenvalues of $M = I - p^{-1}A$ are small. The second options of iteration is **multi-grid** for one job in Gauss Seidel and Jacobi is very well, they takeaway high frequency component to drop a smooth error. The main idea is to shift to a coarser grid- where the reminder of error can be destroyed. It is often significantly successful. The third option is **Krylov spaces**, which consist of all combination of b, Ab, A^2b, ... and Krylov methods expect the best combination.

However Iterative term is mention to a wide range of techniques that implement successive approximation to gain more accurate solution to linear system at each step. In this thesis will touch two types of iterative methods. As thesis referred in this section, direct method can be older, simpler to understand and more implementable, but it still not efficient to use nowadays. While in direct (no stationary) methods are comparatively recent development the analyzing of indirect (no stationary) methods is usually hard to understand, but these methods are still more efficient. These methods are based on the idea of sequences of perpendicular vectors. The rate at which an

iterative method converges depends primarily on the spectrum of the matrix coefficient [5]. Therefore, iterative methods generally include another matrix that transforms the coefficient into one with a more appropriate spectrum. This transformation is called preconditioning. In fact the iteration methods fail without preconditioned. This chapter will introduce the both methods with full detail.

2.3 Multi-Core Processor

A multi-core processor is single computing component with multi or more freelance actual (central processing units) which have units for reading and executing the instructions. This instruction or order is normal CPU instruction like move data, adding and branching, while multi- cores try to apply multi instructions simultaneously. Manufacturers usually try to combine the cores onto a single integrated circuit die (known as a chip multiprocessor or CMP), or onto multiple dies in a single chip package [6]. Processors were in general developed with one core only. Rockwell International in middle of 1980 produced versions of the 6502 with two 6502 cores on one chip as the R65C00, R65C21, and R65C29, joining the chip's pins stand by clock stages [5]. In 2000 AMD and Intel were produced the other multi core styles. The company starts with two cores and continue to work to increase the number cores till nowadays Table 1 explain number of cores and the name of company, which has developed this product and the brand name of these product.

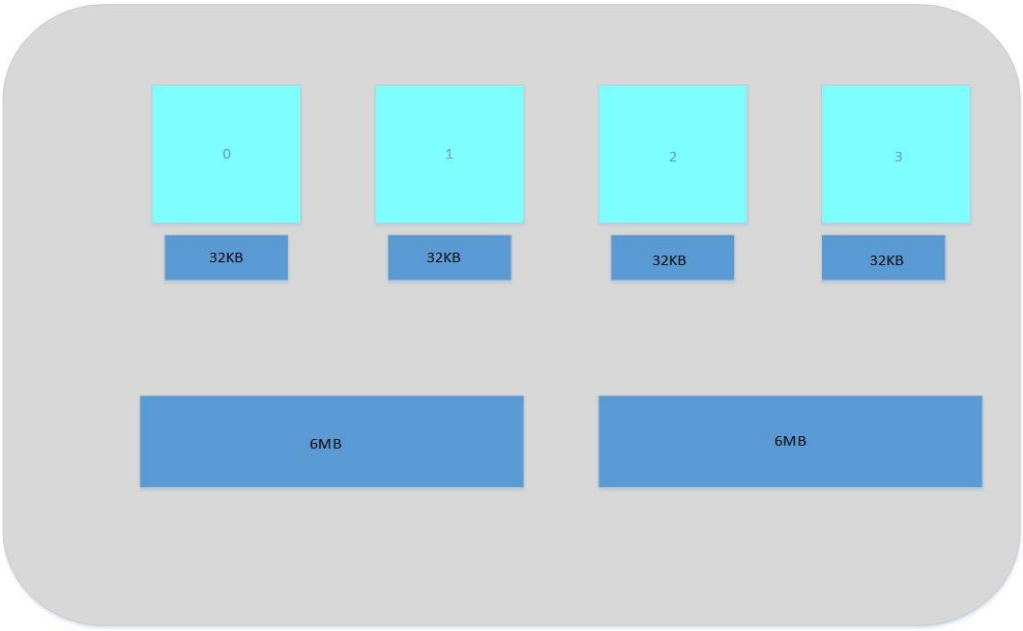
Aboard selection of model m86 which based on multi-core processor has selected to attempt variants of wave front parallelization model see section 2.5.5, to explain its performance efforts. Most of these chips properties a large outer level cache which is portion by two (Intel Harpertown), four (Intel Nelham EP), six (Intel Westmere EP) or eight cores (Intel Nelham ex) [see Table 2]. An extreme numbers of Cores which are joined outer level are L2/L3 cache which mentioned as "L2/L3" group see Fig. (2).

Number of cores	Brand name of Intel	Brand name of AMD
Two Cores	Intel Core Duo	AMD Phenom II X2
Four Cores	Intel's i5 and i7 processors	AMD Phenom II X4
Six Cores	Intel Core i7Extreme Edition 980X	AMD Phenom II X6
Eight Cores	Intel Xeon E7-2820	AMD FX-8350

Table 1. Number of Cores and Brand Name of Each Company

In Harpertown processors, which run onto two cores is denoted as a quad-core chip, because four cores are placed in package as shown in Fig. (3). In this case it is accumulate as two according L2 groups without using L3 for all four cores. Additionally a whole remodeling of memory sub-system permit for a substantial to raise up in memory at the cost of identifying a ccNUMA. It is type of memory style, which used in multi-processor systems, when memory location relative to processor are control memory access time.

Figure 2 (A and B) The memory performance on Jacobi program written by C++



(A) Core 2 Quad

(B) Nehalem EP

Figure 3 Microarchitecture of core 2 quad and Nehalem EP

Microarchitecture	Intel Core2	Intel Nehalem Ep	Intel Westmere	Intel Nehalem EX	AMD ISTANBUL
Model	Xeon X5482	Xeon X5550	Xeon X5670	Xeon X7560	Opteron 2435
Clock [GHz]	3.2	2.66	3.93	2.26	2.6
Core per socket	4	4	6	8	6
SMT threads per core	N/A	2	2	2	N/A
L1 cache	32kB	32kB	32kB	32kB	64kB
associativity	8	8	8	8	2
L2 cache	2x6 MB(shared)	4x256 kB	6x256 kB	8x256 kB	6x512 kB
associativity	24	8	8	8	16
L3 cache	-	8MB	12MB	24MB	6MB
associativity	-	16	16	24	48
Bandwidth[GB/s]	-	-	-	-	-
Theorical socket BW	12.8	32.0	32.0	17.1	17.1
STREAM 1 thread	4.6	11.9	11.0	5.3	7.2
STREAM socket NT/no NT	4.8/5.6	18.5/23.7	21.0/23.6	9.1/13.6	9.8/11.4

Table 2 Test Machine Specification, The Cache Line Size is 64 Bytes for All Processor and Cache Levels.

2.4 Cache Memory

Central processor cache is generally used to decrease time ratio of data accessing to main memory. At the same time it is a smaller and faster memory, which stored data copies of data which extremely used by main memory locations. A case which considered an essential tradeoff between cache latency and hit is that a larger cache have better hit rate but longer latency. When processor required to write or read something from main memory its check first if it is stored on cache or not, the processor as soon as read from or write to cache, that faster than reading from and writing to in main memory. Most of CPUs have three free cache: **instruction cache** use to accelerate runnable instruction fetch. The second one is **data cache** accelerate data fetch and store and **Translation Look aside Buffer (TLB)** which is used to accelerate virtual to physical address translation for both instruction and data. The data cache usually used for pecking order of multi cache level (L1, L2…) see fig (4). The ratio of accessing that produce in cache is called hit rate, which can evaluate the influence of cache for given algorithm or program. Reading misses beads running because they needed data to be transmitted from main memory, which is slower than reading from cache, while writing happened without this delaying, because processor can make running while the data copied to main memory in background.

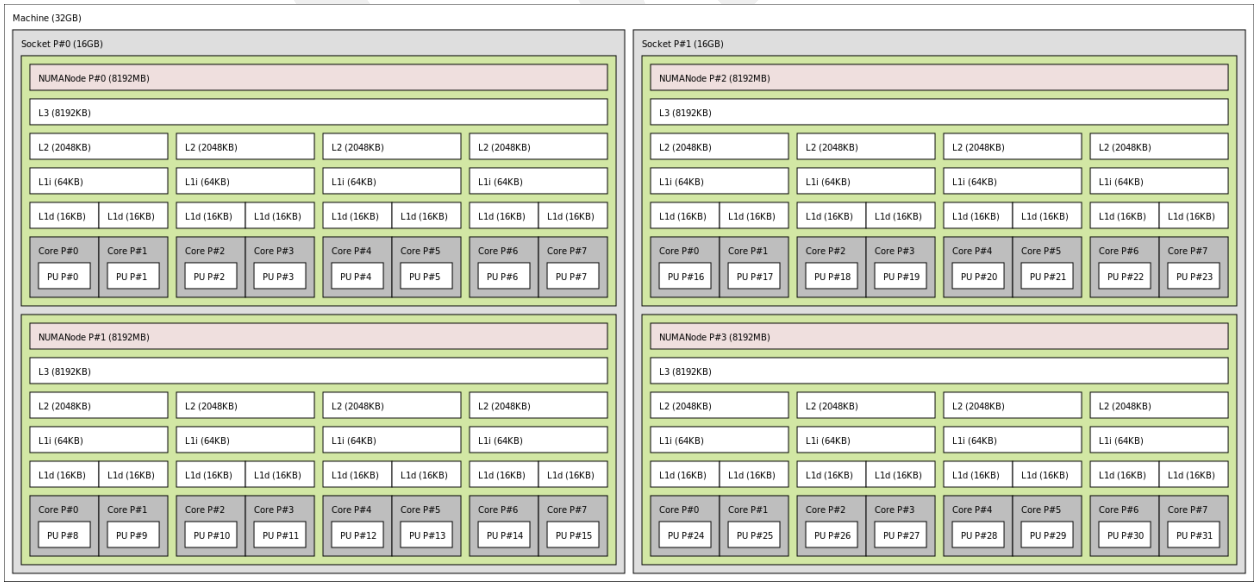


Figure 4 Memory hierarchy of AMD bulldozer.

The advantage of cahce in “temporal locality” , when used constants data like messages, column and high low limits, which are used repeadtly in cache. The other advantage is from “Spatial Locality”, in order to run the next instruction or to process the set of data, it has oftened in next line. The more sequential they are, the greater chance is when the next item is indeed existed in the cache “cache hit”, if the next item is not existed in the cache “cache miss” will happened , retrivung data from main memory should be in this case , that makes process slower. There are several design fo cache memory. This study is point out on these design, cache fetch algorithm is one of these design type, which used to decided when it realy needed to fetch data to cache. This algorithm has many scenario's, one of these scenario’s is guessing that the information will be needed in the next instruction, so it fetching information this process called “prefetching”. The other design algorithm is cache placement algorithm, this algorithm are depended on associatively of information, some are largely associative memories be expensive and somewhat are slow, so in genral the cache is organized as a group of smaller associative memories. In this case only one of the associative memories has to determine if the desired data is existed in cache. Replacement algorithm, when cache memory is fulled , and information is remaded from main memory by CPU, so unnecessary information must be removed from cache in order to replacemented by remanded information, for this process there are many methods, FIFO(First In First Out) or LRU(Least Recentally Used) .

2.5 Jacobi Method

Jacobi method is a famous algorithm for solving different equation on square domain by fixed patterns solving. Jacobian method has many strategies for solving the partial differential equations (e.g. 2D Jacobi, 3D Jacobi). Let us consider 2D array as example. In this example we divided an array in to two part: the first part is the constant part which is represent the boundary part of array, the second part is the variable part which represent body of array. Any element of body contract with fixed value of temperature on four boundary and partial differential equations is solved for elements of body to determine their temperature as average of the four neighbor as well as the value itself. (Fig.(5)). This task is taken as the computational task,

number of iteration are applied on data to recomputed the average repeatedly, and we reach final result when desired accuracy is obtained. (Fig. (6)). In this case by starting from initial solution of 0, The right to left are fixed at 1, while upper and lower case will be 0, after number of iteration the system converge against saddle-shape [7] (Fig.(7)). In Jacobi method the amount of data re-employ is depend on the number of its neighbor values for example 7- point stencil which are most commonly a depend on up, down, right and left neighbors as shown in Fig.(8) In this 7-point stencil four of seven data values are re-employed in every iteration. While in 27 point stencil eight of twenty seven point of stencil are re-employed at every iteration see Fig. (9) . The term of high order stencil is not common like the previous which referred to in this thesis. For example 9-point stencil denote in ultimate variation methods, while 27-point stencil denoted in multi-grid solvers and advection code [8].

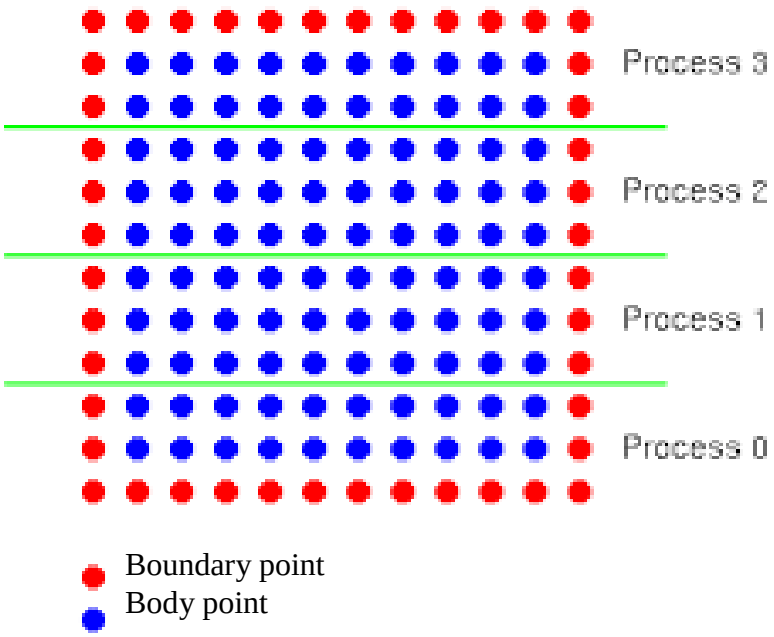


Figure 5 The boundary and body points

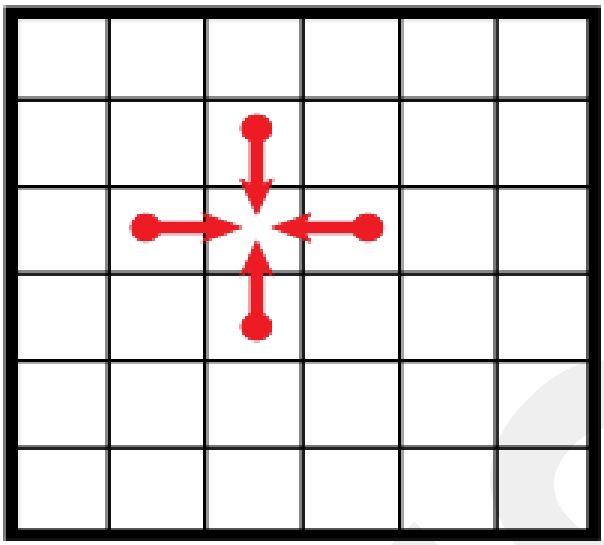
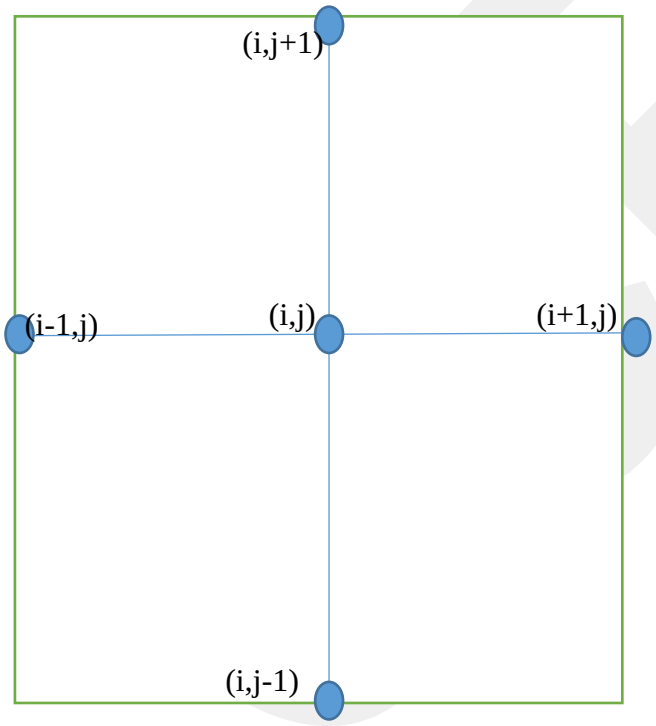


Figure 6 Data dependencies of selected cell in the 2D array**.**




```
For (t=0; t<time step; t++)
    {
       For (i=1 ; i<n-1; i++)
          {
             For (j=1 ; j<n-1 ; j++)
                {
        A1[i,j] = A0[i,j] +
A0[i+1,j]+
 A0[i-1,j]+A0[i,j+1]+A0[i,j-1] ;
                }
          }
Temp=A0;
A0=A1;
A1=temp;
    }
```

Figure 7 2D Jacobi with four neighbor.

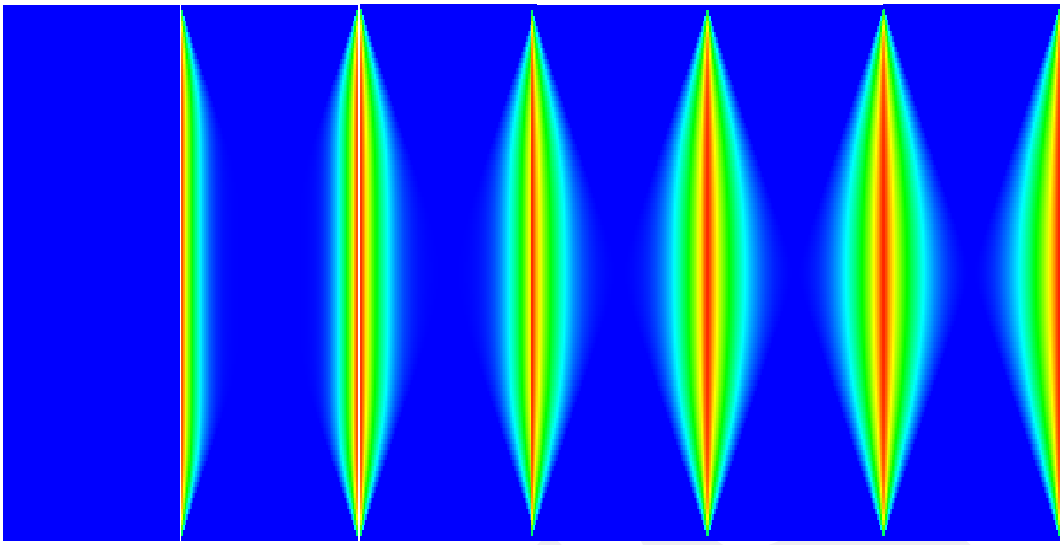


Figure 8 Data iteration on a 100^2array

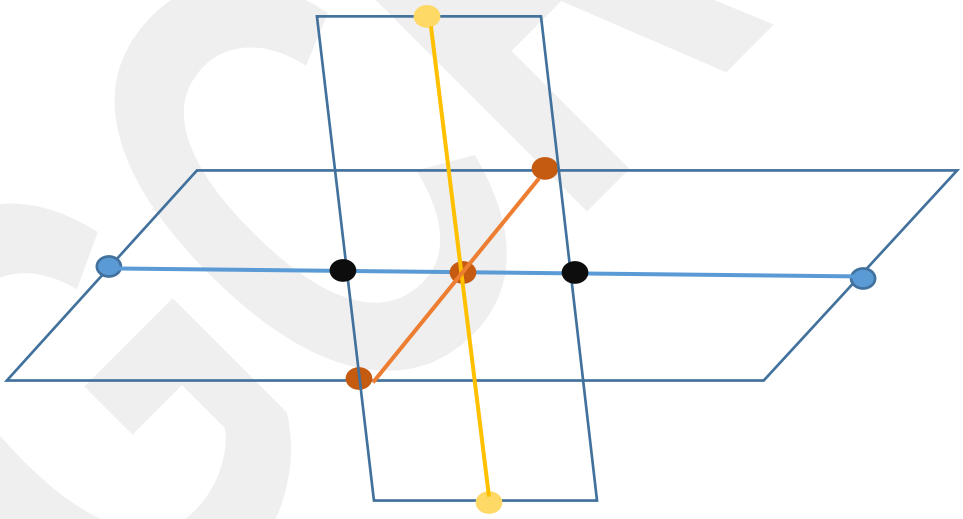


Figure 9(A) 7-point stencil

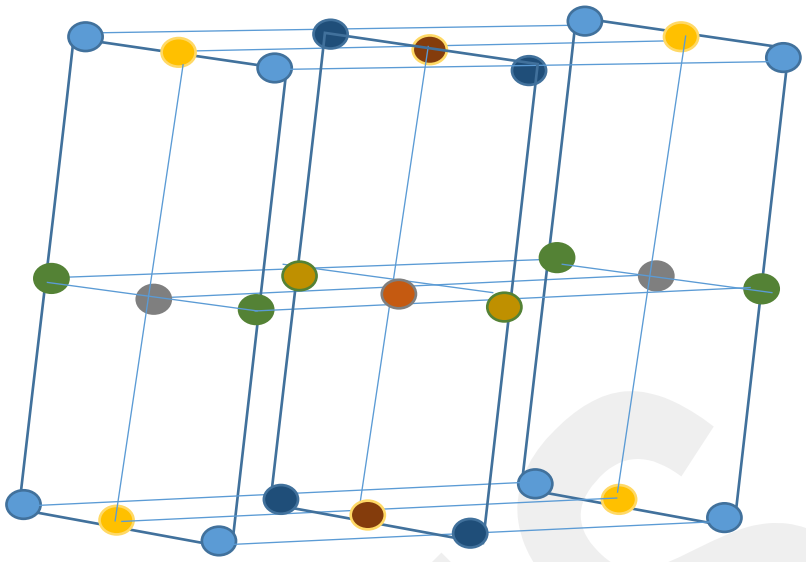


Figure 9(B) 27-point stencil

2.5.1 Description of Jacobi

In this thesis I will try to describe Jacobi method by given example below, if we suppose that we have square system of n linear equation:

$Ax = b$

Where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \ldots\ldots\ldots\ldots\ldots\ldots a_{1n} \\ a_{21} & a_{22} \ldots\ldots\ldots\ldots\ldots\ldots a_{2n} \\ & \vdots \\ a_{n1} & a_{n2} \ldots\ldots\ldots\ldots\ldots\ldots a_{nn} \end{bmatrix}$$

$$x = \begin{matrix} x1 \\ x2 \\ xn \end{matrix},$$

$$b = \begin{matrix} b1 \\ b2 \\ bn \end{matrix},$$

The solution will be obtained iteratively by:

$X^{(k+1)} = D^{-1}\left(b - Rx^{(k)}\right)$.

The element based will be:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}\right), \qquad i = 1,2, \ldots., n.$$

For example if we have this system

$$a_{11}x_1 + a_{12}x_2 + \ldots\ldots\ldots\ldots\ldots a_{1n}x_n = c_1 ,$$
$$a_{21}x_1 + a_{22}x_2 + \ldots\ldots\ldots\ldots\ldots a_{2n}x_n = c_2 ,$$
$$\vdots$$
$$a_{n1}x_1 + a_{n2}x_2 + \ldots\ldots\ldots\ldots\ldots a_{nn}x_n = c_n ,$$

Has single solution

The coefficient matrix a has no zeros on its main diametrical, namelya_{11}, a_{22}, a_{nn}; are nonzero, To solve the first equation for x_1 , the second equation forx_2, and so on to gain rewritten equations:

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - \cdots - a_{1n}x_n),$$
$$x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - \ldots - a_{2n}x_n),$$
$$x_3 = \frac{1}{a_{nn}}(b_n - a_{nn}x_1 - \cdots - a_{n-1}x_{n-1}),$$

This accomplishes considered as one iteration only.

2.5.2 Algorithm and convergence of Jacobi

The initial of Jacobi method will start with:

K=0;

While convergence will not reached do

for i := 1 step until n do

σ=0

for j := 1 step until n do

if j ≠ i then

σ= σ+a_{ij} $x_j^{(k)}$

end if

end (j-loop)

$$x_i^{(k+1)} = \frac{(b_i + \sigma)}{a_{ii}}$$

While bounded linear operator of matrix iteration is smaller than 1 the standard convergence condition will be:

$$P(D^{-1}R) < 1$$

Converge in the method will be guaranteed if the A matrix accurately diametrical dominate, the absolute value of diametrical term is greater than the sum of absolute values of other terms [9]

$$|a_{ii}| > \sum_{j \neq i} |a_{hj}|$$

2.5.3 Jacobi eigenvalue algorithm

Eigenvalue means each of a set of values of a parameter for which a differential equation has a nonzero solution (an Eigen function) under given condition. In general Jacobi Eigenvalue algorithm is used in numerical linear algebra it is an iterative method used for calculating eigenvalues and eigenvectors of real square matrix is named then Carl Gustavo Jacob Jacobi , who first suggest Jacobi theory in 1948, but it has been famous in 1950's with appearance of computers. The simulation of the algorithm is shown below

Let S be a square matrix, and G = G(i,j,θ) be a Givens rotation matrix. Then:

$$S' = GSG^T$$

Is symmetric and similar to S.
Furthermore, S′ has entries:

$$S'_{ii} = c^2 S_{ii} - 2scS_{ij} + s^2 S_{ii}$$

$$S'_{jj} = s^2 S_{ii} + 2scS_{ij} + c^2 S_{jj}$$

$$S'_{ij} = S'_{ji} = (c^2 - s^2)S_{ij} + sc(S_{ii} - S_{jj})$$

$$S'_{ik} = S'_{ki} = cS_{ik} - sS_{jk} \qquad k \neq i,j$$

$$S'_{jk} = S'_{kj} = sS_{ik} + cS_{jk} \qquad k \neq i,j$$

$$S'_{kl} = S_{jk} \qquad k,l \neq i,j$$

When s = Sin(θ) and c =Cos(θ).
Since G is orthogonal, S and S′ have the same Fresenius norm ||·||F (the square-root sum of squares of all components), however we can choose θ such that S′ij = 0, in which case S′ has a larger sum of squares on the diagonal [10]:

$$S'_{ij} = \cos(2\emptyset)\, S_{ij} + \frac{1}{2}\sin(2\emptyset)(S_{ii} - S_{jj})$$

Set this equal to 0, and rearrange:

$$\tan(2\emptyset) = \frac{2S_{ij}}{S_{jj} - S_{ii}}$$

if

$$S_{ii} = S_{jj}$$

$$\emptyset = \frac{\pi}{4}$$

In order to enhance this effect, S_{ij} should be biggest –off diametrical component are parataxis (real) eigenvalue of S.

2.5.4 Block Jacobi Method(Tiling)

The previous methods were refers to point or (line) iterations at Mostly, but in this section we will deal with devising block method , by providing that D is referring to the blocking diametrical matrix ,which used by entering M*M diametrical block on the matrix , the block Jacobi method is gained taking again P=D and N=D-A . This iteration is well- identified only when the diametrical blocks are nonsingular. When A is resolved in P*P square blocks the Jacobi method is

$$A_{ii}\, X^{(k+1)}{}_i = b_i - \sum_{j=1}^{p} A_{ij}\, X^{(k)}{}_j \text{ , i=1.....,P ,}$$

Having also solution vector and right side in blocks of size p is referred by $x_i$and b_i individually, finally the outcome, at any step see Fig. (10). Block versions of Jacobi preconditioner can be concluded from a portioning of variables. The block of Jacobi method requires solving p linear systems matrices of A_{ii} . The term of blocking iterative is known an advance optimization technique which can decreasing the pressure on memory bus manifestly. Jacobi block preconditioner need little storage and it is simple to implement. The regular temporal blocking performs dual reuse on tiny block of the computational domain before getting start to the next block see Fig. (11).

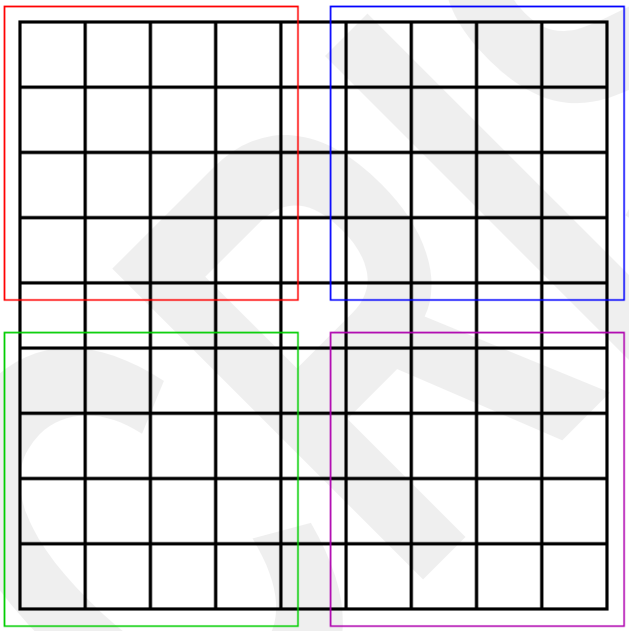


Figure 10 Blocking in 2D method.

2.6 Optimization of Stencil Codes

2.6.1 Single sweep blocking

As shown in Fig. (11) Particularly, with single iteration of time step loop (K), the (i), (j) and (x) loop are represented blocks so that any array points that are approximate each other are gathered to modify. This is permit point exist in code in order to the same iteration of time – loop.

```
for (k = 0; k < timesteps; k++)
      {
         for (kk = 1; kk < nz-1; kk+=tz)
            {
              for (jj = 1; jj < ny-1; jj+=ty)
                   {
                     for (ii = 1; ii < nx-1; ii+=tx)
                       {
                         for (x=1; x<Min(nz-1,kk+tz);x++)
                           {
                            for (j=1; j<Min(ny-1,jj+ty);j++)
                              {
                               for (i=1; i<Min(nx-1,ii+tx);i++)
                                 {
                                  Anext[i,j,k] = A0[i,j,k+1] + A0[i,j,k-1]
+A0[i,j+1,k] +      A0[i,j-1,k] + A0[i+1,j,k] + A0[i-1,j,k] - alpha *
A0[i,j,k];
      }
            }
                  }
                    }
                      }
                        }
Sweeping;
                           }
```

Figure 11 The loop k and loop x, i, and j.

2.6.2 Time skewing

This term means that the multiple sweeps of an array jointly portioned into skewed deltoid and parallel piped blocks. The distinction between time skew and single sweep blocking is dwell in the iteration of (k) as shown in Fig.(12) . The iteration of

(k) loop in the skew are inclusive as operation of every computation block. Because re-modeling the constraints in the midst of neighboring a new point via ultimate iterations of the (k) loop if we suppose that the number of iteration are unknown. Every computation block must shift its group of points backward by fixed number of shifting each iteration until it cover the whole array.

```
for (kk=1; kk < nz-1; kk+=tz)
        for (jj = 1; jj < ny - 1; jj+=ty)
             for (ii = 1; ii < nx - 1; ii+=tx)
                 for (k = 0; k < timesteps; k++)
                         for (n=min_z; n<max_z; n++)
                            for (j=min_y; j<max_y; j++)
                                for (i=min_x; i<max_x; i++) {
                                        Anext[i,j,k] = A0[i,j,k+1] + A0[i,j,k-1] +
A0[i,j+1,k] +      A0[i,j-1,k] + A0[i+1,j,k] + A0[i-1,j,k] -alpha * A0[i,j,k];
                                        }
s = A0;
A0 = A1
A1 =s;
```

Figure 12 K loop time steps.

2.6.3 Single sweep parallelization

As shown in the Fig.(13) this strategy multiple threads are employed to guess its iterations, all threads concurring before getting in the second iteration of encirclement (K) loop.

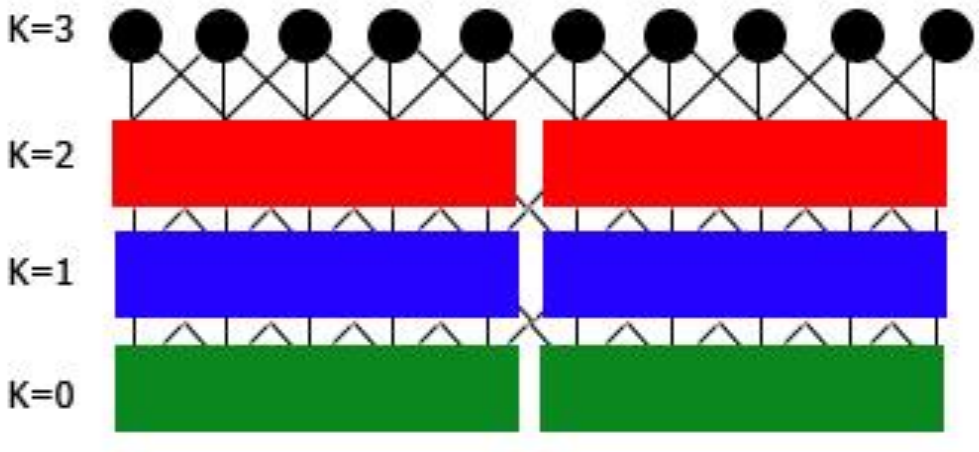


Figure 13 Single sweep parallelization.

In the Fig.(13) the array points that belong to the same block are joined together. The computation blocks that have the same color in this figure are guessed by varied threads.

2.6.4 Pipeline parallelization

This strategy parallelize the external (KK) loop by obviously spawning new threads to guess fixed zone of its iteration in parallel. To be sure the guessing's is right, before guessing any computation block, every thread obviously concurs with the others and wait for all pre-request computation block have been stopped as shown in Fig.(14). After guessing every block, each thread concur with its neighbor threads to enable guessing of following blocks. As a result of this concurring scheme, every computation block is guessed immediately when it has been ready and neighboring computation block are guessed by varied threads in pipelined style.

Figure 14 Pipeline parallelization.

The array points which are closed each other in the same block are gathered in a group, same color in Computation blocks are guessed concur by varied threads.

2.6.5 Wave-Front parallelization

This method is differs from pipeline parallelization by guessing time- skewed blocks by fixing concurring , the time – skewed blocks are added up to a jointly in wave front style. Inclosing specific spatial domain sequential wave-fronts are performs by threads scheduling to multi-core processor chip with joined outer level cache. The Fig. (15) used two dimension Jacobi to shows the efficiency and effectiveness of this method. Also this method can be used in three dimension style.

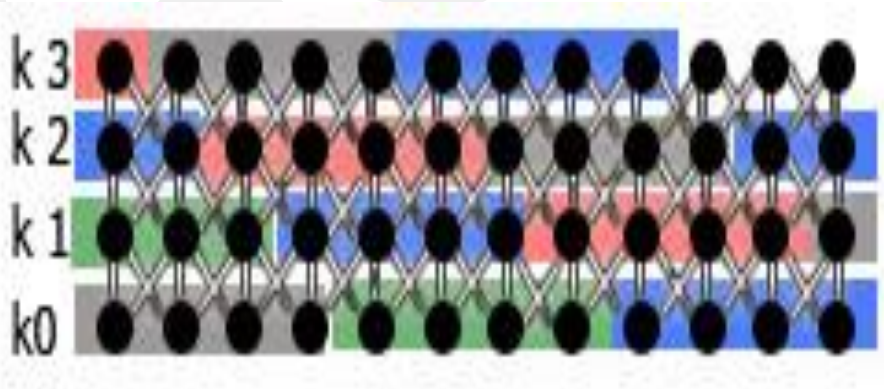


Figure 15 Wave front parallelization**.**

As shown in Fig. (15) The array points which are closed each other in the same block are gathered in a group, same color in computation blocks are guessed concur by varied threads. At the same time single Jacobi carding over full grid is restrictive by memory. Naturally all threads should be concurred after each K iteration to avoid race conditions. When the cache is significant enough to load sequential (i, j) tiles of either x and y, the full scale data can be lessen by one third. Instead of two load and one store for single station update, the requirement still for two (spatially blocked applying containing reading for ownership), then it requires two load (load x and read for ownership on y) and two store (cache line extortion on x and y) in order to make two update.

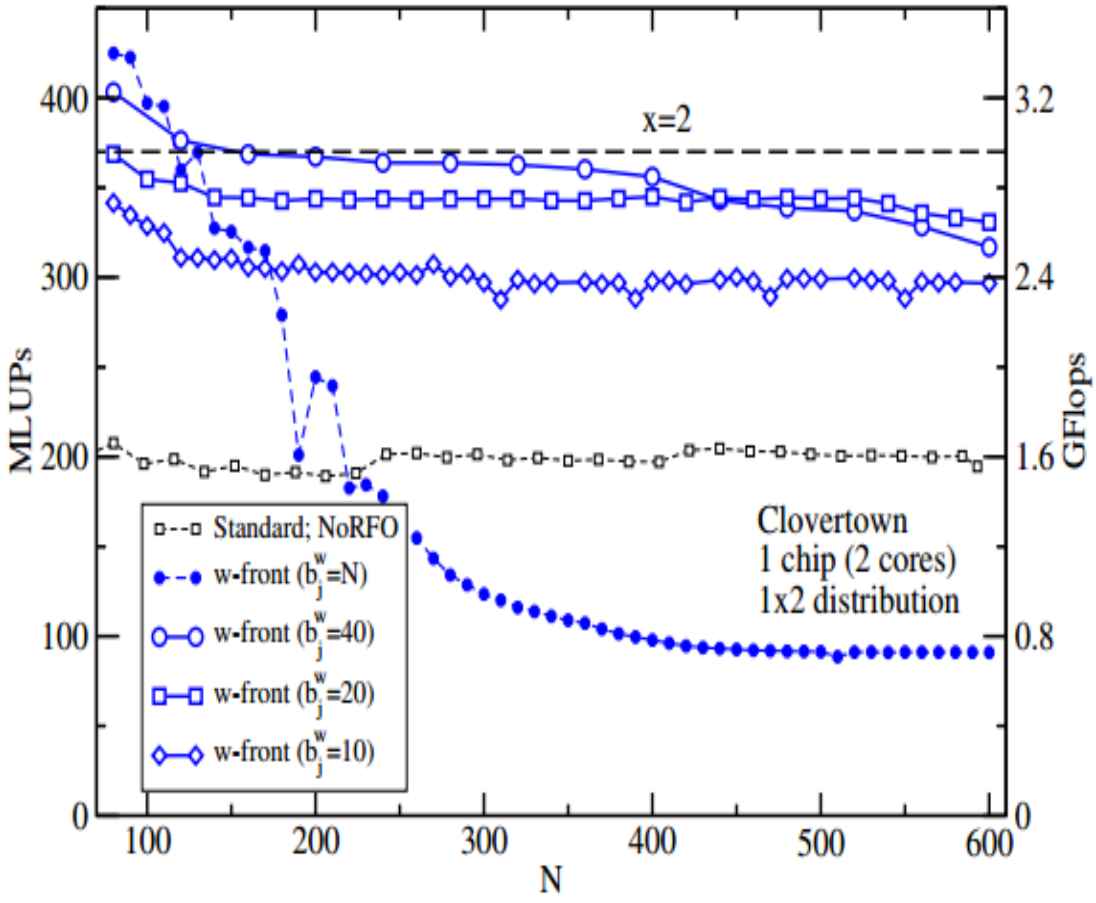


Figure 16 Clovertown system: Single chip (2 cores) performance running two wave-fronts and using different blocking factors in j-direction.

2.7 Three Dimension Jacobi

Three dimension stencil codes become outspread significantly, numerical computation discovery that they have destitute memory conduct with considering microprocessor cache. Optimally software's can lessees cache misses by fetching data in to cache only one time for all iteration accesses. Three dimension partial differential equation solvers have trouble with cache performance, because incoming some data are generally too away apart, needing array elements to brought into cache repeatedly per array update. This problem of cache appeared clearly in three dimension codes more than two dimension codes.

Three dimension kernel as shown in Fig.(17) has 7 stencil points that access to seven columns in three boundary planes at the same time, with the distance $2b^2$, which leading toward (x,y,z+1) and (x,y,z-1) array signal, two whole b*b plans request to stay in cache see Fig.(18), so just three dimension which has size 32*32*Z have ability to take complete advantage of 16KB / L1 cache.

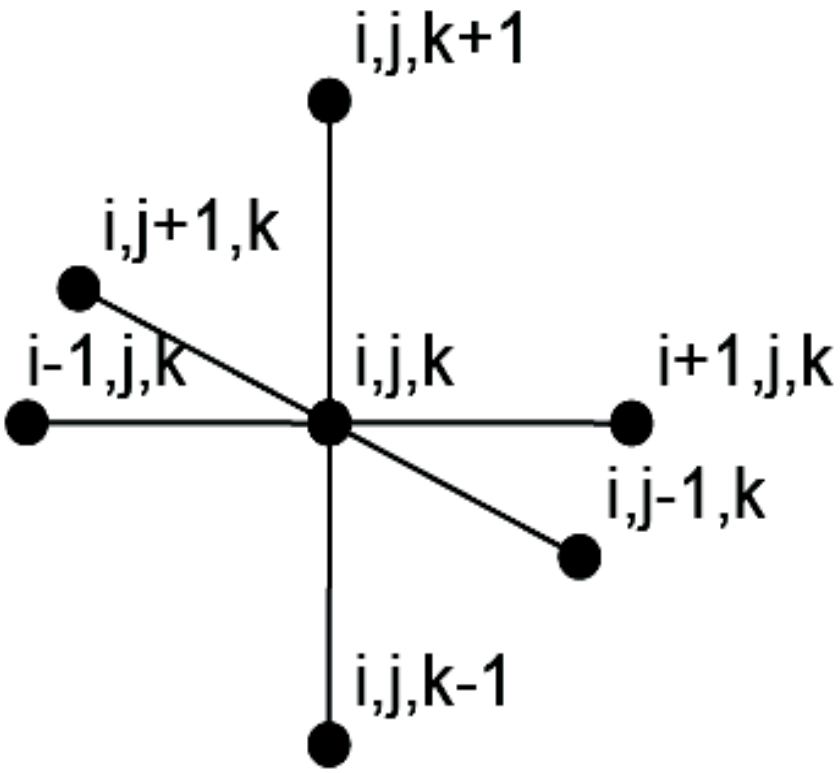


Figure 17 (7) Point of 3 dimension

Also for larger secondary memory L2 cache, the group reusing data wasted for three dimension array more than 358*358*Z. An array will request to bring data into cache twice or more time in order to execute by kernel, and this decreases the performance significantly. From this point the tiling (blocking) which referred in 1.2 section is an optimal solving, it is improving the locality by moving reuses to the same data contiguous in time.

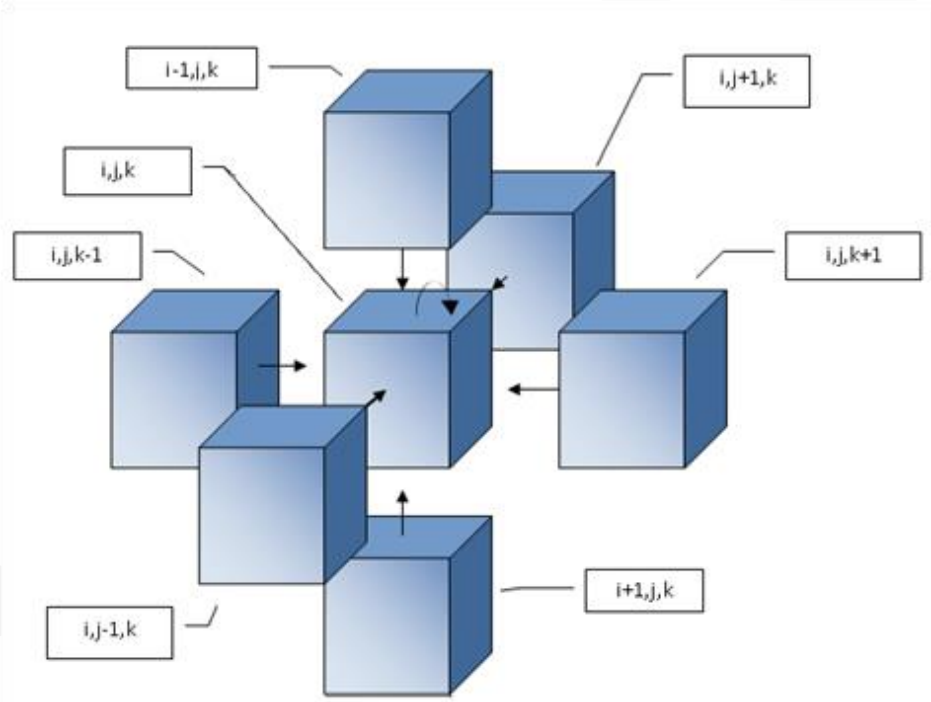


Figure 18 Three dimension Jacobi 7-point.

```
for (int r=1;r<4;r++) //number of iteration
{
        for (int x=1;x<high-1 ;x++)
        {
                for (int y=1;y<width-1 ;y++)
                {
                        for ( int z=r-1;z<length-1;z++)
                        {
                                A1[x][y][z]=(A0[x][y][z]+A0[x-1][y][z]+A0[x+1][y][z]+A0[x]
[y-1][z]+A0[x][y+1][z]+A0[x][y][z+1]+A0[x][y][z-1])/7.0;

                        }
                  }
            }
}
Then the sweeping will be in
for(x=1;x<high-1;x++){
        for(y=1;y<width-1;y++){
                for (z=1;z<=length-1;z++){

                         t=A0[x][y][z];
                        A0[x][y][z]=A1[x][y][z];
                        A1[x][y][z]=t;
                        cout<<A0[x][y][z];

                           }
                      }
                }
// t is the temperature
//width is  x-coordinator
//high is  y- coordinator
// length is z- coordinator
```

Figure 19 The sweeping in three dimension Jacobi.

2.8 Tiling Three Dimension

The main transformation blocks (tiles) as little loops as required to safeguard group reuse in three dimension loop nests. The main idea of blocking is to enable reusing data effectively decreasing size of surface through procedure of the K loop. This is completed by blocking the domestic (X, Y) loops see the Table 3. Firstly x and y are sector-mined to shape block monitoring loops XX and YY. Then XX and YY will be outermost permuted plane. Blocking is very popular method, used enhancing data locality and minimize accessing to main memory and this will increase the speed of computing data. The 27 point three dimension stencil code across three dimension array in methodical style. Each point in array are computed in accumulative weighted according to array point its direct neighbors which are 26 point [11]. The summation is stored in the corresponding array- point in the following three dimension (array field) as shown in the Fig. (19).

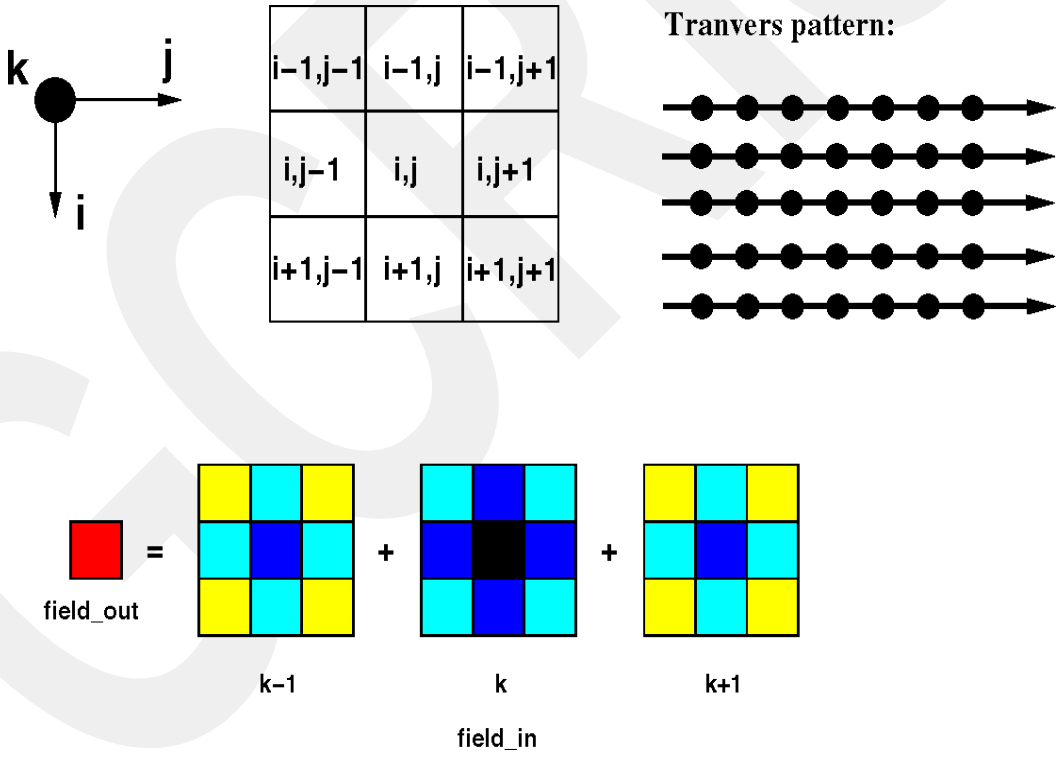


Figure 20 Field-out in three dimension method.

2.9 Gauss-Seidel Method

This is an iterative method , which used in numerical linear algebra , it's also called Liebmann method or method of successive displacement, this method which named by Carl Friedrich Gauss and Philip Ludwing Von Seidel, who were German mathematicians. It has an ability to applying any array with non-zero ingredient on diametric. Convergence which thesis deal with in Jacobi method is the main element in this method warranted if an array "Either diametric dominant or symmetric and positive" [12].

If we suppose that the square linear system is n with unknown x:

$$AX=b;$$

It is releasing by iteration in:

$$L * x^{(k+1)}=b-Ux^{(k)}$$

Where the matrix A is resolved by lower triangle as showed in Fig.(20)

$$A = \begin{matrix} a_{11} & a_{12} & a_{1n} \\ a_{21} & a_{22} & a_{2n} \\ a_{n1} & a_{n2} & a_{nn} \end{matrix} , \quad x=\begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix} , \quad b=\begin{matrix} b_1 \\ b_2 \\ b_3 \end{matrix} ,$$

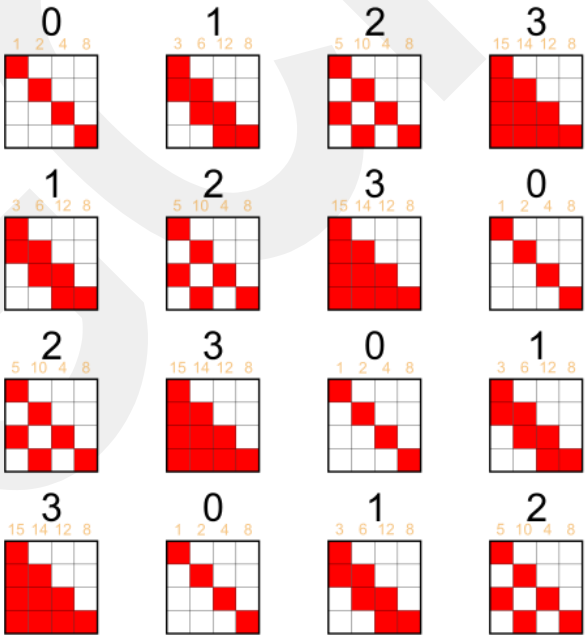


Figure 21 The lower triangle.

Grid	size of one row	fit in cache	size of one plane	fit in cache	size of 3D grid	FIT in cache
16x16x16	128 byte	primary	2 kbyte	primary	32 kbyte	primary
32x32x32	256 byte	primary	8 kbyte	primary	256 kbyte	secondary
40x40x40	320 byte	primary	13 kbyte	primary	512 kbyte	secondary
64x64x64	512 byte	primary	32 kbyte	primary	2 Mbyte	secondary
100x100x100	800 byte	primary	80 kbyte	secondary	8 MByte	memory
128x128x128	1 kbyte	primary	128 kbyte	secondary	16 Mbyte	memory
256x256x256	2 kbyte	primary	512 kbyte	secondary	128 Mbyte	memory
512x512x512	4 kbyte	primary	2 Mbyte	secondary	1 Gbyte	memory
1024x1024x1024	8 kbyte	primary	8 Mbyte	memory	16 Gbyte	memory

Table 3. The Location of Three Dimension Arrays in Memory According to Size

And after that, the resolution of A into its lower triangle component, and its upper triangle content accurately given by :

A=L*+U , $L* = \begin{matrix} a_{11} & 0 \dots \dots & 0 \\ a_{21} & a_{22} \dots \dots & 0 \\ a_{n1} & a_{n2} & a_{nn} \end{matrix}$, U= $\begin{matrix} 0 & a_{11} & a_{1n} \\ 0 & 0 & a_{2n} \\ 0 & 0 & 0 \end{matrix}$,

Then the Linear system could written by:

L*x= b-Ux ,

Then now Gauss Seidel solving the left side of expression of x depending on preceding element of right side

$x^{(k+1)} = L^{-1}_{*}\left(b - Ux^{(k)}\right)$.

In general, by getting the benefits of triangular from of L_*, the value of $x^{(k+1)}$ can be computed successively using forward substitution:

$x^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij} x^{(k+1)}{}_j - \sum_{j<i} a_{ij} x^{(k)}{}_j \right)$ i,j = 1,2,……..n

.

The value- wise for Gauss Seidel very similar to Jacobi method, the $x^{(k+1)}{}_j$, uses just the element $x^{(k+1)}$, which is previously have been computed. And just the value of $x^{(k)}$ that have not so far to be proceeded to iteration k+1. This means that the Gauss Seidel is unlike Jacobi method only in storage Fig. (21). The large advantages of Gauss Seidel is that is the vector demanded as elements could be overwritten when they are computed. However Gauss Seidel can be similar to (SOR Successive Over – Relaxation) if the $w = 1$.

2.10 Amendment Jacobi

Jacobi iterative is math strategy science 169 years-old. This relic from long before super computer, is widely outcast nowadays as so slow comparing with the other methods. Before few month ago from writing this thesis Xiang. Y and Mittal.R, present a strategy that make Jacobi iterative faster about 100 time than classic Jacobi iterative, when they used a fixed difference parataxis of elliptical equations on large grid. The method preserves the main naivety of classical Jacobi method and it is based on a scheduling of over-and under relaxation by mathematical term containing a maximization of convergence proportion are derived and ideal scheme are identified. The convergence ratio prophesy from the testing is validated by numerical proof. The essential accelerating of the Jacobi method approved current method has the potentiality to significant accelerating large-range imitation in computer machine, additionally using this technique in elliptic equation [13].

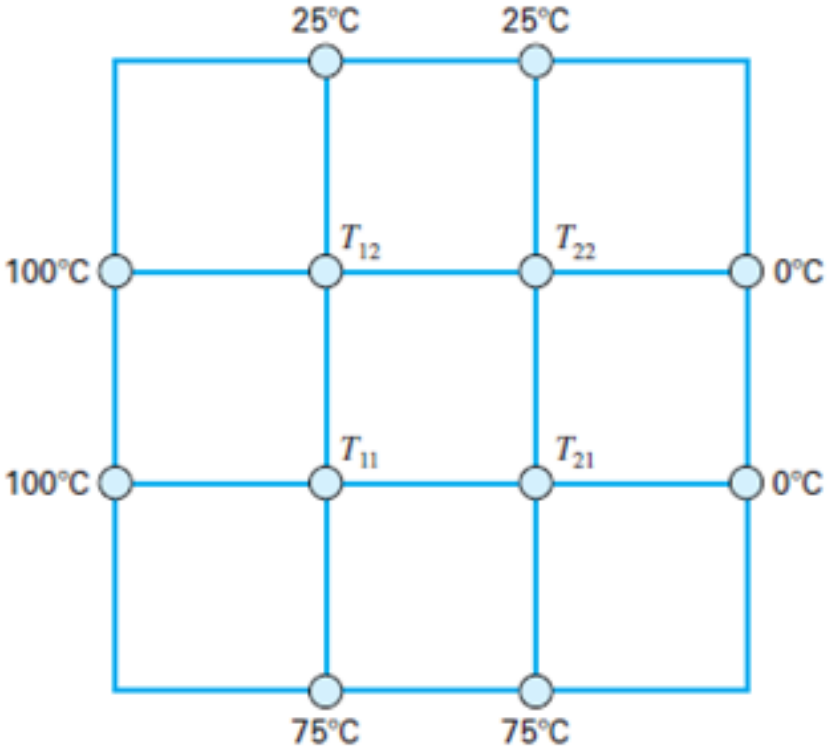


Figure 22 Solving temperatures by Gauss Seidel Method

CHAPTER 3

PRELIMINARY EXPERIMENT

3.1 Open-MP in C++ Language

Open-MP is an application programming interface (API) for multi-core parallelization Consisting **Source code directives, Functions and Environment variables**. The advantage of this (API) is too much for example, easy for using, incremental parallelization, flexible (loop-level or coarse-grain) and portable (it works on SMP machine too). Actually there is a disadvantage for this (API) and that disadvantage is it work only with shared memory systems and shared memory system is single address space for all processors shared memory system also called Symmetric Multiprocessing (SMP) as showed in Fig.(22). The goal of Open-MP is distributing works among threads, and this thesis will discuss two method in this section, first is loop level which specified the loops are parallelized and this method is used by automatic parallelizing tools. The second one is parallel region and also known as coarse-grained usually used in message passing (MPI) see Fig. (23)A&B. The cause of using Open-MP in this thesis is that it is most high level parallel

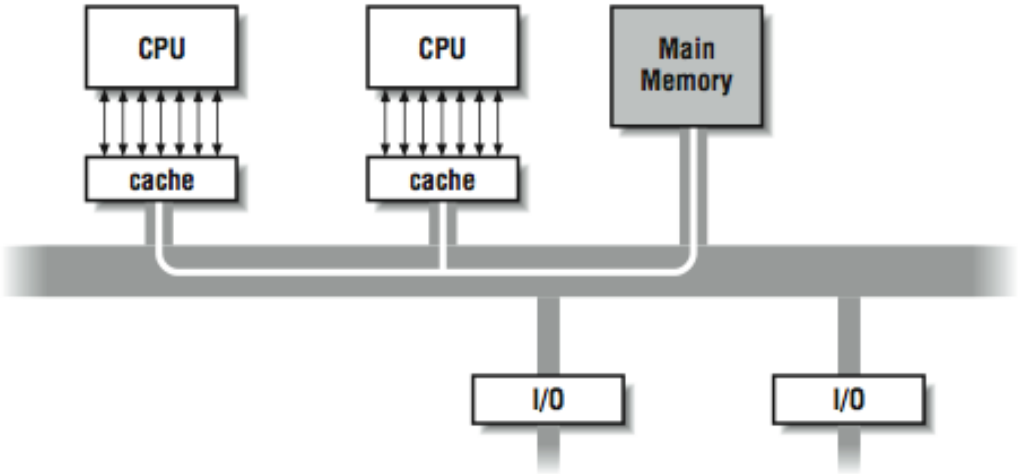
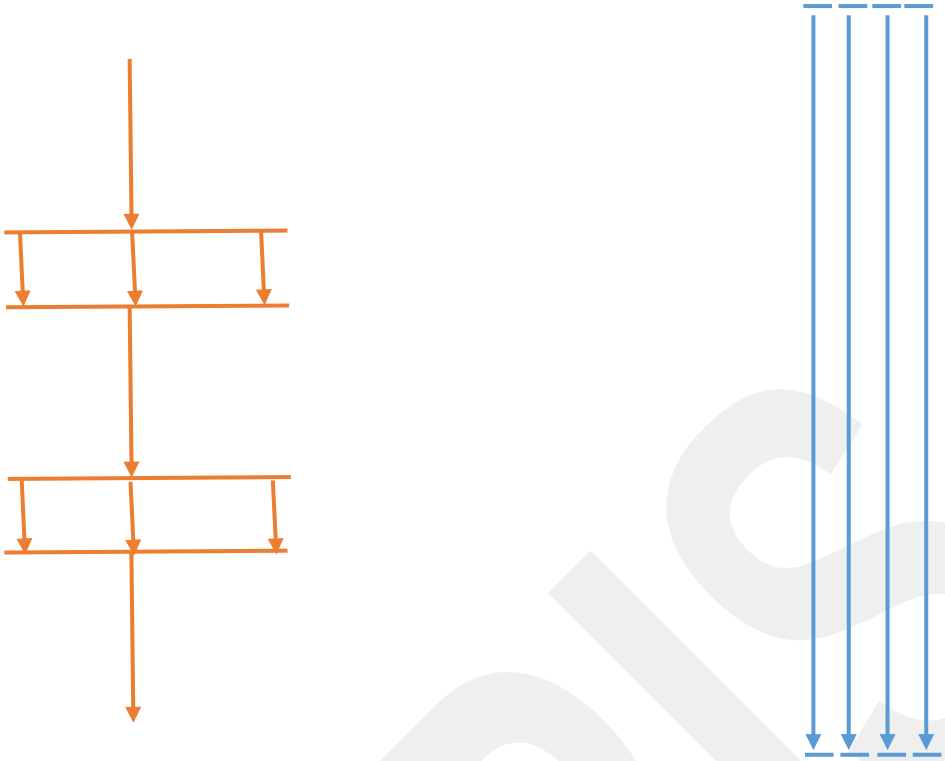


Figure 23 Symmetric multiprocessing

language. It is designed for three general purpose language: C, C++ and FORTRAN language. [14] The open-MP Architecture Review Board (ARB) broadcast its essential API in Fortran language v1.0 in 1997, after one year they make projection for C/C++ standard, in 2000th year open-MP shared with Fortran v2.0 with many specifications, in 2002 year the specification shifted to C/C++ standard, in 2005 the joined the specification of Fortran and C/C++ in version 2.5. In May 2008 they involved new characteristics to open-MP in FORTRAN and C/C++ about task and task construct known as version 3.0. The last update appeared in July 2013 by publishing version 4.0 and putting new feature inside such as support accelerate atomic, error handling, task extensions, user defined reductions, SIMD support, and Fortran 2003 support. Open-MP is accomplishment of multithreading, a parallelizing style which by master thread embranchment fixed number of slave threads and the system separates among them. As shown in Fig. (24), these thread then executed simultaneously, with the runtime medium allocating threads to different processors. The part of code that is meant to execute in parallel is marked accordingly, with a preprocessor instructive that will a reason the thread be formed before the part is run. There is an ID number for each thread linked to it which can produced using function called (omp_get_thread_num ()). The ID of thread should be integer and ID of 0 should included in the master of thread. The thread combined back into the master thread that carry on towards the end of program. By the way, each threads run the parallelized part of code separately. Work participation structure could involve dividing task among the threads, so each thread runs its allocated section of code. Using open-MP both task parallelism and data parallelism carried out.

According on usage the runtime medium allocates threads to processors, such as machine factors and another factors. The runtime medium can specify the number of threads according to medium variables, or it can do so using function in codes. In C/C++ language using function in open-MP are involved header file (omp.h).



(A) Loop-Level **(B) Parallel Region**

Figure 24 (A and B) Loop-Level and Parallel Region systems.

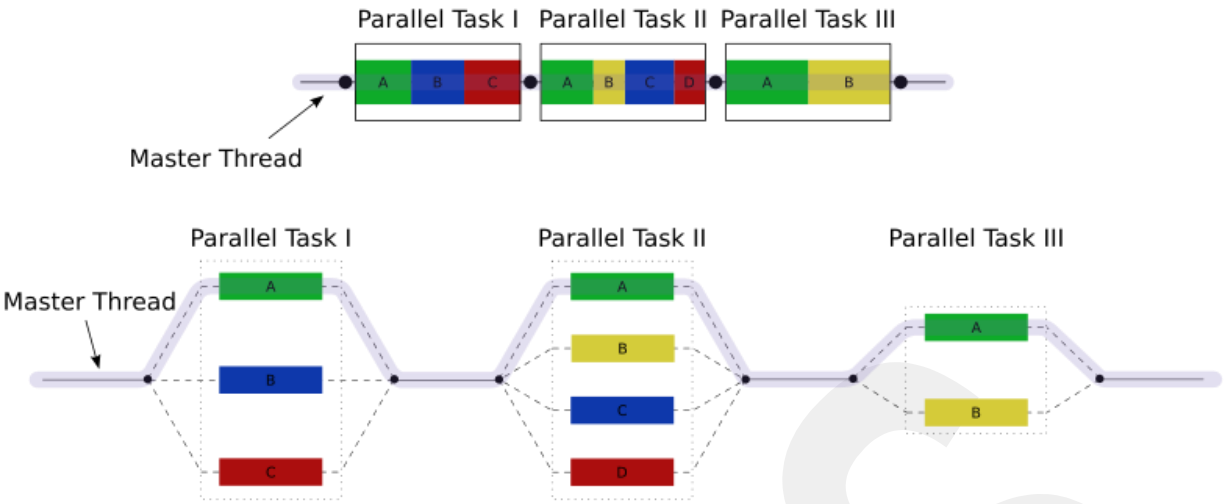


Figure 25 An illustration of multithreading. Where the master thread forks off a number of threads which execute blocks of code in parallel.

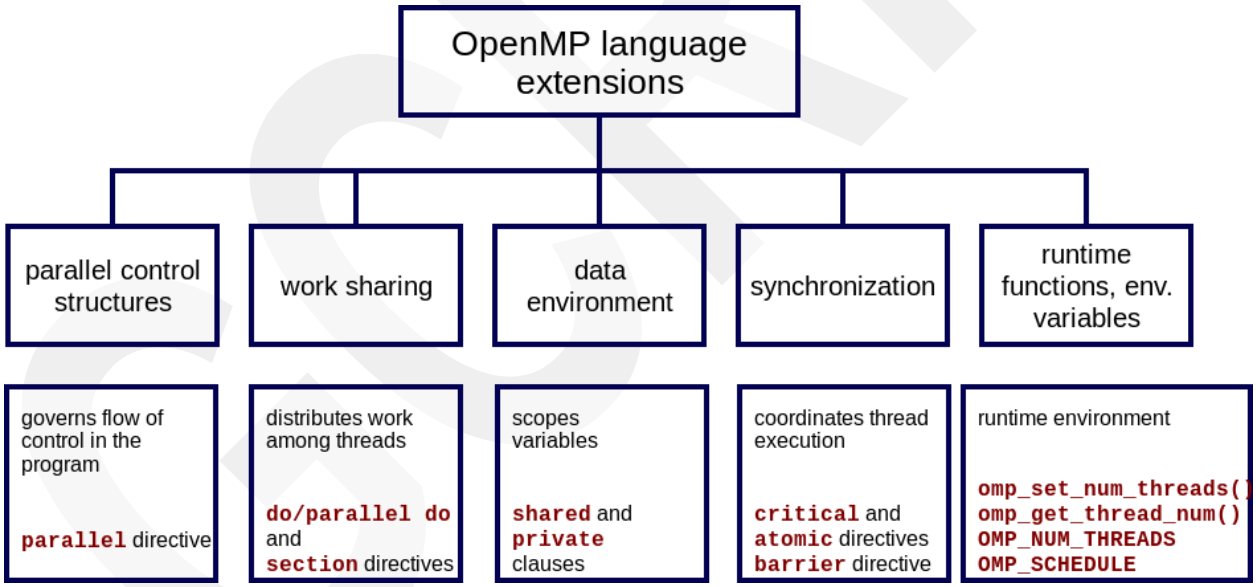


Figure 26 The chart of open-MP constructs.

3.2 Open-MP Creation

As noted in section 3.1 the open-MP is API so the creation of thread caused by the core elements, like another operations such as workload distributions, data medium management, thread concurrence, user level runtime and medium variables see Fig.(25).

Before start explaining the creation of open-MP there are some rules for writing the reader should takes it in to account in this chapter :

1. The codes start by (+), and finish by (+).
2. The codes written in *Italic*.
3. The explanation of codes will proceeded by //.
4. The codes given is not whole codes in this thesis project thesis owner keep the important parts of codes for himself, and some codes given is to explain the open-MP publically(not used in thesis project).
5. This section explain the how to apply open-MP step by step and how to use it in optimal usage by closing unnecessary windows services.
6. The thesis worked on Visual studio 2012.
7. The experiment carried out on two computer (Intel I3, Intel I7) will defined the specification in detail below.

After writing the rules should be written, this section start explaining the codes by giving at least an example for the codes, after using the header omp.h, it should programming language like C/C++ should prepared to use open-MP codes and that will be by opening C++.Then opening a program which use the open-MP after printing omp.h on headers section, click on (project)→(properties) . After clicking properties a new windows appear have many options choose the open-MP section and put yes inside the field, that means the open-MP (API) is running in the computer, in this case the cores are opened according to the size of needing.

This problem faces many programmers, who think that there coding are not right or there is some problem in code writing, after opening the open-MP (API) in the computer, they should be check that the all cores are working without (Parking) as showed in the Fig.(26). To avoid this problem in Jacobi

method it can increase number of iteration in order to make a big press to the processor and forced using all core in order to comparing it with other type. For instance if the comparing between two computers actually happened first computer has the following Specifications:

1. Core I3 →4 Cores.
2. Frequency 2.13 GH.
3. RAM → 4GB.
4. System type → 64 bit.

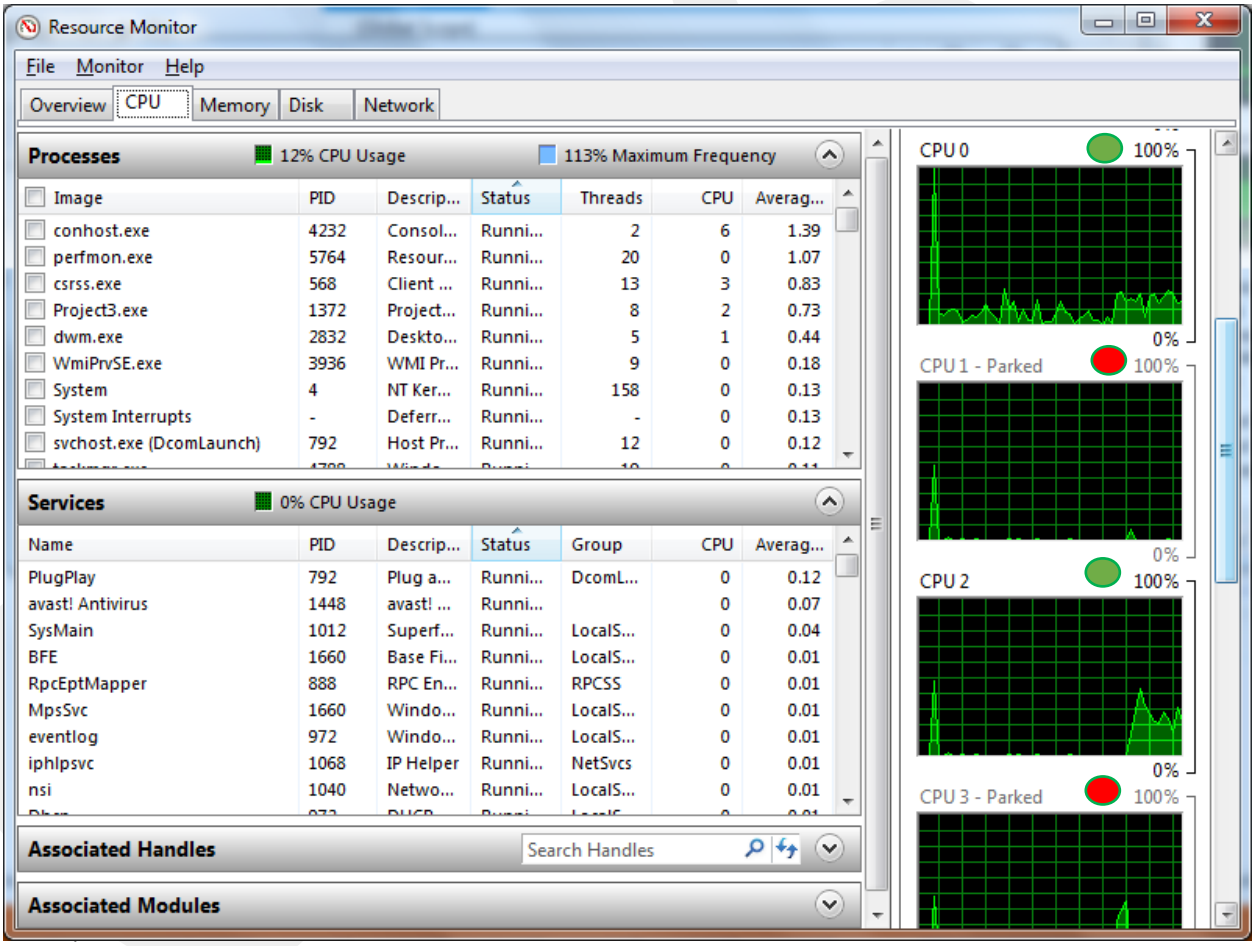


Figure 27 Cores status after running open-MP program

Means that this core is working now

Means that the core is parked

So if the programmer try to solve normal equation even with open-MP (API) in program it will not open all the cores for working because the program not needing for that. And the essential result comparing two computers which over the mentioned of this thesis will be a proofing for this theory. So may be (Intel I3) which have 4 Cores will execute the program faster the (Intel I7) which have 8 Cores, the reason is that if the 4 Cores are opened to running they will be equal but there is many another factors will win the (Intel I3) to (Intel I7). The table [001] shows the comparing of (Intel I3) using all cores which is 4 Cores and (Intel I7) which half number of cores (4 cores) because it is not need to use 8 Core for small computation, And the result is amazing because the result is not benefit of (Intel I7), this table is the Approximately result for Jacobi iteration 5-Points (standard Jacobi, and tiling Jacobi 2 Dimensions) Because the thesis is deal with this two experiments. It is worth mentioning that this table is only to support the theory described in this section and it has not relation with the real result of main experiment and the essence of thesis. At the same time this result is very important in understanding how the cores work.

As seen in Table 4 the small iteration is not gives the real results which we want because it is not use whole efforts of processor, and this means that the difference number of cores between two computer will not appeared unless forced all cores to run. The results show us that whenever the array size is grow the computation result will be more significantly than the smaller array size, the selected green part is the part of Intel I7 overcome to Intel I3, Because it used all cores which are 8 cores to solve the re-computations, while the gray part in table 001 is the part of outdo Intel I3 to Intel I7, as mentioned in this section the overperforming of Intel I3 to Intel I7 is that they are equal in cores in some operation at that time another factors Intervenes in the run time like cache memory, frequency and Ram memory. At measuring to the performance of open-MP in the computer system it take 5.4, while the user performance is 3.2 and the system will take 2.0 from the general performance. These value is differ according many factors such as:

Array Size	Tiling Size	Intel I3 (Running per sec)	Intel I7 (Running per sec)
120*120	No block	35.6 sec	60.1 sec
120*120	3*3	12.3 sec	12.6 sec
120*120	4*4	10.3 sec	11.9 sec
120*120	5*5	9.2 sec	10.3 sec
240*240	No block	147.5 sec	211.4 sec
240*240	3*3	81.4 sec	85.8 sec
240*240	4*4	49.4 sec	52.0 sec
240*240	5*5	44.9 sec	45.4 sec
480*480	No block	641.6 sec	969.5 sec
480*480	3*3	575.8 sec	519.2 sec
480*480	4*4	395.4 sec	330.1 sec
480*480	5*5	273.4 sec	227.0 sec
480*480	10*10	185.5 sec	170.2 sec
960*960	No block	2252.4 sec	2780.2 sec
960*960	5*5	Not responding	1709.4 sec
960*960	10*10	Not responding	995.1 sec

Table 4 The Difference Between Intel I3 and Intel I7 Executing Jacobi Program with 5 Iteration for Every Elements.

1. Behavior: Which memory is accessed by single thread?
2. Open-MP parallelization costs: What's the amount of time are spent to handling the open-MP constructs?
3. Synchronization costs: What's the wasted time to accessing the critical regions?
4. Sequential costs: What's the costs of sequential work that replicated?

5. Load imponderables costs: What's the costs imbalance between the concurrence points?

3.3 Open-MP Usage

The thesis get up on open-MP, most experiments and analyzing and studying in this thesis is deal with open-MP (API), so the explaining this (API) is appropriate for this section at least. This (API) is extend at last 10 years rapidly, many algorithm is developed to deal with this (API), the usage of open-MP is one of factors which focused on in this section. As mentioned in previous section that before writing open-MP program, it must note down the omp.h, in header class. Then opening the open-MP properties inside programming language then start writing the code. There is many codes for open-MP this section will explain it in details. The first code which this thesis deal with is **directive format** this syntax is be formally with open-MP as following:

(+)#pragma omp directive-name [clause [[,] clause]...] new-line(+)
//each directive start with #programa omp

Several directive in open-MP may be form of sequential *#pragma* preprocessing while specified their syntax. The second code which the thesis deal with is **Loop Construct** which written by following sentences:

(+)#pragma omp for [clause [[,] clause] ...] new-line(+)
For-loops
// where clause is one of the following:
Private (list)
Firstprivate (list)
Lastprivate (list)
Reduction (reduction-identifier: list)
Schedule (kind [, chunk_size])
Collapse (n)
Ordered
Nowait

This syntax of loop constructer specified the iteration of one or more loops will be executed in parallel by threads in the group in the context of their tacit tasks. The iteration separated across threads that existed in the group of

running parallel region to the loop region links [15]. The third code is **Section Construct** it is non iteration work-sharing that involved set of structure blocks which distributed among and run by threads in group. Any block structure is run once by one of thread in the group of its tacit tasks.

(+)#pragma omp sections [clause[[,] clause] ...] new-line
{
[#pragma omp section new-line]
structured-block
[#pragma omp section new-line
Structured-block]
...
} (+)

// where clause is one of the following:
private (list)
firstprivate (list)
Lastprivate (list)
Reduction (reduction-identifier: list)
Nowait

The fourth one is Single Construct its construct specifies which deal with structured block that run by only one thread in group, but it should be notes that is not important if run by master or slave threads, in the context of its tacit tasks [16].

(+)#pragma omp single [clause[[,] clause] ...] new-line
Structured-block(+)

//where clause is one of the following:
Private (list)
Firstprivate (list)
Copyprivate (list)
no wait

The last one is Simd Construct which can be run on the loop to mention that the loop could be transformed in to SIMD loop that multiple instruction of loop can be run sequentially using SIMD structure [17].

(+)#pragma omp simd [clause[[,] clause] ...] new-line
For-loops(+)

// where clause is one of the following:
Safelen (length)
Linear (list[:linear-step])
Aligned (list[:alignment])
Private (list)
Lastprivate (list)
Reduction (reduction-identifier: list)
Collapse (n)

The SIMD directive local's limitations on the structure which deal with for loops.

After the explaining of the constructs of codes it should be notes that the optimal run time will occur according the type of operating systems, and an optimal operating system which not have taken efforts from CPU is MS-DOS without using windows, Kernel of Ubuntu in Linux. However, if Windows is used in computing run time or in applying the program we should be careful about the effects of services running in background and close them if possible.

3.4 Summary

This chapter try to proof the whole running of cores needs to be greater pressures on the processor to forced the whole core to works, by given example in this chapter 5 iteration for all matrix element is very small to using all cores so four cores are working comfortingly and parked another four cores as showed in Fig. (26) In 8 cores case (Intel I7) but in 4 cores case it work all cores. Next chapter avoid this problem by increasing the number of iteration rising up an array size and block size, enough to run the 8 cores in (Intel I7) to getting an optimal runtime for the experiments and getting really result to proofing an experiment, by the way this chapter was a preamble to intromission to next chapter.

CHAPTER 4

EXPERIMENT & RESULTS

4.1 Thesis Project

The thesis is focused on many things, some of them are attached to the theory of algorithm which solved by software and the other things are related with architecture of processor and cache memory as mentioned in previous section, this section is discuss the theory of the thesis (mathematic computations, tiling movement on array, the result of this experiment, the ration of distance between this experiments and previous experiment), and on the other hand discuss the software using in this thesis (open-MP codes, the difference between the previous codes and this code, the iteration numbers effective to proof that the core number influence to the computing runtime).

4.2 The Theory Part of Thesis

As mention in the Chapter Two the Jacobi method has many algorithm to gets a result but not all of it is efficient, or in other word the runtime of these execution is not an optimal time, and because this thesis try to accelerate these operation by using all resource of computer like (using all cores, optimal reusing data in cache memory etc..), so it was supposed to adding a new algorithm or at least, enhancing an existed algorithm. This thesis select the second one actually, by enhancing a tiling two dimension Jacobi. As showed in Fig.(27) two dimension Jacobi tiling is work by moving block (tile) step by step, and this operation make delaying in the program, because it is re-compute some elements which was inside in previous tiling. The re-computing operation is made the delaying in the computation because the algorithm forced it to re-compute the element. This operation is varied according the block (tile) size, for instance in the big sizes block, the delay

time will be very large comparison with small size block. To avoid this problem we suggest an enhancement to this algorithm. The enhancement will deal with block movement, and this shifting will not be by column or row for each sweep. The shifting will be by (block-1) this mean it will shift by block, but it moves backward by one step to sure that the boundary of first block will be involved in the second block because Jacobi is not computed in the boundary see Fig.(28). This operation will avoid to re-computing the previous block elements and this leads to a decrease of work time and keeps more reused data in memory cache.

The ratio of enhancement is different according blocking size and array size, and sometimes the ratio may be one. For example (3*3) block is like this because there is no more element in the block to re-compute, so this experiment takes big block such us (5*5), or (10*10). The difference between the standard stencil code (Jacobi without tiling), and Jacobi tiling by (block-1) are shown in the figure below:

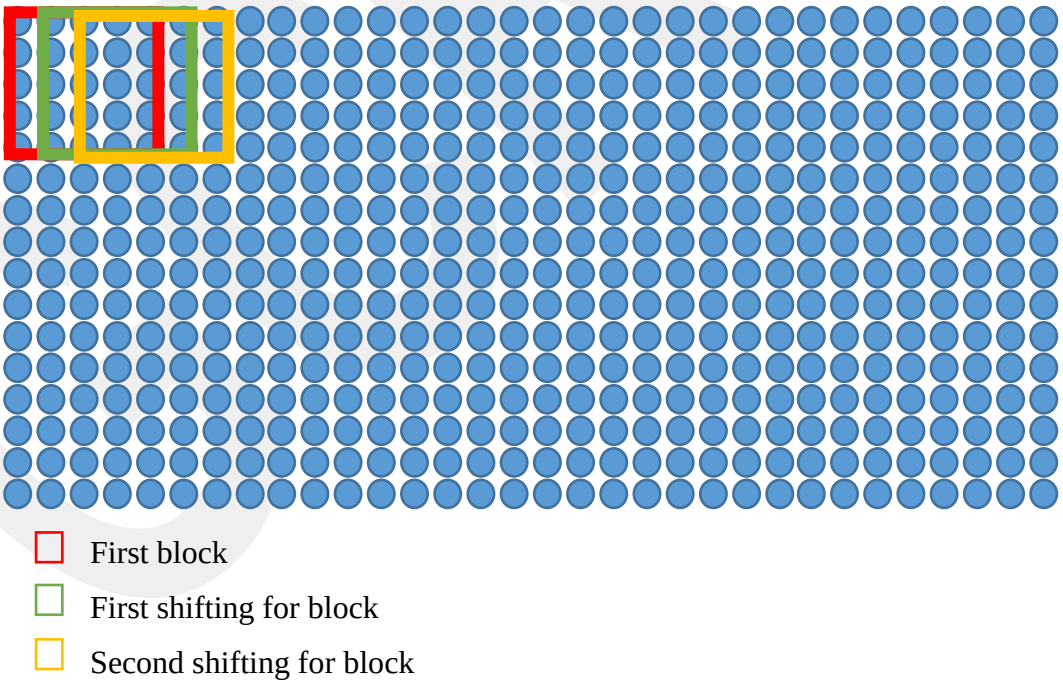


Figure 28 Tiling Jacobi by one step

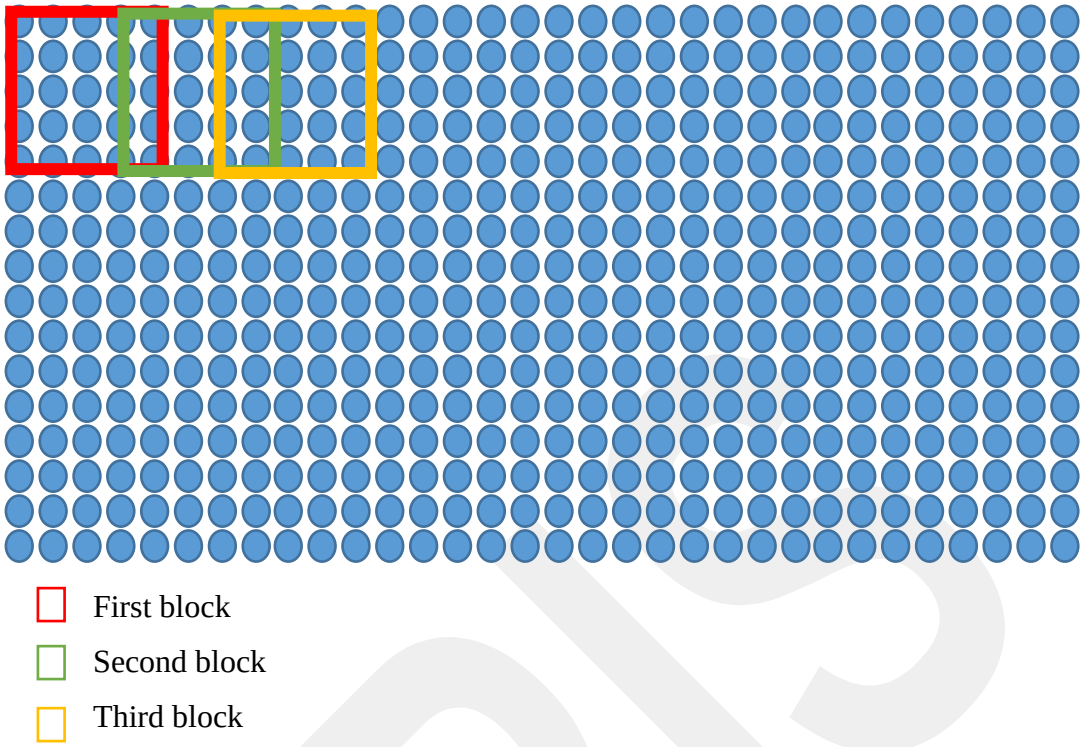


Figure 29 Tiling Jacobi by block-1

4.3 Software of Thesis

This part of the thesis is present the software are used in the program according three pivots: the first one will be standard Jacobi software without tiling, the second pivot will dealing with the simple tiling Jacobi and the last one will explaining by the codes the main experiment of this thesis.

4.3.1 Standard Jacobi without Tiling

This algorithm is mention in theory in the second chapter, but now this section will deal with the software and practically. Actually it is very slow comparing with the other algorithm significantly, sometimes the result of running time comparing with other algorithm is reached to half. This experiment is insure these number practically. First of all run open-MP, then follow the steps as mentioned in previous sections, then:

#include <omp.h> // A header code help to run open-MP codes

And it is important to compute the time so it is important to putting the time header.

#include <ctime> // Time header

In the main part, in order to Fork a team of threads giving them their own copies of variables should use:

#pragma omp parallel private(nthreads, tid) // of course you should defined nthreads and tid as int before this code

Then to get threads number it can writing this code:

```
tid = omp_get_thread_num();
printf("hi, the thread no.is = %d\n", tid);
```

Note: all these steps before starting to write a program.

The program of Standard Jacobi is:

```
#pragma omp parallel //Defines a parallel region, which is code that will be executed by multiple threads in parallel.
                    #pragma omp for //Loop Construct
        for( r=0;r<8;r++) // No of iteration
                {
                for( i=1;i<max-1;i++) // shifting by element, max is refer to an array size
                        {
                         for( j=1;j<max-1;j++) //shifting by elements, max is refer to an array size
                                {
                                  a1[i][j]=(a0[i][j]+a0[i+1][j]+a0[i-1][j]+a0[i][j+1]+a0[i][j-1])/5; // Jacobi computation

                }
                    }
```

```
    #pragma omp parallel
                    #pragma omp for
        for(i=1;i<max-1;i++)
            {
                for(j=1;j<max-1;j++)
```

```
            {
                s=a0[i][j]; //sweeping
        a0[i][j]= a1[i][j];// sweeping
        a1[i][j]=s; //sweeping
       cout<<a0[i][j]<<"     ";

            }
      }
                        }
```

4.3.2 Two Dimension Jacobi Tiling Method

This section deal with the software of normal tiling in Jacobi method, as mentioned in previous sections the disadvantage of this method, but it still faster in computation than Jacobi without tiling. The codes of this method is as explaining below:

```
#pragma omp for
        for(r=0 ; r<8 ;r++) // no. of iteration
            {
                for( i=1; i<max-1; i++)
                {
                    for( j=1; j<max-1; j++)
                    {
                      for( ii=1;ii<max-1;ii+=min-1) // shifting block toward
x, max is represent an array
                            {
                                for(jj=1;jj<max-1;jj+=min-1)// shifting
block toward y, max is represent an array
                                    {

a1[ii][jj]=(a0[ii][jj]+a0[ii+1][jj]+a0[ii-1][jj]+a0[ii][jj+1]+a0[ii][jj-1])/5;
                                    }
                            }
                    }
                }
            }
```

The sweeping will be as showed below:

```
#pragma omp parallel
                    #pragma omp for
          for(ii=1; ii<max-1; ii++)
            {
                for( jj=1; jj<max-1; jj++)
```

```
                {
                        s=a0[ii][jj];
                  a0[ii][jj]= a1[ii][jj];
                  a1[ii][jj]=s;
                  cout<<a0[ii][jj]<<"     ";
        }
                  }
```

4.4 Experiment Software

The experiment focused on the shifting style, the shifting is very important factor to increasing the speed of runtime. This factor changes the program speed significantly, but the code is not changed as the changing of result is a small specification on code which change software's destination and make the software run faster than the previous two methods.

```
#pragma omp parallel
                #pragma omp for
        for(r=0 ; r<8 ;r++) // no. of iteration
            {
                for( i=1; i<max-1; i++)
                {
                        for( j=1; j<max-1; j++)
                        {
                          for( ii=1;ii<max-min+1;ii+=min-1) // shifting by x
block, max is an array, min is a block
                                {
                                        for(jj=1;jj<max-min+1;jj+=min-1) //
shifting by y block, max is an array, min is a block

                                                {

a1[ii][jj]=(a0[ii][jj]+a0[ii+1][jj]+a0[ii-1][jj]+a0[ii][jj+1]+a0[ii][jj-1])/5;
                                                }
                                }
                        }
                }
          }
```

The sweeping will be as below:

```
                  #pragma omp parallel
                #pragma omp for
    for(ii=1; ii<max-1; ii++)
      {
            for( jj=1; jj<max-1; jj++)
              {
                        s=a0[ii][jj];
```

```
            a0[ii][jj]= a1[ii][jj];
            a1[ii][jj]=s;
            cout<<a0[ii][jj]<<"     ";
        }
    }
```

4.5 Experiment Results

The result of this experiment should be divided into two types, the first one should discuss the succession of the experiments by proving overcoming of this experiment to the other two methods, and this become a reality by a table for all this method run time see Table 5 Fig.(29). The second part of this experiment should prove effectiveness of use larger number of cores to show that the runtime be faster the less number of core and this experiment uses two computer (Intel I3, Intel I7), the properties of these two computer is existed in previous sections in this chapter, then the experiment make a table for this part to proof that the Intel I7 working is better than Intel I3, because Intel I7 uses 8 Cores while the second one is uses 4 Cores see Table 6 Fig.(30).

Array size Block size if existed	Standard Jacobi Runtime	2D Jacobi Tiling step by step Runtime	Experiment Jacobi algorithm Runtime
120*120 **5*5**	96.1 Second Not Block case	15.3 Second Block 5*5	11.6 Second Block 5*5
120*120 **10*10**	96.1 second Not block case	13.9 Second Block 10*10	8.4 Second Block 10*10
240*240 **5*5**	392.9 Second Not block case	105.2 Second Block 5*5	50.9 Second Block 5*5
240*240 **10*10**	392.9 Second Not block case	99.8 Second Block 10*10	46.6 Second Block 10*10
480*480 **5*5**	1571 Second Not block case	1175.1 Second Block 5*5	262.6 Second Block 5*5
480*480 **10*10**	1571 Second Not block case	1096.0 second Block10*10	210.1Second Block 10*10

Table 5 Over Performing the Result of Experiment to Other Results.

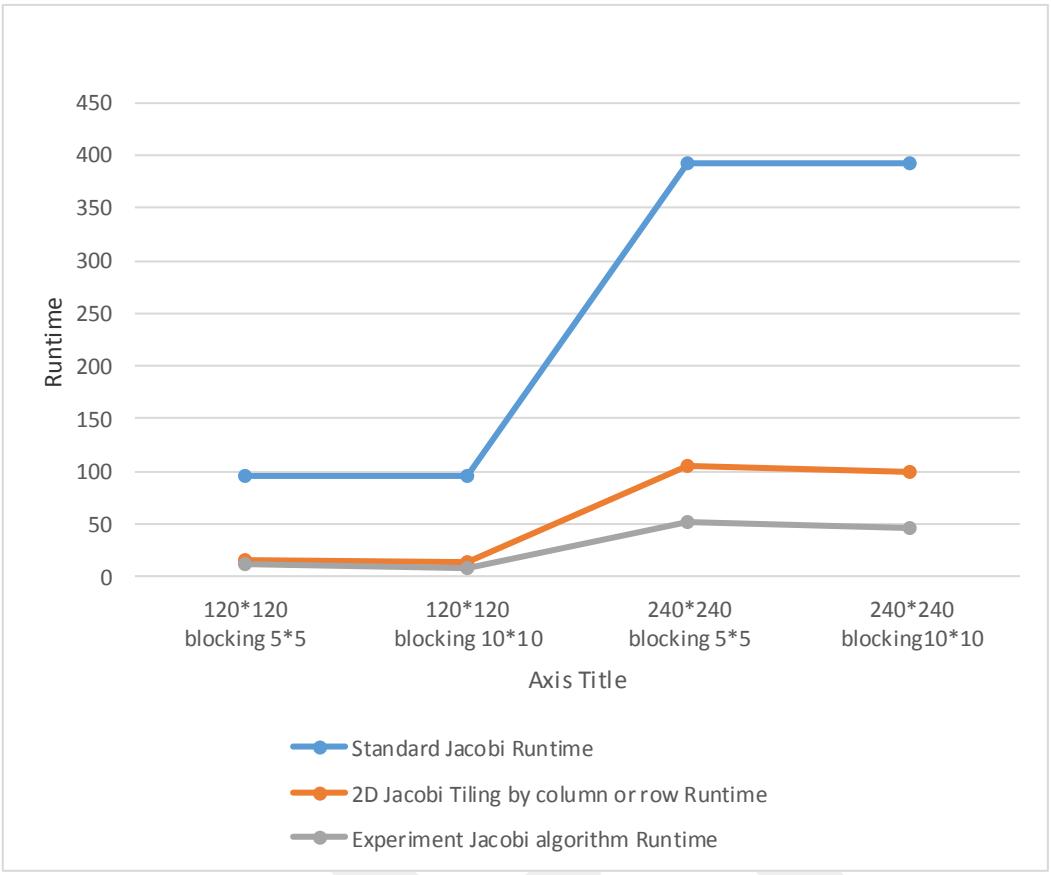


Figure 30 Over performing the runtime of the experiment to other algorithms.

Array size Block size	INTEL I3 Runtime	INTEL I7 Runtime
240*240 3*3	99.8 Second	61.5 Second
240*240 4*4	79.0 Second	52.8 Second
240*240 5*5	70.3 Second	50.9Second
240*240 10*10	58.9 Second	46.6 Second
480*480 3*3	926.3 Second	473.5 Second
480*480 4*4	870.9 Second	287.7 Second
480*480 5*5	569.7 Second	262.6 Second
480*480 10*10	223.5 Second	210.1 Second

Table 6 INTEL I7 Over Performing to INTEL I3 in Runtime on Thises Theory Program.

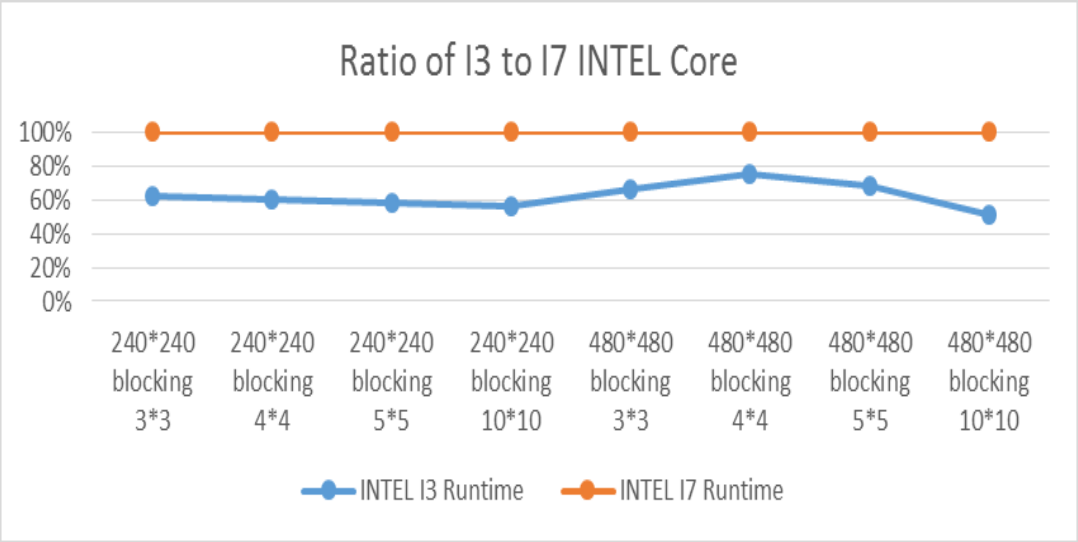


Figure 31 Over Performing of I7 INTEL Processor to I3 INTEL Processor on Thesis Theory.

4.6 Summary

The main aim of this thesis is highlight to two dimension Jacobi method by improving it, to give an optimal runtime by using enhancement method of blocking, this experiment is focused on optimal using of CPU usage by using all cores, and optimal using of cache memory. Then we compare between Intel I7 and Intel I3 processors and prove that the Intel I7 is faster in executing and computing a big array than Intel I3. In the second part of the experiment we try to prove the software effectiveness comparing it to other two algorithm We find that this enhancement ratio is different according to size of an array and size of block.

CHAPTER 5

CONCLUSION & FUTURE WORKS

5.1 Conclusion

This study proves the ability to enhancement of the Jacobi algorithm using stencil codes and bringing data to the cache in aspecific order. We obtained improvements up to %50 to 60% in big arrays as shown in Fig.(31).

These properties lead this study to get successful results. This study focused on three pivots: The first algorithm. Then make a comparison between Intel I7 which has

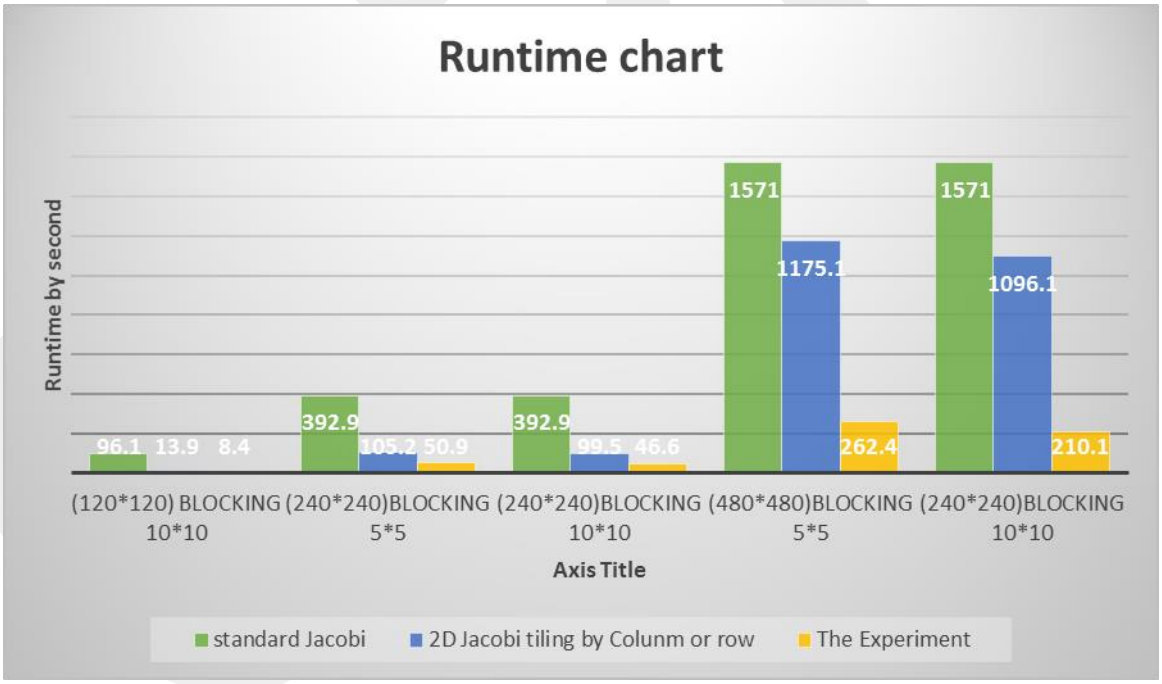


Figure 32 Over performing the experiment runtime to other algorithm.

(8 cores) and Intel I3 which has (4 cores) and getting the result which support our theory as shown in Fig. (32). The third one is the difference between running program with or without open_MP(API) a hidden one in experiment (not mentioned in the results), because it is axiomatic to give a good result although the starting of this thesis was according that result and from this point the experiment started .

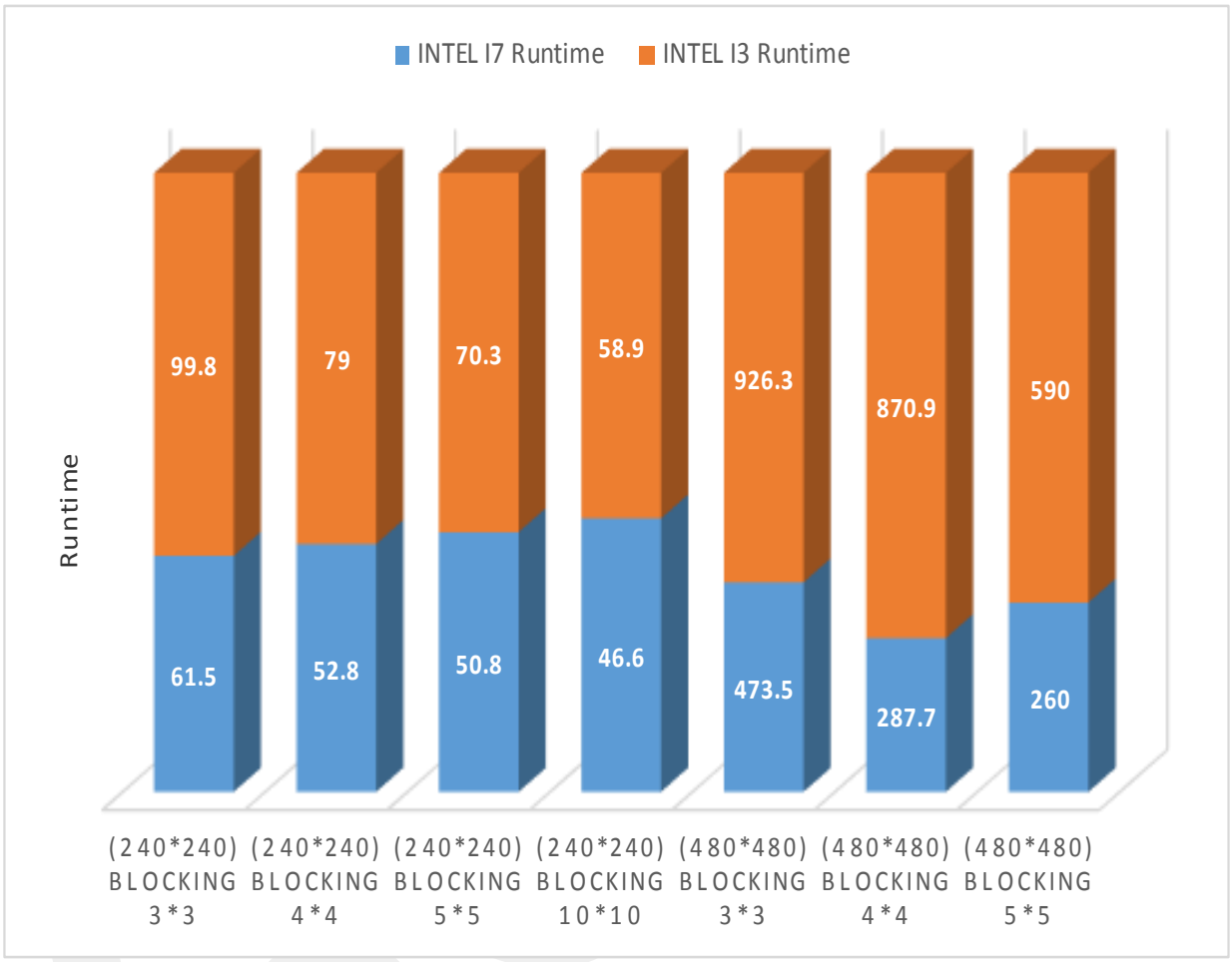


Figure 33 Over Performing the INTEL I7 to INTEL I3 in runtime.on thesis theory.

5.2 Future Works

Since Jacobi method is more flexible than the other algorithm then it can be easy to apply this study which contain an enhancement on 2D Jacobi tiling algorithm to another Jacobi algorithm type for instance 3D Jacobi tiling algorithm. It will be very bestowal in the result more than this experiment, this experiment can be leap to discover new algorithm, and it is giving way to open up prospects to enhancement in various trends, as mentioned in chapter two, there are many methods such as Jacobi itself. Therefore this enhancement can be applied to any method in order to get improved results [18].

REFERENCES

1. **Lülfesmann M. and Kawarabayashi K. I., (2014),** "*Sub-Exponential Graph Coloring Algorithm for Stencil-Based Jacobian Computations",* Journal of Computational Science, vol.5 no.1, pp. 1-11.

2. **Datta K., Kamil S., Williams S., Oliker L., Shalf J. and Yelick K., (2009),** *"Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors",* SIAM Review, vol. 51 no. 1, pp. 129-159.

3. **Christen M., Schenk O., Messmer P., Neufeld E. and Burkhart H., (2008),** *"Accelerating Stencil-Based Computations by Increased Temporal Locality on Modern Multi-and Many-Core Architectures. In High-Performance and Hardware-Aware Computing",* Proceedings of the First International Workshop on New Frontiers in High-performance and Hardware-aware Computing, pp. 47-54.

4. **University of Nice Sophia Antipolice,** http://math.unice.fr/~frapetti/CorsoF/ cours3.pdf, (Data Download Date: 2/11/2014).

5. **Barrett R., Berry M. W., Chan T. F., Demmel J., Donato J., Dongarra J. and Van der Vorst H., (1994),** *"Templates for The Solution of Linear Systems Building Blocks for Iterative Methods",* SIAM, vol. 43, pp. 33.

6. **Rahman S. M. F., Yi Q. and Qasem A. (2011)**, *"Understanding Stencil Code Performance on Multicore Architectures",* In Proceedings of the 8th ACM International Conference on Computing Frontiers, pp. 30.

7. **Datta K., Murphy M., Volkov V., Williams S., Carter J., Oliker L. and Yelick K. (2008),** *"Stencil Computation Optimization and Auto-tuning on State-of-the-Art Multicore Architectures",* In Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, pp. 4.

8. **Li R. and Saad Y., (2013),** *"GPU-Accelerated Preconditioned Iterative Linear Solvers",* The Journal of Supercomputing, vol.63, no.2, pp.443-466.

9. **Krishnamoorthy S., Baskaran M., Bondhugula U., Ramanujam J., Rountev A. and Sadayappan P., (2007, June),** *"Effective Automatic Parallelization of Stencil Computations",* In ACM Sigplan Notices, vol. 42, no. 6, pp. 235-244.

10. **Rutishauser H., (1966),** "*The Jacobi Method for Real Symmetric Matrices",* Numerische Mathematik, vol.9, no.1, pp.1-10.

11. **Marin G. and Mellor-Crummey J., (2008),** "*Pinpointing and Exploiting Opportunities for Enhancing Data Reuse".* In Performance Analysis of Systems and Software, ISPASS 2008, IEEE International Symposium, pp. 115-126.

12. **Huang H. and Chang S., (2003),** "*Gauss-Seidel-Type Multigrid Methods".* Journal of Computational Mathematics-International Edition, vol.21, no.4, pp. 421-434.

13. **Yang X. I. and Mittal R., (2014),** *"Acceleration of The Jacobi Iterative Method by Factors Exceeding 100 Using Scheduled Relaxation".* Journal of Computational Physics, vol. 274, pp. 695-708.

14. **OpenMP,** *http://openmp.org/wp/2014/11/code-challenge-announced-at-sc14/ (Data Download Date: 1/12/2014).*

15. **Dagum L. and Menon R., (1998),** *"OpenMP: An Industry Standard API for Shared-Memory Programming",* Computational Science & Engineering, IEEE, vol.5, no.1, pp. 46-55.

16. **Darlington J., Ghanem M. and To H. W., (1993),** *"Structured Parallel Programming",* In Programming Models for Massively Parallel Computers Proceedings, IEEE, pp. 160-169.

17. **Chapman B., Jost G. and Van Der Pas R., (2008),** *"Using OpenMP: Portable Shared Memory Parallel Programming"*, MIT Press, Vol. 10, pp. 23.

18. **Rivera G. and Tseng C. W., (2000),** *"Tiling Optimizations for 3D Scientific Computations"*, In Supercomputing, ACM/IEEE 2000 Conference, pp. 32-32.

APPENDICES A

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: ALHILALI, AMAR

Date and Place of Birth: 19 May 1986, KERKUK

Marital Status: Single

Phone: +90 507-440 94 36

Email: Amarkhorshid@gmail.com

EDUCATION

Degree	Institution	Year of Graduation
M.Sc.	Çankaya University, Computer Engineering	2015
B.Sc.	Kerkuk Technique University, Software Engineering	2009
High School	Kerkuk High School	2005

WORK EXPERIENCE

Year	Place	Enrollment
2011-2012	Al-QALAM University Computer Engineering Department	Assistant
2009-2011	Al-NASR Company for Reaserch and Study.	Coordinator

FOREIN LANGUAGES
Advanced English, Arabic, Beginner French.

HOBBIES
Wing Chun, Travel, Novel, Swimming, Poetry.