International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-9, Issue-5, September 2021
https://doi.org/10.21276/ijircst.2021.9.5.5
Article ID IRP1208, Pages 32-37
www.ijircst.org

Empirical Investigation of Cloud, Grid and Virtualization Using Compiler Optimization Level for Threads Processes

Solanke Ilesanmi[1], Alomaja Victor Ojumu[2], Ajayi Abiodun Folurera[3], and Ajao Aisha Omorinbola[4]

[1, 2, 3] Department of Computer Technology, Yaba College of Technology, Yaba Lagos, Nigeria
[4] Department of Computer Science, Federal College of Fisheries and Marine Technology, Lagos. Nigeria

Correspondence should be addressed to Solanke Ilesanmi; solankesanmy@gmail.com

ABSTRACT- This research focused on the implementation of Open MP. It considers the parallelization of an application code that simulates the thermal gradient of material in two dimensions. A 'C language' program code called jacobi2d.c that solves a rectangular 2-dimensional heat conductivity problem using Jacobi iterative method was used. The boundary conditions required to compute a temperature distribution for a rectangular 2D problem are: Top 300C, Bottom 500C, Left 400C and Right 900C with a range of problem sizes enter as a run-time parameter to alter the problem sizes and convergence criteria. Also, there were computations and readings for iterations and runtime for four values of M and N which were selected for 01, 02, and 03 optimizations. In Table 1 Readings, four values were selected for each of the iterations. The results show the performance of the runtime as the processor increases from 01-optimization, to 02-optimization and finally to 03-optimization. It can be deduced from the representation that the run time of the values reduces as more resources are allocated to execution through the increase in optimization level. Also, in Table 2 Readings, the runtime decreases as it moves from thread1, thread2, thread3, and thread4, comparing the last values for thread1 which are M is 180, N is 200, and their runtime which is 42.797187001. Also the last values for thread2 which are M is 180, N is 200, their runtime which is 21.772106003. When the two runtimes were compared, it was discovered that there was a decrease in the runtime because the more the thread increases, the more system resources they share such as a processor which may affect their runtime by increasing it.

KEYWORDS- Cloud Computing, Compiler Optimization, Grid, Thread Processes.

I. INTRODUCTION

Cloud computing emerges as a new computing paradigm which aims to provide reliable, customized and QoS guaranteed computing dynamic environments for end-users [11]. A cloud computing is an approach where large scale related capability computer resources and infrastructures are provided in form of services across the Internet to numerous customers. A grid machine can be described as an infrastructure that can be used for solving dynamic problems, such as multi-processes, resources allocation and a centralized control mechanism, using a standard set of protocols and interfaces to deliver significant quality services [3].

The enormous computing resources demand of a process can be solved by a parallel computing implementation specifically developed to work in Grid environments of multiprocessor computing resources. The different parallel computing approaches (intra-node, inter-node and inter-organisations) are not sufficient to address the computing resources demand of such a big problem [1].

II. CLOUD, GRID AND VIRTUALIZATION

This research enhances the context of Grids, Clouds, and Virtualization. Grids computing ensure the delivery of computing power and resources on demand. However, despite the various contributions of active research in the area of grid computing, no viable commercial grid computing provider has emerged. In daily activities, some users or consumers of computational resources will always need to make provision for their own supercomputers that can guarantee speed, timely delivery and multi-processing. The concept of Cloud Computing is not a completely new approach, research has established that there is a relative correlation between Grid Computing, cloud computing and other relevant technologies such as utility computing, cluster computing, and distributed systems in general [5]. The concept of Virtualization is a technology that enables many different Clouds to be integrated. Some researchers focused the definition of grid, cloud and virtualization around on-demand access to computing, data, and services. A grid system comprises of hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [6].

The concept of Cloud computing is an information technology approach that is characterized by ubiquitous access to shared resources and services that can be provided rapidly with minimal interference, via the Internet facilities. In cloud computing, there is sharing of resources such as

software and hardware in order to achieve a coherence services and cost saving approach. In the field of computer science, a thread of execution comprises of the smallest sequence of executable instructions that can be processed independently by a scheduler, which forms part of the operating system. Different operating systems processes threads and processes differently from one another, but mostly a thread is made up of the component of a process. Many threads can executed within a single process, executing concurrently and sharing the same computer resources such as memory, while some processes do not share memory resources or computer resources. Most times, the threads of a process can share its executable code and the values of its variables at any given time [8].

The issue of Virtual Machine (VM) concept dates back to the early 60s, this approach was introduced by IBM as a means to provide concurrent, interactive access to their mainframe computers. A VM was an example of the physical machine and gave users the illusion of accessing the physical machine directly. Virtual Machine was developed and used to enable time-sharing and resource-sharing at the same time on the expensive hardware resources. Virtualization has been helpful in reducing the cost of hardware resources and to improve the overall productivity by accommodating as many users as possible to work on it simultaneously [3]. However, as technology advances and become more integrated, the hardware has become cheaper and affordable and at the same time multiprocessing operating systems has been invented and integrated [2].

III. METHODOLOGY

The development of mobile and portable computing devices has created an enabling environment for minimizing the power consumed by a computer program. In the early development, the limitations of the computer memory have been a limiting factor which hampers the performance of optimizations. Because of all these influence, optimizations do not often produce optimal result, and in fact an optimization sometimes may become an impediment to performance of resources. As a result of these problems, there is a need for compiler optimization and compares of runtime in threads processes in order to determine the processor performance.

The objective of this research is the Implementation of an OpenMP which considers the parallelization of an application code that simulates the thermal gradient of a material in two dimensions using C language program code called jacobi2d.c and compares the results of the optimization and threads processes in order to determine the performance of the runtime as the processor increases.

The study adopted a combination of qualitative and quantitative research methods. It explores the concept of cloud, grid and virtualization. Also utilizes parallelization of an application code which simulates the thermal gradient of a material in two dimensions using a C language program code called **jacobi2d.c** that solves a rectangular 2 dimensional heat conductivity problem using Jacobi iterative method to test the runtime and determine the performance of the processor. This research analysis was conducted using University of Greenwich UK CMS grid machine resources.

IV. DATA PRESENTATION AND ANALYSIS

A. Compiler Optimization

STEP 1

In the step one of this research, there is a modification to the jacobi2d.c code to reflect the following boundary conditions, at top 30C, bottom 50C, left 40C, and at the right 90C. The tolerance was set to 0.0001, the result was set to not printing using 0, and 1 for printing of the boundary sizes. Four different values were selected for M and N for different optimization, which comprises of 01-optimization, 02-optimization and 03-optimization.

- **Reflection of boundary sizes**

Compiler optimization is a process to minimize the time taken to execute a program. The Figure 1 indicates the reflection of boundary sizes, while Figure 2 comprises of compiler, program and runtime parameters. In this step 1, the jacobi2d.c code was modified to:

```
// fix boundary conditions
for (i=1; i <= m; i++) {
        t[i][0] = 40.0;
        t[i][n+1] = 90.0;
}
for (j=1; j <= n; j++) {
        t[0][j] = 30.0;
        t[m+1][j] = 50.0;
```

Fig. 1: Reflection of boundary sizes.

```
si305@student 79 % cd cgv
/home/si305/cgv
si305@student 80 % gcc -O1 jacobii2d.c -o jacobii2d
si305@student 81 % jacobii2d 220 220 0.0001 0
iter = 39338  difmax = 0.00009999646
The time spent is 25.322168  and char 0
si305@student 82 % jacobii2d
         *** Usage: parta1 <integer n> <integer m> <float tol> <integer turnprin
t>
si305@student 83 % jacobii2d 260 260 0.0001 0
iter = 50275  difmax = 0.00009999393
The time spent is 58.745318  and char 0
si305@student 84 % jacobii2d
         *** Usage: parta1 <integer n> <integer m> <float tol> <integer turnprin
t>
si305@student 85 % jacobii2d 300 300 0.0001 0
iter = 61630  difmax = 0.00009999820
The time spent is 76.422497  and char 0
si305@student 86 % jacobii2d
         *** Usage: parta1 <integer n> <integer m> <float tol> <integer turnprin
t>
si305@student 87 % jacobii2d 400 400 0.0001 0
iter = 90696  difmax = 0.00009999933
The time spent is 197.952488  and char 0
```

Fig. 2: Screen shot for Compiler, program name and runtime parameters

B. Optimization Level

The following computations are the readings of iterations and runtime for the four values of M and N which was selected for 01, 02 and 03 optimizations. Four values were selected for each of the iterations, and the table below shows the four values and the result of the iteration and the runtime for each of the iterations performed.

Compiler: gcc jacobii2d.c –o jacobii2d, Executable name: jacobii2d

Table 1: Readings for 01, 02, 03 optimization Runtime for values M and N

	Value M	Value N	Runtime	Iterations	Difmax
READINGS FOR 01-OPTIMIZATION	220	220	25.322168	39338	0.00009999646
	260	260	58.745318	50275	0.00009999393
	300	300	76.422497	61630	0.00009999820
	400	400	197.952488	90696	0.00009999933
READINGS FOR 02-OPTIMIZATION	**Value M**	**Value N**	**Runtime**	**Iterations**	**Difmax**
	220	220	16.488465	39338	0.00009999646
	260	260	28.701726	50275	0.00009999393
	300	300	46.694705	61630	0.00009999820
	400	400	151.077582	90696	0.00009999933
READINGS FOR 03-OPTIMIZATION	**Value M**	**Value N**	**Runtime**	**Iterations**	**Difmax**
	220	220	16.314877	39338	0.00009999646
	260	260	27.160550	50275	0.00009999393
	300	300	45.176557	61630	0.00009999820
	400	400	130.857994	90696	0.00009999933

In the above table 1, comprises of the readings for 01, 02, 03 optimizations for four selected values of M and N. It was discovered from the readings and the results of the runtime that the runtime decreases as the optimization level increases; this indicates that the processor allocates more resources and thereby increasing the rate of execution runtime. In order to explain this in a more graphical form, a bar chart was used to illustrate the runtime performance.

This bar chart explains the performance of the runtime as the processor increases from 01-optimization, to 02-optimization and finally to 03-optimization. We could deduce from the graphical representation that the run time of the values reduces as more resources are allocated to execution through the increase in optimization level.

From the above table 1, the last runtime for each of the optimization values was selected, which comprises of 01 optimization, 02-optimization and 03-optimization.
For all the last values of the each optimization:

M is 400, N is 400

Computing the runtime for the extracted last values in each optimization, we have:

Table 2: Extracted last values of M, N in each optimization

Optimizations	Runtime
01-optimization	197.952488
02-optimization	151.077582
03-optimization	130.857994

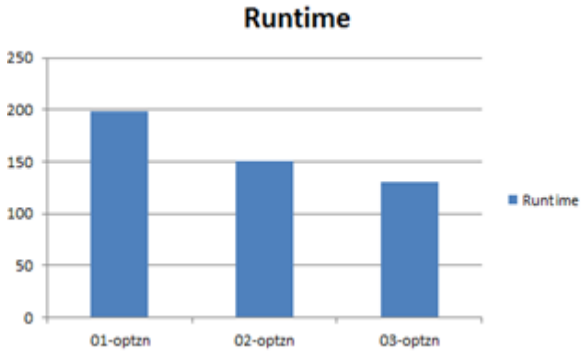

Fig. 3: Runtime Result of 01,02,03 optimizations for value M,N

In summary of the chart analysis above, it is recorded that for values M 400, N 400 with following runtime (197.952488) for 01-optimization, runtime (151.077582) for 02-optimization and runtime (130.857994) for 03-optimization, it was discovered that when the processors are optimized, or when higher optimization are used to run a set of values, the runtime tends to decrease with respect to the increase in optimization. The result of the chart above, can therefore be concluded that the blue lines which signifies the rate of the runtime decreases with the respect to increase in optimization.

C. Open MP Parallel Version of Jacobi Code

STEP 2

In step 2, there is a modification to the application created in step1 to be able to get parallel of the code with openmp using the code (#pragma omp parallel for default (shared) private(i,j)). Timer was also included using the timer code (omp_get_wtime();) so as to determine the parallel runtime of the code, it was also tested on four (4) threads or processors to be able to measure the performance and record the parallel runtime.

```
#pragma omp parallel for default(shared) private(i,j)
    for (i=1; i <= m; i++) {
        for (j=1; j <= n; j++) {
            tnew[i][j] = (t[i-1][j]+t[i+1][j]+t[i][j-1]+t[i][j+1])/4.0;

#pragma omp parallel for default(shared) private(i,j,diff)
    for (i=1; i <= m; i++) {
        for (j=1; j <= n; j++) {
            diff = fabs(tnew[i][j]-t[i][j]);
            if (diff > difmax) {
                difmax = diff;
```

Fig. 4: Openmp Parallel code

D. Threads Level

Compiler:gcc-fopenmp jacobiopenmp.c -o jacobiopmmp, Executable name: jacobiopenmp

V. DISCUSSION OF FINDINGS

The table 2 above shows the various readings for value M and N for about four (4) threads, the readings consists of the values, runtime, iterations and Difmax for each of the value used. The same set of values was used for all the four threads in order to make comparison between the values, especially their runtime. Based on the readings taken from the four (4) threads, the following were discovered:

- The runtime decreases as it moves from thread1, thread2, thread3 and thread4, comparing the last values for thread1 which are M is 180,N is 200, and their runtime which is 42.797187001 . Also the last values for thread2 which are M is 180, N is 200, their runtime which is 21.772106003. When the two runtimes were compared, it was discovered that there was a decrease in the runtime because the more the thread increases, the more system resources they share such as a processor which may affect their runtime by increasing it.
- The iterations are the same for all the values for thread1, thread2, thread3, thread4.
- The Difmax are also the same for thread1, thread2, thread3, thread4.

Table 3: Readings for 1,2,3,4 Threads Parallel Runtime for values M and N

	Value M	Value N	Runtime	Iterations	Difmax
READINGS FOR THREAD 1	70	80	1.489187002	7589	0.00009998674
	100	120	5.872882999	14161	0.00009997808
	140	160	18.256341003	23698	0.00009998894
	180	200	42.797187001	34672	0.00009999279
READINGS FOR THREAD 2	**Value M**	**Value N**	**Runtime**	**Iterations**	**Difmax**
	70	80	0.807676002	7589	0.00009998674
	100	120	3.106940001	14161	0.00009997808
	140	160	9.461168002	23698	0.00009998894
	180	200	21.772106003	34672	0.00009999279
READINGS FOR THREAD 3	**Value M**	**Value N**	**Runtime**	**Iterations**	**Difmax**
	70	80	0.742388003	7589	0.00009998674
	100	120	2.592556998	14161	0.00009997808
	140	160	7.853976000	23698	0.00009998894
	180	200	16.931387000	34672	0.00009999279
READINGS FOR THREAD 4	**Value M**	**Value N**	**Runtime**	**Iterations**	**Difmax**
	70	80	0.576768998	7589	0.00009998674
	100	120	1.973710999	14161	0.00009997808
	140	160	5.704780001	23698	0.00009998894
	180	200	14.126476999	34672	0.00009999279

In addition to the results of the readings in above table 3.
M is 180, N is 200

Table 4: Extracted last values of M and N in each thread

THREADS	RUNTIME
01-Thread	42.797187001
02-Thread	21.772106003
03-Thread	16.931387000
04-Thread	14.126476999

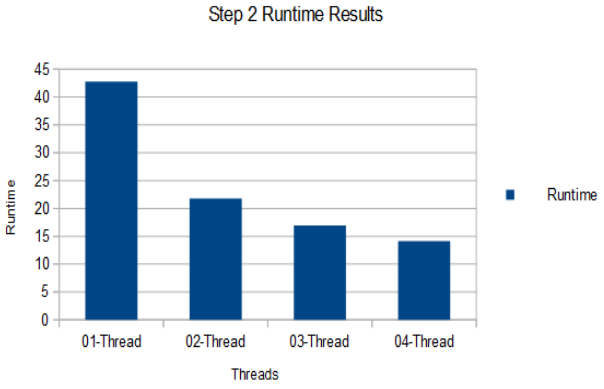

Fig. 5: Step 2 Runtime results

In Figure 5 interprets the runtime for each thread. For the Figure 5, the last values of each thread were used for thread01, thread02, thread 03 and thread04. The last values of all the threads are:
This graph (Figure 5) shows additional information on the decrease in runtime as the number of threads increases.

VI. CONCLUSION

This research work demonstrates using Open MP to parallelize a practical application. It shows how parallel performance tuning using Open MP as well as compiler optimizations can be used to achieve improved performance. The parallelization of an application code which simulates the thermal gradient of a material in two dimensions was used. The research work explains the performance of the runtime as the processor increases from 01-optimization, to 02-optimization and finally to 03-optimization. We could deduce from the result representation that the run time of the values reduces as more resources are allocated to execution through the increase in optimization level. Also, it was discovered that there was a decrease in the runtime because the more the thread increases, the more system resources they share such as a processor which may affect their runtime by increasing the run time.

CONFLICTS OF INTEREST

The authors declare that they have no conflicts of interest.

CONTRIBUTIONS TO KNOWLEDGE

In contributing to knowledge, the study has demonstrated how parallel performance and compiler optimization can be used to achieve improved processor performance, the following recommendations are suggested: Better performance can be achieved when grid machine resources are not frozen or overloaded by multiple users who are running parallel code at the same time. Also, it is important to note that for a swarm or large user grid platform, provisions of robust grid machine resources is necessary in order to avoid freeze or overload.

REFERENCES

[1] Aparício, G., Blanquer, I., & Hernández, V. (2006, June). A parallel implementation of the k nearest neighbours classifier in three levels: Threads, mpi processes and the grid. In International Conference on High Performance Computing for Computational Science (pp. 225-235). Springer, Berlin, Heidelberg.

[2] Chiueh, S. N. T. C., & Brook, S. (2005). A survey on virtualization technologies. Rpe Report, 142.

[3] Cafaro, M., & Aloisio, G. (2011). Grids, clouds, and virtualization. In Grids, Clouds and Virtualization (pp. 1-21). Springer, London.

[4] Dagum, L., & Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. Computational Science & Engineering, IEEE, 5(1), 46-55.

[5] Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008, November). Cloud computing and grid computing 360-degree compared. In Grid Computing Environments Workshop, 2008. GCE'08 (pp. 1-10). Ieee

[6] Foster, I., & Kesselman, C. (Eds.). (2003). The Grid 2: Blueprint for a new computing infrastructure. Elsevier.

[7] Lombardi, F., & Di Pietro, R. (2011). Secure virtualization for cloud computing. Journal of Network and Computer Applications, 34(4), 1113-1122.

[8] Lamport, L. (1979). How to make a multiprocessor computer that correctly executes multiprocess programm. IEEE transactions on computers, (9), 690-691.

[9] Mc Evoy, G. V., & Schulze, B. (2008, December). Using clouds to address grid limitations. In Proceedings of the 6th international workshop on Middleware for grid computing (p. 11). ACM.

[10] Smith, R. (2009). Computing in the cloud. Research-Technology Management, 52(5), 65-68.

[11] Wang, L., Tao, J., Kunze, M., Castellanos, A. C., Kramer, D., & Karl, W. (2008, September). Scientific cloud computing: Early definition and experience. In High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on (pp. 825-830). IEEE.

[12] Youseff, L., Butrico, M., & Da Silva, D. (2008, November). Toward a unified ontology of cloud computing. In Grid Computing Environments Workshop, 2008. GCE'08 (pp. 1-10). IEEE

ABOUT THE AUTHORS

Dr. Solanke Ilesanmi is currently working as a Lecturer in the Department of Computer Technology, Yaba College of Technology. He holds a Bsc, Msc and Ph.D in Computer Science. His research interest includes Green Computing, Machine Learning, Software Complexity, Artificial Intelligence and Data Mining.

Alomaja Victor Ojumu is currently working as a Lecturer in the Department of Computer Technology, Yaba College of Technology. He holds Professional Certifications, Higher Diploma in MIS, PGD Computer System Networking, Bsc Computer Science, Msc in Computer Science. His research interest includes Software Complexity, HCI, Machine Learning and Networking.

Ajayi Abiodun Folurera is currently working as a Lecturer in the Department of Computer Technology, Yaba College of Technology. She holds a B.Tech in Computer Science and currently running her Masters degree in Computer Science. Her research interest includes, Networking, Machine Learning and Information Systems.

Ajao Aisha Omorinbola is a Lecturer at Federal College of Fisheries and Marine Technology. She holds a Bsc and Msc in Business Information System from Middlesex University UK. Her research interest includes Management Information Systems, E-commerce, BIS.