Performance Analysis of a Parallel Computing Algorithm Developed for Space Weather Simulation

T. A. Fashanu[1], Felix Ale[1], O. A. Agboola[2], O. Ibidapo-Obe[1]

[1]Department of Systems Engineering, University of Lagos, Nigeria; [2]Department of Engineering and Space Systems, National Space Research & Development Agency, Abuja, Nigeria
Email: facosoft@yahoo.com

ABSTRACT

This work presents performance analysis of a parallel computing algorithm for deriving solar quiet daily (Sq) variations of geomagnetic field as a proxy of Space weather. The parallel computing hardware platform used involved multi-core processing units. The parallel computing toolbox of MATLAB 2012a were used to develop our parallel algorithm that simulates Sq on eight Intel Intel Xeon E5410 2.33 GHz processors. A large dataset of geomagnetic variations from 64 observatories worldwide (obtained for the year 1996) was pre-processed, analyzed and corrected for non-cyclic variations leading to [366 x 276480] partitioned matrix, representing 101,191,680 measurements, corresponding to International Quiet Day (IQD) standard for the longitude, latitude and the local time of the individual stations. The parallel algorithm was four times faster than the corresponding sequential algorithm under same platform and workload. Consequently, Amdahl and Gustafson's models on speedup performance metric were improved upon for a better optimal result.

Keywords : Parallel Computing, Multicore architecture, parallel slowdown, Geomagnetic field, Space weather

1 INTRODUCTION

The standard evaluation and performance analysis of parallel computing system is a complex task. This is due to factors that determine speedup and parallel slowdown parameters. In order to re-evaluate some intrinsic speed-related factors, an experimental and empirical case study of parallel computation may be more appropriate for generalization and determination of relationship that exist among the benchmark metrics. According to Hwang and Xu [11], parallel computations involve the use of multiple and high-capacity computing resources such as vector processors, storage memories, interconnect and programming techniques to solve and produce optimized results of large and/or complex problems. Such tasks commonly found in financial modeling, scientific and engineering disciplines involve intensive computation in a shortest possible time. Parallel computing is a platform that harmonizes all essential computing resources and components together for seamless operation. It ensures scalability and efficient performance under procedural but stepwise design. It is worthy to note that the ultimate aim of parallel computation is to reduce total cost of ownership and increase performance gain by saving time and cost. [13], [2], [27] and [21] have also proved that parallel computation increases efficiency as well as enhanced performance of a well-developed parallel algorithm could produce optimized results. [14], [19] and [5] extend the platforms and architecture of platforms of parallel computing to include multicores, many-core GPU, Beowulf, grid, cloud and cluster computing. Their variations largely depend on the architectural design and topology of the vector processors, memory models, bus interconnectivity with attendant suitable benchmark parameters and standard procedures for performance analyses.

The aim of this analysis is to evaluate the efficiency of our developed algorithm in terms of runtimes, optimization of results and how well the hardware resources are utilized. The evaluations were used to improve on benchmark models for performance analysis. Execution runtime is the time elapsed from when the first processor starts the execution to when the last processor completes it. Gropp et al [6] and [20] split the total runtime on a parallel system into computation time, communication time and idle time while vector processors operate on large dataset concurrently. Wulf & McKee [26] explained how compiler automatically vectorizes the innermost loops by breaking the work into segments and blocks. The functional units are pipelined to operate on the blocks or loops within a single clock cycle as the memory subsystem is optimized to keep the processors fed at same clock rate. Since the compilers do much of the optimization automatically, the user only needs to resolve complexity and where there are difficulties to the compiler such as vectorization of loops and variable slicing. MPI, compiler directives, OpenMP, pThreads packages and other in-built job schedulers of application generators are among tools for parallelizing programs so as to run on multiple processors.

Gene and Ortega [4] along with Gschwind [7] noted that advancement in multicore technology has paved way for development of efficient parallel algorithms and applications for complex and large numerical problems which are often found in science, engineering and other disciplines of human endeavours Multicore technology includes integration of multiple CPUs on a chip and in-built caches that could reduce contention on main memory for applications with high temporal locality. Caches are faster, smaller memories in the path from processor to main memory that retain recently used data and instructions for faster recall on subsequent access. Task parallelism favours cache design and thereby increases optimization and faster speed of execution. There is a direct relationship between processors performance and efficiency of parallel computing. Although floating-point operations per second is a common benchmark measurement for rating the speed of microprocessors, many experts in computer industry feel that FLOPS measurement is not an effective yardstick because it fails to take into account factors such as the condition under which the microprocessor is running which includes workload and types of operand included in the floating-point operations. This has led to the creation of Standard Performance Evaluation Corporation. Meanwhile, parallel computation with vector processors and optimized programming structure require enhanced methodology for performance evaluation.

Solar quiet daily (Sq) variation of geomagnetic field is a current in the ionosphere responsible for magnetic perturbation on the ground that is measureable by ground-based magnetometers through magnetic components variability. In a study carried out by [22], it was found out that variability of solar quiet at middle latitude resulting from ionospheric current density is a precursor for Sq variation. Although, much work has been done using seasonal, annual, hourly and diurnal geomagnetic dataset, the use of finer and higher time resolution of one-minute data which involve data-intensive computations would provide more useful result and better insight to observation and prediction of Sq variation and subsequently helpful for monitoring Space weather and its effects on Space-borne and ground based technological systems. Analysis of high-performance parallel computation of Sq algorithm using one-minute time resolution of 1996 magnetic dataset from 64 observatories worldwide led to the enhancement of Amdahl's performance analysis model [1].

The paper is structured as follows. In the next section, we describe briefly the background this work. Section 3 shows methods of analysis and Sq parallel algorithm we developed. Section 4 discusses the results obtained and we conclude the paper in Section 5.

2. PERFORMANCE ANALYSIS OF PARALLEL SYSTEMS

Amdahl [1] and Gustafson [8] propounded laws that defined theories and suitable metrics for benchmarking parallel algorithms performance. Gene Myron Amdahl's law [1] specifically uncovers the limits of parallel computing while Gustafson's law [8] sought to redress limitation encountered on fixed workload computation.

Amdahl's [1] law states that the performance improvement to be gained from using high-performance parallel computing mode of execution is limited by the fraction of the time the faster mode can be used. Amdahl's model assumes the problem size does not change with the number of CPUs and wants to solve a fixed-size problem as quickly as possible. The model is also known as speedup, which can be defined as the maximum expected improvement to an overall system when only part of the system, is improved. Woo and Lee [25] as well as other authors have interpreted and evaluated Amdahl [1] and Gustafson's laws [8] on benchmarking parallel computing performance in terms of various computing platforms. Hill and Marty [10] analyzed performance of parallel system on multicore technology and evaluated how scalable a parallel system could be from pessimistic perspective. Xian-He and Chen (2010) reevaluated how scalable a parallel algorithm is by the analysis and application of Amdahl's theorem [1].

2.1 Performance Metrics

The metrics used for evaluating the performance of the parallel architecture and the parallel algorithm are parallel runtime, speedup, efficiency, cost and parallel slowdown factor as follows:

2.11 Speedup is given as:

$$S_p = \frac{T_s}{T_p} \quad or \quad \frac{T_s}{T_1} \tag{1}$$

Where T_s = runtime of the Sq serial program

- T_p = runtime of the Sq parallel algorithm using p processors
- T_1 = runtime using one CPU
- T_s = runtime of the fastest sequential program

Speed of sequential algorithm subjectively depends largely on the methodology of array or matrices manipulation and programming structure. Thus, serial runtime, T_s, must be developed to be the fastest algorithm for a particular problem, on this premise Amdahl's law could be proven right and validated. Therefore, it is rarely assumed that T_1 equals T_s.

The interpretation of Amdahl's Law [1] is that speedup is limited by the fact that not all parts of a code can run completely in parallel. According to Amdahl's laws, equation (2) defines speedup (S_p) as a function of sequential and parallel segments of a program with respect to number processors (*p*) used.

$$S_p = \frac{1}{f + \frac{1-f}{p}} \tag{2}$$

Where the term *f* denotes the fraction of program operations executed sequentially on a single processor and the term *(1 - f)* refers to the fraction of operations done in optimal parallelism with *p* processors.

Generally, algorithm or program code is composed of parallel and serial sections. Obviously, it is impracticable to speed up parallel section absolutely due to bottleneck caused by sequential section. In equation (2), when the number of processors goes to infinity, the code speedup is still limited by *1 / f*.

Amdahl's law indicates that the sequential fraction of code has much effect on speedup and overall efficiency of the processors allocation. This accounts for the need for large and complex problem sizes when employing multiple parallel processors so as to obtain better performance on a parallel computer. Amdahl [1] proved that as the problem size increases the opportunity for parallelism grows and the sequential fraction shrinks.

In addition to Amdahl's law [1], John L. Gustafson [8] found out that complex problems show much better speedup than Amdahl predicted. Gustafson concentrated on computation time instead of the problem size. He explained that increasing number of CPUs to solve bigger problem would provide better results in the same time. From the foregoing, the Amdahl's law was simplified to re-define speedup with respect to computation time as follows:

Computation time on the parallel system, T_p, can be expressed as:

$$T_p = t_s + t_p \tag{3}$$

Where t_s = computation time spent by the sequential section

t_p = computation spent by the parallel section

While computation time on the sequential system, T_s, can be expressed as:

$$Ts = t_s + p * t_p \tag{4}$$

Applying equation (1)

$$S_p = \frac{t_s + p * t_p}{t_s + t_p} \tag{5}$$

Therefore,

$$f = \frac{t_s}{t_s + t_p} \tag{6}$$

$$S_p = f + p(1-f) \tag{7}$$

The Gustafson's law clearly states that the variation of problem is indirectly proportional to sequential part available with respect to execution runtimes. These laws were previously applicable in mainframe, minicomputer and personal computer periods. Experimentally, these laws are still valid for evaluating parallel computing performance in the cluster and multicore technology eras.

Amdahl's equations [1] assume, however, that the computation problem size would not change when running on enhanced machines. This means that the fraction of a program that is parallelizable remains fixed. Gustafson [8] argued that Amdahl's law doesn't do justice to massively parallel machines because they allow computations previously intractable in the given time constraints. A machine with greater parallel computation ability lets computations operate on larger data sets in the same amount of time. In this work, however, the multi-core specifications used performed well under Amdahl's assumptions [28].

2.12 Scalability and Efficiency (λ)

[17], [23] and [9] defined scalability of a parallel program as its ability to achieve performance proportional to the number of processors used. The continual increment of vector processors in parallel computation improves parallel code continuously as can be measured with speedup performance metric. Efficiency is a measure of parallel performance that is closely related to speed up and is often also presented in a description of the performance of a parallel program. Efficiency (λ) of a parallel system is given as:

Feature	Specification
Model	Dell PowerEdge 2950
Processor clock speed	Intel(R) Xeon(R) CPU E5410 @ 2.33GHz (8 CPUs)
Front-side Bus speed	1333MHz
L2 Cache	4MB per core
Memory	16378MB RAM
Operating System	Windows Server® 2008 HPC Edition (6.0, Build 6001) (64-bit)

$$\lambda = \frac{S_p}{p} * 100\ \% \qquad (8)$$

Factors determining efficiency of parallel algorithm include the following:

- Load balance – this involves distribution of tasks among processors
- Concurrency – this involves processors working simultaneously
- Overhead – this involves additional work (other than the real processing) either not available or an extension responsibility to sequential computation

3. METHODOLOGY

In this study, we developed sequential and parallel algorithms with their corresponding program codes for computation of solar quiet daily (Sq) variations of geomagnetic field in horizontal, vertical and nadir components. One-minute resolution of geomagnetic data from 64 observatories worldwide were used in preference to conventional resolution of seasonal, yearly, monthly, diurnal or hourly time resolution. Although, one-minute time resolution of the magnetic data used for the analysis generated large matrices, the parallel computation provided better result in a reasonable time. Some previous works by [12], [15], [18], [3], [16] were limited to low time resolution magnetic data due to non-availability of high-performance computing resources in terms of complexity involved parallel programming and parallel hardware. The Sq model gives a near-comprehensive spatial and temporal representation of Earth's complex magnetic field system. According to [24], Sq considers sources in the core, crust, ionosphere, and magnetosphere which accounts for induced fields in the E-region.

Sequential and parallel MATLAB m-file programs were developed to solve for Sq mean values on monthly representations. Performance improvement was made to code segments that used inner loop for 1440 iterations and outer loop for 64 times along side with variables slicing so as to reduce average runtimes and thereby improving overall performance. The main sliced variables were indexed and arranged logically so as to enable ease of parallelization and subtasks allocation to eight processors. Table 1 shows the system specifications of the parallel hardware architecture used for processing the Sq algorithm. The system comprises of dual quad (8*CPUs) with L2 cache of 4MB per core and primary memory of 16GB RAM. The most critical challenge involved was the division and allocation of the Sq algorithm subtasks to each processor in the multicore design shown in figure1.

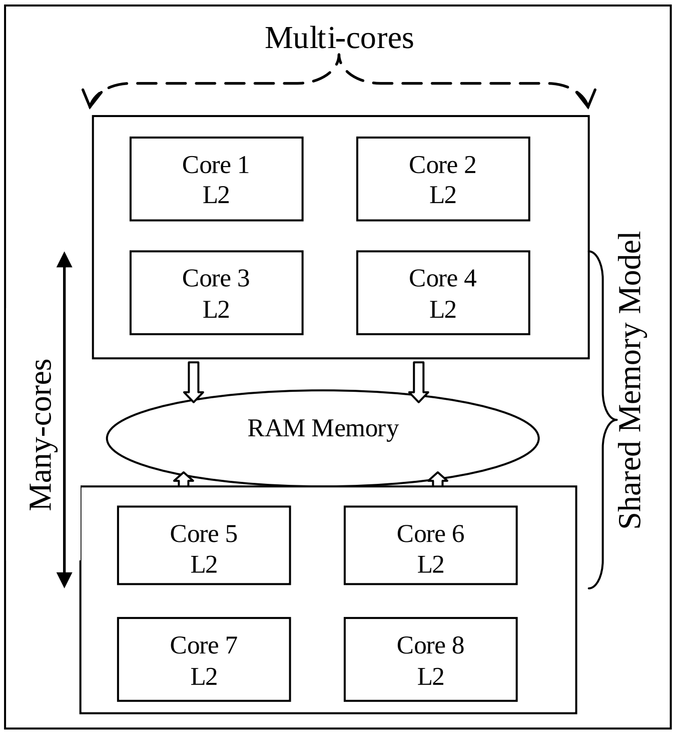
F

Table 1: Systems parameters

The analysis of Sq parallel algorithm was based on Amdahl's laws and models with respect to the fixed workload that involved computation of Sq using one-

Processor (ρ)	Execution time(s)	Computation time(s)	Overhead time(s)
1	801.549	791.347	10.202
2	409.737	402.514	7.223
3	302.236	294.904	7.332
4	242.909	235.39	7.519
5	218.074	210.46	7.614
6	208.901	201.054	7.847
7	192.848	184.768	8.08
8	185.517	177.03	8.487

minute magnetic data of year 1999 when solar activities were empirically proved to be minimal. The multicore systems involved Intel Xeon *E5410* processors as high-performance platform for easy parallelization of Sq



computations and application development. Table 2 shows the execution, computation and overhead runtimes of the parallel algorithm on eight parallel processors. Execution time is total observed time for the parallel algorithm to run using a given number of processor. Computation time is the actual duration for processing the problem, while the overhead time is the time taken to slice the computation into sub-tasks, assign tasks to processors, and collect the intermediary results and close the ports as expressed in equations (9) to (11).

Execution time (E_t) includes computation time (C_t) and overhead time (O_t) as shown in equation (9).

$$E_t = C_t + O_t \tag{9}$$

The Computation time can be further divided into 2 parts as follows:

$$C_t = t_s + t_p \tag{10}$$

Where t_s = computation time needed for the sequential section

t_p = computation time needed for the parallel section

Clearly, if the problem was parallelized, only t_p could be minimized. Considering an ideal parallelization, we obtain:

$$C_t = t_s + \frac{t_p}{p} \tag{11}$$

Table 2: Execution, computation and overhead runtimes of the parallel algorithm

Fig.2 shows the schematic of the parallel algorithm developed and the improved model for speedup performance metric on the basis of Amdahl's theorem.

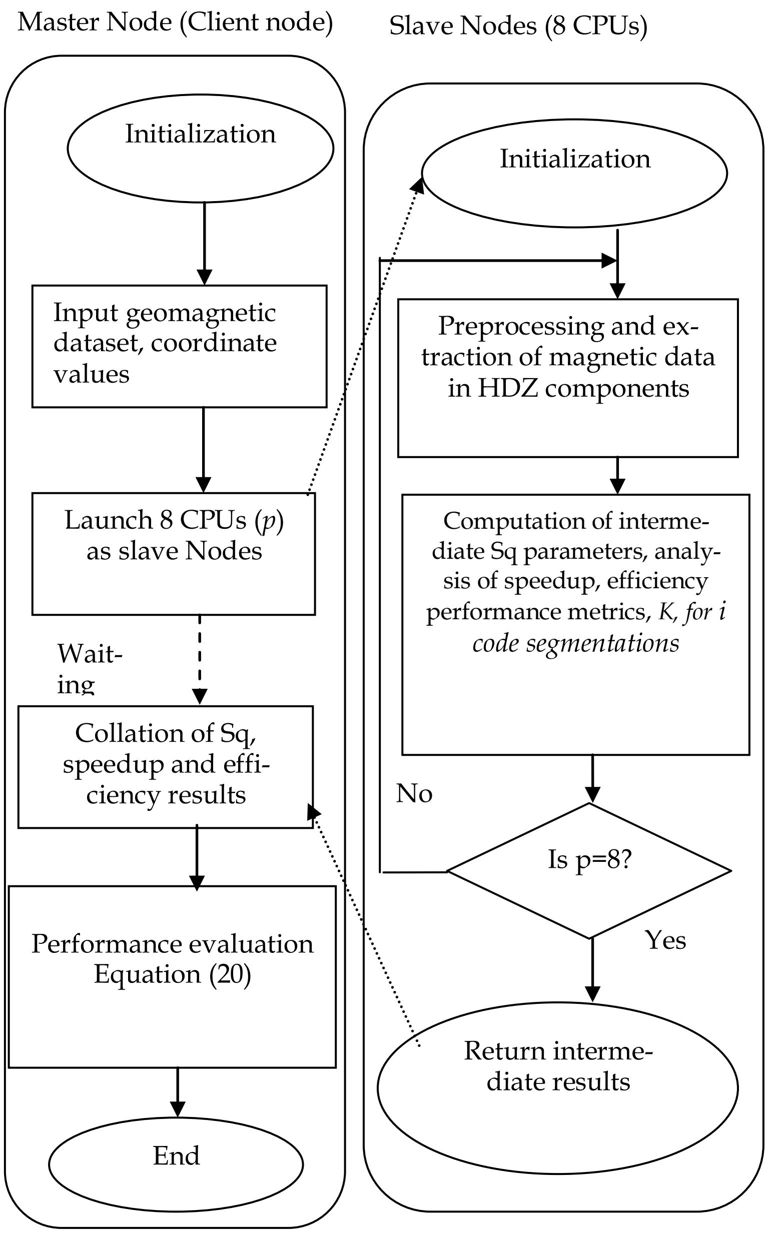

Master Node (Client node)
Slave Nodes (8 CPUs)
Initialization
Input geomagnetic dataset, coordinate values
Launch 8 CPUs (p) as slave Nodes
Waiting
Collation of Sq, speedup and efficiency results
Performance evaluation Equation (20)
End
Initialization
Preprocessing and extraction of magnetic data in HDZ components
Computation of intermediate Sq parameters, analysis of speedup, efficiency performance metrics, K, for i code segmentations
No
Is p=8?
Yes
Return intermediate results

Fig. 2: Sq parallel Algorithm and Performance Analysis

$$R = \frac{\Sigma(t_1 - \overline{t_1})(t_2 - \overline{t_2})}{\sqrt{\Sigma(t_1 - \overline{t_1})^2 \Sigma(t_2 - \overline{t_2})^2}}$$

Major causes of overhead time were attributed to processors load imbalance, inter-processor communications (synchronization), time to open and close connections to the slave nodes and idle time. Considering equation (1), the percentage of sequential portion carried out in the computation can be empirically expressed as follows:

$$f = \frac{p - S_p}{S_p(P-1)} * 100\% \qquad (12)$$

From table 2, when p =8, S = 4.32, that is, S(8) = 4.32. Substituting these values into equations (2) and (12), the percentage of sequential portion; f ≈12.2% while the percentage of parallel portion is ≈ 87.8% as indicated in section 4 of table 4 and 5. This result confirmed Amdahl explanation that typical values of "f" could be large enough to support single processors. The Sq was computed with 8 processors in a parallel environment and speedup factor was used to display how the developed parallel code scales and improves in performance as shown in fig.3.

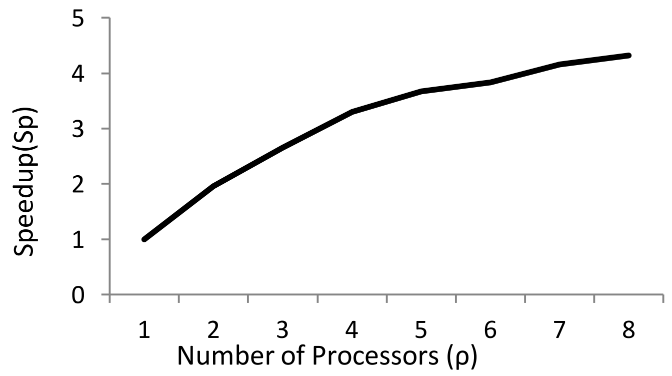


Fig. 3: Speedup vs number of processors

The speedup curve was logarithmic in nature as shown by the fig. 3 above. The logarithmic function explains effect of processors variability on the performance of a parallel system. The strong value coefficient of determination (R^2) indicates a strong relationship between the observed and unobserved values. Regression analysis was used to generate an equivalent model as shown in equation 13.

$$S_p = 1.6468*\ln(\rho) + 0.9293 \qquad (13)$$

The coefficient of determination was computed using Person's method as stated in equation (14). The equation for the Pearson product moment correlation coefficient, R, is defined:

(14)

$0 \leq R \leq 1$

Where t_1 are the observed runtimes while t_2 are the predicted runtimes. The coefficient of determination is (R^2 = 0.9951) is indicated on the graphs. It is suffice to note that the polynomial and spine interpolation algorithms applied gave reasonable predicted values up to near the observed values after which Runge phenomenon manifested.

The number of processors was varied from one (1) to eight (8) and the execution runtimes in seconds are recorded in table 3.

Table 3: Performance metrics for the parallel algorithm

CPU (ρ)	Execution Time(s)	Time (min)	Speedup (Sρ)	Efficiency (%)
1	801.55	13.36	1.00	100.0
2	409.74	6.83	1.96	97.81
3	302.24	5.04	2.65	88.40
4	242.91	4.05	3.30	82.49
5	218.07	3.63	3.68	73.51
6	208.90	3.48	3.84	63.95
7	192.85	3.21	4.16	59.38
8	185.52	3.10	4.32	54.01

Ideally, from table 3, if a single CPU took 801.55 seconds to process the Sq problem, it may be reasonable to agree that 8 CPUs would spend 801.55/8 = 100.19 seconds to solve the same problem. However, it was observed that execution runtime using eight processors is 185.52 seconds. The difference was due to overhead time intrinsically incurred in the process of parallelization at software and hardware levels.

3.1 The relationship of the Performance Metrics

A comparison analysis was carried out between the performances of serial and parallel algorithms for the solution and execution of Sq algorithm considering the runtime of the fastest serial algorithm. Fig.4 shows the efficiency of the parallel computation as a function of how well the processors were utilized to carry out the assigned jobs or tasks.

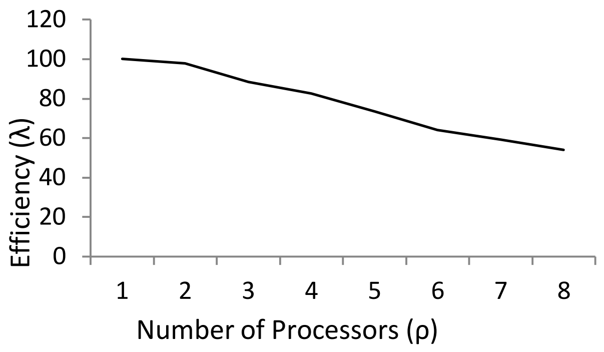


Fig. 4: Efficiency of the parallel algorithm

The efficiency curve shows a negative gradient as the efficiency reduces with increased number of processors. This metric measures the effectiveness of parallel algorithm with respect to the computation time. Therefore, the efficiency (λ) is a function of load-balance (μ), concurrency (τ), and parallel overhead (γ) parameters as expressed in equations (15) and (16). The subtasks, *U*, must exit, job divisible, for a parallel computation to hold.

$$\lambda = f(\mu, \tau, \gamma) \quad (15)$$

$$\mu = \sum_{i=1}^{n} U_i \quad (16)$$

Load-balance (μ) was implemented by distributing subtasks (with subscript i) to processors, p, from *i* to *n* where n is the total number of parallel segments such as loop into which the program code or algorithm can be divided. Efficiency describes the average speedup per processor. Table 3 vividly shows that the nominal value of efficiency at a given processor lies between 0 and 1 (or simply, 0% ≤ φ ≤100%). Efficiency (λ) of the parallel algorithm was observed to be directly proportional to load-balance and concurrency, and at the same time indirectly proportional to the overhead cost of the workload. We derived mathematical relationships among the parallel computing performance parameters in equations (17) to (19) as follows:

$$\lambda \propto \frac{\mu * \tau}{\gamma} \quad (17)$$

So,

$$\lambda = \frac{k * \mu * \tau}{\gamma} \quad (18)$$

This gives,

$$k = \frac{\lambda * \gamma}{\mu * \tau} \quad (19)$$

Where k = constant parameter for parallel overhead

The parameter *(k)* is a quantifiable parallel-parameter that largely depends on systems parameters (hardware and software – algorithm, compilers and memory) as they actually influence the processing speed of the algorithm. Therefore, the systems of the parameters of the parallel architecture must be stated as a reference point and pre-conditions for the determination of the efficiency of a parallel algorithm.

4. RESULTS AND DISCUSSION

The sequential program runs for 18.5 minutes on a single CPU, the optimized parallel version of the code runs for 801.55 seconds on one CPU, thus affirming that Ts ≠ T(1) occurs due to overhead cost variations incur by processors in parallel. However, the parallel program runs for 13 minutes on a single CPU. Parallelization details are given in table 4 and 5.

Table 4: Sequential or parallel algorithm running on a single CPU

P(1)
100% Sequential

Table 5: The percentage runtimes for parallel and sequential section of the algorithm

Number of Processors	Percentage Parallelized	Percentage Serialized
P(1)	11%	12% Sequential
P(2)	11%	
P(3)	11%	
P(4)	11%	
P(5)	11%	
P(6)	11%	
P(7)	11%	
P(8)	11%	

According to the Sq parallel computation runtimes, the values of speedup are less than the number of processors, that is, $S_p < p$.

This confirms the Amdahl's theory for an ideal system of fixed-workload with attendant difficulty in obtaining speedup value equal to the number of processors. The execution parallel program has a fraction, *f =12%,* which could not be parallelized and therefore was executed sequentially. Table 4 shows that 88% of the program was parallelized using eight processors. Therefore, Amdahl's equation was modified and improved upon so as to account for parameter (*k*) as an overhead function that determines accuracy of the speedup and relative overall performance of a parallel algorithm on parallel systems as shown in equation (20).

$$S_p = \frac{1}{f + (1-f)/p} + \sum_{i=1}^{n} \frac{1}{k^i} \qquad (20)$$

Where *n* = number of threads and processes of the parallel algorithm or code

5. CONCLUSION

Parallelization improves the performance of programs for solving large and/or complex scientific and engineering design problems. Metrics used to describe the performance include execution time, speedup, efficiency, and cost. These results complement existing studies and demonstrate that parallel computing and multicore platforms provide better performance of space weather parameter, Sq with the use of higher time resolution magnetic data.

In this study, we analyze performance of parallel computation of Sq using data-intensive one-minute time series magnetic data from 64 observatories worldwide. The fraction of the Sq program that could not be parallelized was identified and minimized according to Amdahl's model and recommendations multicore platform of eight parallel processors under fixed-time and memory-bound conditions The algorithm and model developed showed that speedup anomalies result from excessive overheads by Input and Output operations, sequential bottle algorithm, memory race condition, too small or static problem size, extreme sequential code segments and workload imbalance. The anomalies were tackled by evaluating the derived parallel-parameter *K*, as an intrinsic augment to Amdahl's model as expressed in the derived mathematical expression. This improved model further complements the previous results obtained by some other authors [28].

ACKNOWLEDGEMENT
This research was supported in part by National Space Research & Development Agency (NASRDA), Abuja, Nigeria for provision of parallel computing systems which were configured for parallel computation of complex projects in Aerospace Science and Engineering.

REFERENCES

[1] Amdahl, G.M., 1967, Validity of the single-processor approach to achieving large scale computing capabilities, in: *Proc. Am. Federation of Information Processing Societies Conf.*, AFIPS Press, pp. 483-485.

[2] Culler D.E., Singh J.P., Gupta A., 1999, Parallel Computer Architecture: A Hardware/Software Approach, *Morgan Kaufmann Publishers*.

[3] Ezema, P.O., Onwumechili, C.A. and Oko, S.O. (1996). Geomagnetically quiet day ionospheric currents over the indian sector III. Counter equatorial currents. *J. atmo T err. Phys , 58*, 565-577.

[4] Gene H. Golub. and James M. Ortega., 1993, *Scientific Computing: An Introduction with Parallel Computing*, Academic Press, Inc.

[5] Grama, A. et al, 2003, *An Introduction to Parallel Computing: Design and Analysis of Algorithms*, Addison Wesley, 2nd edition

[6] Gropp, W. et al, 2002, *The Sourcebook of Parallel Computing*, Morgan Kaufmann

[7] Gschwind M., 2006, Chip Multiprocessing and the Cell Broadband Engine, *ACM Computing Frontiers*

[8] Gustafson J.L., 1988, Reevaluating Amdahl's Law, *Comm. ACM*, pp. 532-533.

[9] Hennessy J., Patterson D., 2006, *Computer Architecture: A Quantitative Approach*, 4th ed., Morgan Kaufmann

[10] Hill M., Marty M.R., 2008, Amdahl's law in the multicore era, *IEEE Computer* 41 (7), 33_38.

[11] Hwang K. and Xu Z., 1998, *Scalable Parallel Computing: Technology, Architecture, Programming*, McGraw-Hill, NY, ISBN, 0-07-031798-4

[12] Hutton, R. (1967), Sq currents in the American equatorial Zone during the IGY-II. Day to day variability. *J. atmos., Terr. Phys., 29, , 26*, 1429 - 1442.

[13] Jordan, Harry. F., Gita Alaghband 2002, *Fundamentals of Parallel Processing*, Prentice Hall

[14] Joseph, J. and Fellenstein, C., 2003, *Grid Computing*, Prentice Hall

[15] Kane, R. (1971), Relationship between II ranges at equatorial and middle latitudes. *J. atmos. Terr. Phys. , 33*, 319-327.

[16] Kastogi, R.G., Rao, D.R.K., Alex, S., Pathan, B.N., and Sastry, T.S.,. (1997). An intense SFE and SSC event in geomagnetic H, Y, and Z fields at the Indian chain of observatories. *Atm, Geophysicae , 15*, 1301 - 1308.

[17] Ladd, S., *Guide to Parallel Programming*, Springer-Verlag, 2004

[18] Malin. S.R.C. and Gupta, J. (1977). The Sq current system during the International Geophysical year. Geophys. J. R. astr. Soc., , 49, 515-529.

[19] Michael J. Quinn, 1994, *Parallel Computing Theory and Practice*, McGraw-Hill, Inc.

[20] Michael J. Quinn, 2004, *Parallel programming in C with MPI and OpenMP*. New York : McGraw - Hill, 2004. 507 s. Am

[21] Morad T. et al., 2005, *Performance, Power Efficiency, and Scalability of Asymetric Cluster Chip Multiprocessors*, Computer Architecture Letters, vol. 4

[22] Rabiu A.B., 2001, *Seasonal Variability of Solar Quiet at Middle Latitudes*, Ghana *Journal of* Science, 41, 15-22

[23] Sun X.H., Ni L., 1993, Scalable problems and memory-bounded speedup, *Journal of Parallel and Distributed Computing*, 19, 27-37

[24] Tan, A., Lyatsky, W, 2002, *Seasonal Variations of Geomagnetic Activity at High, Middle and Low Latitudes, American Geophysical Union*, Spring Meeting, abstract #SM51A-14

[25] Woo D. H., Lee H. H, 2008, Extending Amdahl's law for energy efficient computing in the many-core era, *IEEE Compute*r 41 (12) 24_31

[26] Wulf W.A., McKee, S.A., 1995, *Hitting the memory wall: Implications of the obvious, Computer Architecture* News 23 (1) 20_24

[27] Wyrzykowski, R., 2004, *Parallel Processing and Applied Mathematics*, Springer

[28] Xian-He Sun, Yong Chen, 2010, Reevaluating Amdahl's law in the multicore era, *Journal of Parallel Distributed Computing* 70, 183-188, Elsevier Inc.