Asian Journal of Research in Computer Science

8(4): 21-34, 2021; Article no.AJRCOS.67791
ISSN: 2581-8260

Parallel Scheduling of Grid Jobs on Quadcore Systems using Grouping Methods

Goodhead T. Abraham[1*], Evans F. Osaisai[2] and Nicholas S. Dienagha[1]

[1]Computer Science Department Niger Delta University, Bayelsa State, Nigeria.
[2]Mathematics Department, Niger Delta University State, Nigeria.

Authors' contributions

This work was carried out in collaboration among all authors. Author GTA designed the study, performed the statistical analysis, wrote the protocol, and wrote the first draft of the manuscript. Author EFO managed the analyses of the study. Author NSD managed the literature searches. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/AJRCOS/2021/v8i430207
Editor(s):
(1) Dr. Manish Mahajan, CGC College of Engineering, India.
Reviewers:
(1) Liang-ze-nan, Hohai University, China.
(2) Abhishek Gudipalli , Vellore Institute of Technology, India
Complete Peer review History: http://www.sdiarticle4.com/review-history/67791

Original Research Article

Received 05 March 2021
Accepted 10 May 2021
Published 17 May 2021

ABSTRACT

As Grid computing continues to make inroads into different spheres of our lives and multicore computers become ubiquitous, the need to leverage the gains of multicore computers for the scheduling of Grid jobs becomes a necessity. Most Grid schedulers remain sequential in nature and are inadequate in meeting up with the growing data and processing need of the Grid. Also, the leakage of Moore's dividend continues as most computing platforms still depend on the underlying hardware for increased performance. Leveraging the Grid for the data challenge of the future requires a shift away from the traditional sequential method. This work extends the work of [1] on a quadcore system. A random method was used to group machines and the total processing power of machines in each group was computed, a size proportional to speed method is then used to estimates the size of jobs for allocation to machine groups. The MinMin scheduling algorithm was implemented within the groups to schedule a range of jobs while varying the number of groups and threads. The experiment was executed on a single processor system and on a quadcore system. Significant improvement was achieved using the group method on the quadcore system compared to the ordinary MinMin on the quadcore. We also find significant performance improvement with increasing groups. Thirdly, we find that the MinMin algorithm also gained marginally from the quadcore system meaning that it is also scalable.

**Corresponding author: E-mail: at.goodhead@gmail.com*

Keywords: Quadcore; multicore-environment; parallel scheduling; multi-scheduling; machine grouping; job grouping.

1. INTRODUCTION

Most scheduling algorithms are hardly parallelisable [2]. Hence, most attempts at parallelisation rely on the underlying hardware, this practice does not exploit the parallelism potentials of the underlying hardware to the maximum. The dawn of the multicore era combined with the growing processing need of the computing world (especially on the Grid) calls for a change from sequential scheduling to parallel scheduling or a method that enhances parallel scheduling. This paper employs grouping methods to exploit parallelism on multicore systems.

The remainder of this paper is organized as follows: Section 2 discusses related work. Section 3 presents the proposed random grouping method of grouping machines and speed proportional to size method of grouping jobs. It also discusses the simulations used, experimental setup and the configuration of the machines. Section 3 also discusses the result of the experiment, analyses and evaluation of the result. Section 4 makes recommendation. Conclusion and thoughts for future work are presented in section 5.

2. RELATED WORK

Amdahl [2] hinted that parallelising the sequential portion of an algorithm would z result in speedup of processing rate for every processor added to the system. However, [3] pointed out that hardware gains are not been reflected comparatively in software gains (Moore's dividend). [4] also revealed how Microsoft code was eating up the Moore's dividend by doubling up in every 866 days leading to the claim by [5] that multicore systems are not being fully exploited but have the potential for high performance computing if programmed efficiently.

Advances in the hardware technology has birthed the multicore era , [6] which is currently seen as the hardware technology of today and the foreseeable future [7,8].

Multicore systems are mainly for parallel processing and performances speedup, to gain from multicore systems, [3] suggested parallelisation of codes and design and development of parallel programming languages. These advances in computing hardware technology therefore necessitates a shift in programming culture from sequential to parallel [9,10]. However, most sequential algorithms are not completely parallelisable; this necessitates a general method that enhances parallelism in the execution of all algorithms. Grouping jobs and machines before executing independent scheduling instances within the groups (multischeduling) enhances parallelism on the multicore systems and increases throughput [11]. A parallel scheduling method that exploits the benefits of multicore technology will be helpful in the defined path of the Grid, facilitating increased throughput and increase scalability. This work aim is to develop a dynamic grouping method that enhances parallelism and increases throughput in the scheduling of grid jobs.

2.1 Group Scheduling

The benefits of job grouping have been exploited severally and extensively by researchers; [12] discussed the advantages of distributing jobs among independent grid sites and implemented a method that executed jobs in parallel in multiple grid sites. [13]. grouped small fine-grained jobs to form coarse-graine jobs before scheduling to reduce the communication computation ratio (CCR) that impedes performance. [14] also grouped similar jobs and identifies the group to which the newly submitted job belongs. [15] grouped jobs based on processing capability, bandwidth and memory-size of processing elements. [16] employed job grouping to maximize resource utilization, scalability, robustness, efficiency and load balancing ability of the grid for scheduling of jobs in grid computing. [17] grouped fine-grained jobs into coarse-grained jobs before sending to grid resources. The method successfully minimized the processing time of jobs.

The researches above exploited grouping not for parallelisation but for improved processing. This research 'Parallel scheduling of grid jobs on duocore systems using grouping method' exploits groups not to reduce CCR or reduce the processing time but to create platforms for parallelism and multi-scheduling. The method selects grid jobs and grid machines into independent group pairs before employing several instances of scheduling (using multiple threads) to simultaneously execute in parallel (multischeduling). This is targeted at increasing parallelism and attains high degree of scheduling throughput.

2.2 Some Parallel Scheduling Attempts

Several attempts have been made by researchers to exploited parallelism on parallel systems; [18] used a non-deterministic approach (memetic algorithm) to solve the scheduling problem in a GPU environment. [19] algorithm used a two-list method to solved problems on a GPU using CUDA (Compute Unified Device Architecture) and [10] parallelized the Doolittle algorithm on multicores and achieved a performance better than the serial version of the Doolittle algorithm. [20] also implemented a parallel version of self-organizing maps algorithm (SOM) on a parallel architecture and achieved good performance against the CPU version.

Most of these researches never exploited any known method to increase parallelism but relied solely on the underlying hardware. Also, they executed on specialised hardware CUDA and GPUs.

This work is based on the general-purpose environment and the method is based on a deterministic algorithm that gives control of execution time. Thirdly, this research exploits grouping to achieve greater efficiency through parallelisation.

2.2.1 Parallel scheduling methods using grouping

[21] executed a non-dynamic priority grouping method and achieved speedup in scheduling of jobs. [11] also executed dynamic grouping methods and achieved significant gain in scheduling Grid jobs. Noting that grouping of Grid jobs and machines offers a platform for parallelism, [22] explored various strategies to group machines and Grid jobs before scheduling on HPC system. All the methods recorded increase in throughput based on the volume of jobs scheduled [1]. Experimented a random grouping method and recorded significant improvement on a duocore machine. The researchers noted that investigating further on a system with more cores was necessary to know the effect of more cores on the grouping method.

This work is an effort aimed at investigating the effect of grouping and parallel-scheduling on a quadcore system. The method used is same as the one on the earlier study on duocore systems.

2.2.2 Parallel scheduling of Grid jobs on a quadcore system

This study aims at harnessing the parallelism inherent in the multicore architecture by exploiting a method that group jobs and machines before multi-scheduling independently in the groups (the MinMin scheduling algorithm is used as benchmark). The same number of machine/job groups is used in each run while varying the number of threads. Grouping jobs and machines before scheduling on multicores allows multiple independent scheduling instances to occur simultaneously (multi-scheduling).

The MinMin algorithm [23] calculates the completion time of all jobs on all machines and assigns jobs with the minimum completion time to the processor with earliest completion time. The MinMin algorithm was used as benchmark because it has been used by several researchers as benchmark [24-30,18] and it offers easy and robust implementation.

2.3 Machine Grouping

Machines were randomly selected into groups, for each machine randomly selected to a group, the total processing power of each group is updated with the processing power of the selected machine. The algorithm used to randomly group machines as shown in Table 1 is the same algorithm used in [1].

2.4 Job Grouping

The job grouping method assigns jobs to machine groups based on a ratio calculated from the processing power of the machines in the group. Called size_proportional_to_speed, the method estimates the size of all jobs and ensures equitable distribution using proportionality of jobs to machine. Jobs are assigned to groups based on the ratio of the performance configuration of the machines in that group. From the algorithm in Table 2, the first set of N jobs is allocated to the first group based on the ratio, then the next N jobs are allocated to the next machine group based on the group's ratio. This continues until all jobs are allocated. This algorithm for job allocation is the same algorithm implemented in [1].

2.5 Grid Site

Grid-sites are characterised with unique attributes viz: Network Bandwidth, number of computing machines, Grid Id. Also, machines in each Grid -site are unique with distinct attributes like CPU; RAM; Bandwidth. The simulation of Grid sites in the experiment was done to reflect these distinct attributes. Table 3 shows the features and characteristics of Grid site used in the simulation experiment.

2.6 Grid Machines

Every Grid site contains hundreds to thousands of computing machines, and each computing machine is distinct by its configuration. Grid machines or compute resources are characterized by distinct features like the machine's identification (MId), speed of processor (SP), number of processor cores (NPC) and RAM size.

2.6.1 Simulation of grid, CPU speed and number of cores

The Grid was simulated to be characterized by the following attributes: Category; CPU; RAM; Bandwidth. For example {A; 1200; 2000000; 1000} represents Grid site A, CPU 1200, RAM 2000000, and Bandwidth 1000.

The computer machine was defined with the following attributes: CORES; CPU; RAM. For instance {2; 2000; 2000000} represents a Grid resource (machine) with 2CPUs, 2000 MHz (2GHz) and 2000000B (2MB). Table 4 shows the characteristics of Grid machines used.

Table 1. Algorithm to group machines randomly

```
Random Algorithm to select machines for grouping
Step1: Start
Step2:  Determine g (g is the number of job groups)
Step3:  $Sum_{Group}$ = 0 (initialize group processing power)
Step4:  $Sum_{Total}$ =0 (initialize total processing power of all groups)
Step5: Randomly
    a.  Select a machine $M_i$
    b.  Insert to group i
    c.  Increment group processing power by $M_{iP}$ ( $Sum_{Group} = Sum_{Group} + M_{iP}$ )
    d.  Increment group
    e.  Increment Total group processing power $Sum_{Total} = Sum_{Total} + M_{iP}$ )
Step6: Do step 5until all machines are assigned
Step7: Stop
```

Table 2. Algorithm to estimate size of job for allocation to groups

```
Job allocation algorithm
Step1: Start
Step2: Get processing power of all machines in each group $Sum_{Group}$
Step3: Get cumulative processing power of all groups $Sum_{Total}$
Step4: sum total size of all jobs $Sum_{Jobs}$
Step5: Estimate size of jobs to be allocated to each group
Step6: Select first N jobs making up the ratio for group i
Step7: Increment group count
Step8: Select next N jobs making up the ratio for next groups
Step9: Repeat steps 7 to Step 8 until all jobs are assigned
Step10: Stop
```

Table 3. Characteristics of Grid site used in the simulation

Features	Characteristics	Attributes
Network Bandwidth	Every Grid site is connected to the Grid via a network and the speed of the network connecting the Grid site determines to an extent the performance of the Grid. The network bandwidth (NBW) or speed of a Grid site is therefore used as one of the attributes to characterize a Grid site.	Network bandwidth or speed (NBW) are sub categorized into; Super-Fast (SF), Very Fast (VF), Medium Fast (MF) and Not Fast (NF) with weights 4, 3, 2 and 1 respectively.
Number of Machines	This feature refers to the number of computers that the Grid site contains. The number of machines within a Grid site can be arbitrary. It can be any number, in some cases due to computer system characteristic of failure and repair, the number can vary from time to time.	This number varies over time hence there is no need for categorization
Grid ID	This is the identification features of the Grid site. The Grid ID can be the name or number used to identify the Grid	Name or number or combination of both

Table 4. Characteristics of grid site used in the simulation

Grid Site	Characteristics			Grid Site	Characteristics		
	Number of machines	Speed of CPU	Number of CPU/ Cores		Number of machines	Speed of CPU	Number of CPU/Cores
A 240 Machines	30	1GHz	1	C 480 Machines	60	1.5GHz	2
	30	2GHz	1		60	2GHz	2
	30	3GHz	1		60	3.5GHz	2
	30	4GHz	1		60	4GHz	2
	30	1GHz	2		60	1.5GHz	4
	30	2GHz	2		60	2GHz	4
	30	3GHz	2		60	3.5GHz	4
	30	4GHz	2		60	4MHz	4
B 400 Machines	50	1.5GHz	2	D600 Machines	50	1.5GHz	2
	50	2GHz	2		50	2GHz	2
	50	3.5GHz	2		50	3.5GHz	2
	50	4GHz	2		50	4GHz	2
	50	1.5GHz	4		50	1.5GHz	4
	50	2GHz	4		50	2GHz	4
	50	3.5GHz	4		50	3.5GHz	4
	50	4GHz	4		50	4GHz	4
					50	1.5GHz	8
					50	2GHz	8
					50	3.5GHz	8
					50	4GHz	8

2.6.2 Source of jobs to the system

Jobs used for the experiment were downloaded from the Grid Workloads Archive designed by [31] to make traces of Grid workloads available to researchers and developers.

2.7 Experimental Design

Two experiments each were carried-out on a single processor system ad on a quadcore system.

The first experiment executed the MinMin algorithm on a single processor system to schedule a range of (from 1000 jobs to 10000 jobs in steps of 1000).

The second experiment executed on a quadcore system. It used the MinMin algorithm to schedule a range of jobs from 1000 jobs to 10000 jobs in steps of 1000.

The third experiment was executed on a single processor system. It used the random method to group machines then applied the size_proportional_to-speed method to group jobs before implementing the MinMin scheduling algorithm within the paired groups to schedule from 1000 jobs to 10000 jobs in steps of 1000.

The fourth experiments also executed on a quadcore system; it used the random method to group machines then applied the proportionality method to group jobs before implementing the MinMin scheduling algorithm within the paired groups to schedule the same range of jobs as in the first experiment.

In each of the four experiments, the number of groups used was varied between 2, 4 and 8 groups, the number of threads used was varied between 1, 2, 4 and 8. For each of the combinations, the time taken to schedule was recorded. Time of scheduling is the time taken to schedule each set of jobs, that is the time taken to schedule 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, and 10000 jobs in turn by each thread cardinalities. This experiment is the same experiment used in [1].

2.8 System Properties

The properties of the systems are as follows:

1. SINGLE PROCESSOR SYSTEM:

Processor:	Intel(R) Pentium(R) 4 CPU 3.00GHZ3.00GHz
RAM:	1.50 GB
Operating System:	Windows XP Professional Version 2002

2. QUADCORE:

Processor:	Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz 3.40 GHz
RAM:	24.0GB
System Type:	64-bit Operating System, x64-based processor
Operating System:	Windows 8.1

3. RESULTS AND DATA ANALYSIS

This section discusses and compares result between the ordinary MinMin and the Group method. In the discussion, 1Thrds, 2Thrds, 4Thrds or 8Thrds refer to the number of threads used in the experiment. SingleCPU or SingleProc,1CPU or 1Proc refer to result obtained on the single processor machine; Quadcore represent result obtained on the quadcore machine and 2Grp, 4Grp or 8Grp refer to the number of groups used. Also, SpdRnd refers to the method Size-Proportional-to-Speed. A combination of Threads, groups and machines are used in the analysis to denote results. For instance, 2ThrdsSingleCPU 2Grps represent result obtained using 2 threads and two groups on the single processor system.

3.1 Performance of the Min Min Algorithm on the Single Processor and Quadcore Systems

Table 5 and Fig. 1 show the results and performance of the ordinary MinMin on the two machines (single processor and quadcore). Using two threads, it took 1235755 Ms for the single processor and 136818Ms for the quadcore system to schedule the same range of jobs. The MinMin algorithm on the quadcore performed 9.03 times or 88.93% better than MinMin executed on the Single processor system. This is an indication that the MinMin is also scalable as it gained from the underlying parallelism of the quadcore. Despite the MinMin algorithm benefitting from the underlying parallelism of the quadcore, we believe the gain is not significant enough not to use a method that enhances parallelism on quadcore. Analysis between the grouping method and the ordinary MinMin will reveal this. See section 5.2, 5.3 and 5.4.

3.2 Analysis of Result on the Quadcore System

This analysis is for result between the ordinary MinMin executed on the quadcore and group method executed on the quadcore machine. The result is shown in Table 6 while Table 7 shows the computed improvement over the MinMin in multiples (X) and in percent (%).

On the quadcore; using two groups, the group method performed better than the ordinary MinMin by 2 times or 50%. With four groups, the group method performed better than the MinMin by approximately 4 times which represents about 75% and using eight groups, the group method performed better than the MinMin by approximately 7 times representing 86%. Fig. 2 shows that on the quadcore, as the number of group increases, the performance of the group method over the ordinary MinMin also improves. It also shows that as the number of threads increases on the quadcore, the performance of the group method over the MinMin also improved. This is because more threads increase parallelism on the quadcore.

From this, we can deduce that the group method expands the realm of parallelism as it increases scheduling throughput.

On the improvement line in Fig. 2, a linear trendline was inserted and the linear equation was Y ≈ 2.4X – 0.5. This indicates that the improvement (Y) depends on the number of groups (X). Also, the R-Squared value of the trendline is 0.98 which indicates that the trendline fits perfectly to the improvement line.

Table 5. Result and performance of MinMin on a single processor and Quadcore systems

Jobs Limit	MinMin2Thrds (SingleCPU)	MinMin2Thrds (QuadCore)
1000	4500	399
2000	17672	1795
3000	40250	4222
4000	66922	7188
5000	90407	9994
6000	116922	12700
7000	154781	16690
8000	196768	21719
9000	244679	27989
10000	302854	34122
Total	1235755	136818
Average	123575.5	13681.8
Improvement over Single processor in multiple (X)		9.0321
Improvement over Single processor in percent (%)		88.9284

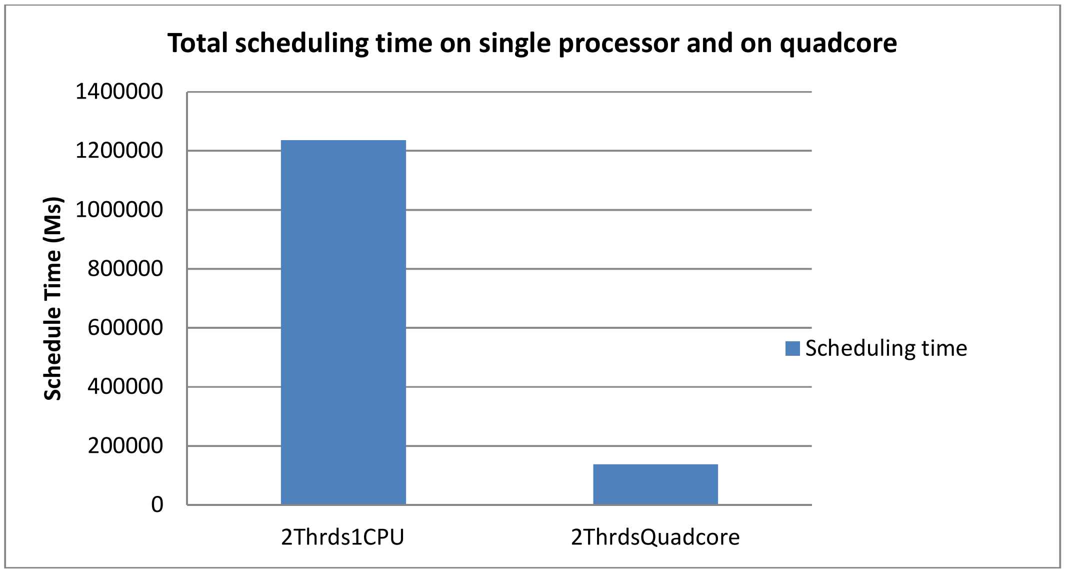

Fig. 1. Total of Min Min on different computer systems

Table 6. Result of MinMin on quadcore machine

	One thread				**Two threads**			
		SpdRnd Method				SpdRnd Method		
No of jobs	MinMin	2Grps	4Grps	8Grps	MinMin	2Grps	4Grps	8Grps
1000	452	192	120	81	403	189	111	78
2000	1924	746	321	186	1775	759	337	196
3000	4178	1683	738	384	4203	1543	648	396
4000	7118	2889	1285	687	7431	2570	1297	671
5000	10013	4093	2230	1082	9903	4339	1928	1178
6000	12825	5921	3193	1522	12984	6433	3089	1568
7000	16685	8408	3964	2465	16872	9079	4315	2361
8000	21633	11298	5918	3447	21956	11261	5758	2855
9000	27624	14667	6914	4396	27778	14340	7082	4506
10000	34406	17414	8959	5405	34570	17385	10593	6379
Total	**136858**	**67311**	**33642**	**19655**	**137875**	**67898**	**35158**	**20188**

		Four threads		
		SpdRnd Method		
	MinMin	2Grps	4Grps	8Grps
	399	193	104	83
	1795	777	313	184
	4222	1643	648	397
	7188	2879	1309	667
	9994	4149	1904	966
	12700	6481	3183	1716
	16690	8559	4529	2283
	21719	10533	5606	2788
	27989	14378	7048	4141
	34122	17681	9935	4967
Total	**136818**	**67273**	**34579**	**18192**

Table 7. Performance improvement on quadcore machine with increasing groups and threads

		SpdRnd Method					
		Improvement in Percent (%)			**Improvement in Multiples (X)**		
	Improvement	2Grps	4Grps	8Grps	2Grps	4Grps	8Grps
1	1 Thread	50.81691	75.41832	85.6384	2.033219	4.06807	6.963012
2	2 Threads	50.75394	74.50009	85.35775	2.030619	3.921583	6.829552
3	4 Threads	50.83030	74.72628	86.7035	2.033773	3.956679	7.520778
Aggr Improvement		50.80038	74.88156	85.89988	2.032537	3.98211	7.104448

3.3 Combination of the Results

This section combines and analyses the results on the single processor machine and on the quadcore machine. Table 8 shows the result and the computed improvements. We compared the best result of the MinMin (the one executed on the quadcore) to result of group method executed on the single processor and on quadcore.

The MinMin executed on the quadcore performed better than the group method executed on the single processor by 4.41, 2.39 and 1.50 times. This represents about 77%, 58% and 32% when using 2, 4 and 8 groups respectively. This is shown in the negative (falling) part of Fig. 3 (marked single processor).

The group method executed on the quadcore system performed better than the MinMin executed on quadcore by 2.03, 3.92 and 6.81 times representing 50%, 74% and 85% using 2, 4, and 8 groups respectively. This is also shown in the rising part of Fig. 3 (marked quadcore).

Within the single processor system, there was an average of 1.7 or 40% improvement between successive groups. 4 group performed better than 2 group by 1.85 times or by 45%, 8 group performed better than 4 group by 1.6 times or by 37% and 8 group performed better than 2 group by 2.95 times or by 57.40%.

Within the quadcore system, there was an average of 1.8 times or 45% improvement between successive groups. 4 group performed better than 2 group by 1.93 times (48%), 8 group perform better than 4 group by 1.74 times (42%) and 8 group performed better than 2 group by 3.36 times (70%).

Between the Single processor and the quadcore, there was an average of 9 times or 85% corresponding improvement between the groups. Correspondingly, 2 groups on the quadcore performed better than 2 groups on the single processor by 8.9 times or 88%. Four groups on the quadcore performed 9.36 times or 89% better than 4 groups on the single processor while 8 groups on the quadcore performed 10.18 times or 90% better than 8 groups on the single processor system.

A trendline fitted through the improvement line indicates a linear growth with equation: Y = 0.492x + 1.7913.
This equation indicates that improvement (Y) of the group method within a computing system can be determined by the number of groups (x).

The R-Squared value of the trendline is 0.2174 which indicates that the proportion of variance between improvement and the number of groups is low. This low fit is because of the use of the group method on a single processor machine was compared against result of MinMin on quadcore machine (two different trends). However, this analysis shows that the MinMin is scalable on the quadcore but not as much as the group method on the same system.

These analyses indicate that the group method performed better on the quadcore and is best suited for the multicore environment than the single processor system.

The ordinary MinMin algorithm benefited marginally from the parallelism on the Quadcore. Hence it performed better than the grouping method executed on a single processor system where no parallelism was guaranteed.

3.4 General Discussion on the Results

From the result and analysis, it can be deduced that within a group, the performance is slightly impacted by differences in threads although this was very minimal and insignificant compared to the impact of groups on performance. Major performance improvements were achieved with groups across all platforms.

The MinMin algorithm executed on the Quadcore system gained marginally from the underlying parallelism of the system hence it performed better than the grouping methods implemented on the single processor system. This was because the single processor system offered minimal parallelism only on the level of threads but not on the hardware which the group method targeted.

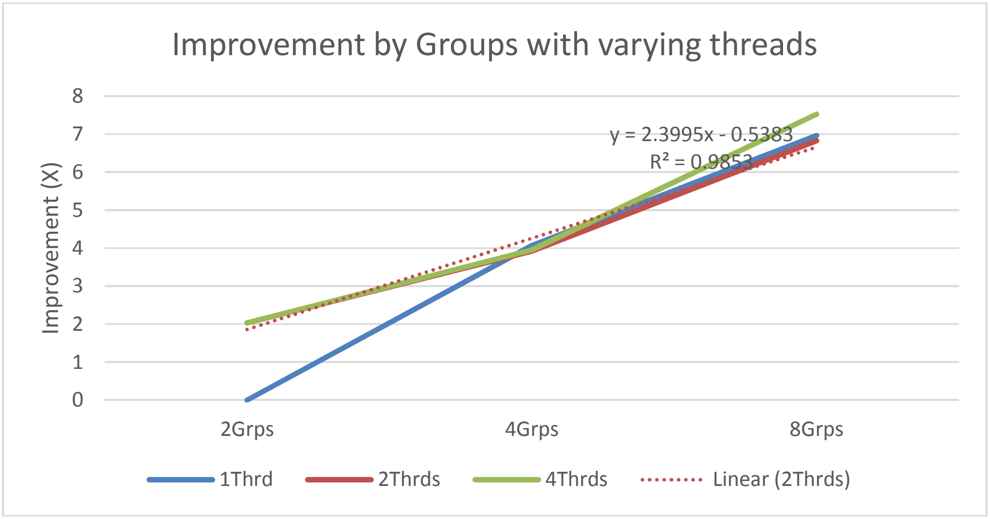

Fig. 2. Performance of group method over MinMin with increasing groups and threads

Table 8. Combined result from single processor and Quadcore

	Single Processor				Quadcore		
	MinMin	**2Grps**	**4Grps**	**8Grps**	**2Grps**	**4Grps**	**8Grps**
1000	403	2360	1094	594	189	111	78
2000	1775	7907	3876	2031	759	337	196
3000	4203	17031	9063	4360	1543	648	396
4000	7431	28468	13656	9359	2570	1297	671
5000	9903	40453	22672	12985	4339	1928	1178
6000	12984	59781	32578	19062	6433	3089	1568
7000	16872	79499	42047	26234	9079	4315	2361
8000	21956	94781	54172	34453	11261	5758	2855
9000	27778	119609	67438	41250	14340	7082	4506
10000	34570	157970	82423	55282	17385	10593	6379
Total	137875	607859	329019	205610	67898	35158	20188
Average	13787.5	60785.9	32901.9	20561	6789.8	3515.8	2018.8
Improvement (X)		4.41	2.40	1.50	2.03	3.92	6.83
Improvement (%)		77	58	32	50	74	85
Improvement Across Group(X), (%)		8grp to 2 grp 2.96, (57%)	4grp to 2grp 1.85, (45%)	8grp to 4grp 1.6, (37%)	8 grp to 2 grp 3.36, (70%)	4grp to 2 grp 1.93, (48%)	8 grp to 4grp 1.74, (42%)
Improvement quadcore vs single processor using corresponding groups (X), (%)					8.95, (88%)	9.36, (89%)	10.12, (90%)

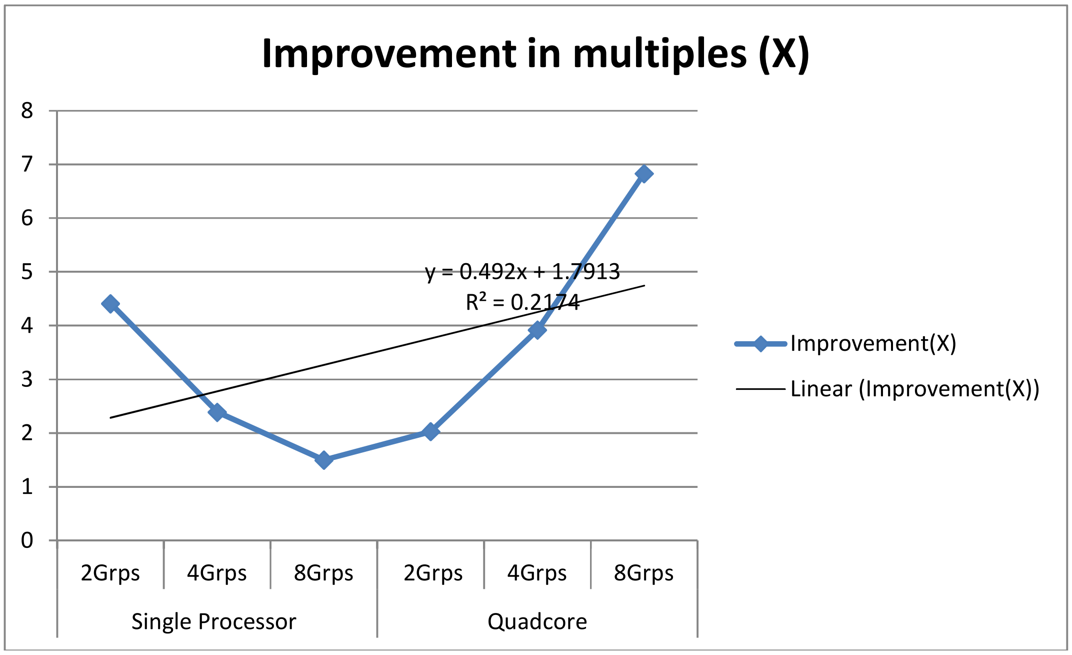

Fig. 3. Aggregate performance

On the quadcore, the group method performed far better than the ordinary MinMin. This is because the method exploited the advantages of the groups and the parallelism on the system, the improvement of the group method over the ordinary MinMin on the quadcore system range from 2 to 7 times depending on the number of groups used.

Though all the machines are not timed at the same speed, we can deduce from the analysis that the grouping method expands the realm of parallelism as it increases scheduling throughput.

4. RECOMMENDATIONS

Haven realized that sequential algorithms do not fully optimize parallelism on parallel systems; to ensure that the computer Grid grows in tandem with advances made in computer hardware technology. We recommend that grid schedulers and applications be integrated with grouping methods to enhances parallelism and increase performance.

5. CONCLUSION

This work aimed at parallelising the scheduling of Grid jobs on a quadcore system using job and machine groups. The method executed on the quadcore system yielded significant improvement compared to the ordinary MinMin on the two platforms. Every major advance in computing technology comes with a paradigm shift in programming [32] the proliferation of multicore systems therefore calls for parallelisation of programming methods. Since not all algorithms are parallelisable; we propose the combine use of job grouping and machine grouping in the scheduling of Grid jobs.

FUTURE THOUGHTS

This research opens a new area of parallelisation of the scheduler by grouping independent jobs and machines. This work can be extended to other scheduling (both sequential and parallel). Though this work targeted independent jobs, the method can also be extended to dependent jobs. This will involve the same application of grouping but cooperating jobs maybe carefully selected to the same group, (and even if they are selected to different groups) they are scheduled to the same Grid site for processing – this will reduce the delay in process communication.

Though different factors characterised the performance of a system, the overall result cannot be normalised or standardized. It will be interesting to experiment on a set of systems from the same family of CPU that shares same features. This will help standardize the result.

Haven experiment this work on single processor system, duocore computer (Abraham and Osaisai 2021), and now a quadcore system, the next step will be to combine and analyse the results for all three platforms and present in a single paper.

COMPETING INTERESTS

Authors have declared that no competing interests exist.

REFERENCES

1. Abraham GT, Osaisai EF. Parallel scheduling of grid jobs on duo-core systems using grouping method. Covenant Journal of Informatics and Communication Technology, in Publica; 2021.
2. Amdahl GM. Validity of the single processor approach to achieving large scale computing capabilities. AFIPS Conference Proceedings - 1967 Spring Joint Computer Conference, AFIPS 1967;483–485.
   Available:https://doi.org/10.1145/1465482.1465560
3. Larus J. Spending Moore's dividend. Communications of the ACM, 2009;52(5):62–69.
   Available:https://doi.org/10.1145/1506409.1506425
4. Myhrvold, N. The next fifty years of software. ACM 97 Conference; 1997.
5. Wang PH, Collins JD, Chinya GN, Jiang H, Tian X, Girkar M, Yang NY, Lueh GY, Wang H. EXOCHI: Architecture and programming environment for a heterogeneous multi-core multithreaded system. Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2007;156–166.
   Available:https://doi.org/10.1145/1250734.1250753
6. Schauer B. Discovery Guides Multicore Processors-A Necessity. In ProQuest discovery guides; 2008.
   Available:http://www.netrino.com/node/91
7. Zhuravlev S, Saez JC, Blagodurov S, Fedorova A, Prieto M. Survey of scheduling techniques for addressing shared in multicore processors. ACM Reference Format. 2012;45(4).
   Available:https://doi.org/10.1145/2379776.2379780
8. Dongarra J, Mathieu F, Thomas H, Mathias J, Julien L, Yves R. Hierarchical QR factorization algorithms for multi-core clusters. Parallel Computing. 2013;39(4–5):212–213.
9. Jin H, Jespersen D, Mehrotra P, Biswas R, Huang L, Chapman B. High performance computing using MPI and OpenMP on multi-core parallel systems. Parallel Computing. 2011;37(9):562–575.
   Available:https://doi.org/10.1016/j.parco.2011.02.002
10. Mustafa B, Rafiya, S, Waseem A. Parallel Implementattion of Doolittle Algorithm using Open MP for multicore machines. 2015 IEEE International Advance Computing Conference. 2015;575–578.
11. Abraham GT, James A, Yaacob N. Priority-grouping method for parallel multi-scheduling in Grid. Journal of Computer and System Sciences, 2015b;81(6):943–957.
    Available:https://doi.org/10.1016/j.jcss.2014.12.009
12. Ernemann C, Hamscher V, Schwiegelshohn U, Yahyapour R, Streit A. On Advantages of Grid Computing for Parallel Job Scheduling. 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid. 2002;339–39.
    Available:https://ieeexplore.ieee.org/abstract/document/1540439/
13. Muthuvelu N, Liu J, Soe L, Venugopal S, Sulistio A, Buyya R. A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids. Australian Workshop on Grid Computing and E-Research, 2005;41–48.
    Available:https://dl.acm.org/citation.cfm?id=1082297
14. Selvi S, Thamarai M, Sheeba, santha K, Prabavathi, K, Kannan G. Estimating job execution time and handling missing job requirements using rough set in grid scheduling. International Conference on Computer Design and Applications. 2010;295–301.
    Available:https://ieeexplore.ieee.org/abstract/document/5541135/
15. Soni VK, Sharma R, Mishra Manoj K. Grouping-based job scheduling model in grid computing. In World Academy of Science, Engineering and Technology.2010;41.
    Available:http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.294.2467&rep=rep1&type=pdf

16. Sharma R, Soni VK, Mishra MK, Bhuyan P, Utpal CD. An agent based dynamic resource scheduling model with FCFS-job grouping strategy in grid computing. Waset ICCGCS; 2010.
Available:https://www.academia.edu/download/5302242/v64-86.pdf
17. Mon TZ, Cho MM. MIPS group job scheduling model for deploying applications. International Conference on Advances in Engineering and Technology (ICAET). 2014;1234–1245.
Available:https://doi.org/10.15242/IIE.E0314069
18. Pinel F, Dorronsoro B, Bouvry P. Solving very large instances of the scheduling of independent tasks problem on the GPU. Journal of Parallel and Distributed Computing Parallel Distributed Computing. 2012;73(1):101–110.
Available:https://doi.org/10.1016/j.jpdc.2012.02.018
19. Wan L, Li K, Liu J, Li K. GPU implementation of a parallel two-list algorithm for the subset-sum problem. Concurrency and Computation: Practice and Experience. 2015;27(1):119–145.
20. Cuomo S, De Michele P, Di Nardo E, Marcellino L. Parallel implementation of a machine learning algorithm on GPU. Journal of Parallel Programming. 2018;46(5):923–942.
21. Abraham GT, James A, Yaacob N. Group-based Parallel Multi-scheduler for Grid computing. Future Generation Computer Systems. 2015a;50:140–153.
Available:https://doi.org/10.1016/j.future.2015.01.012
22. Abraham GT. Group-based parallel multi-scheduling methods for grid computing. Coventry University; 2016.
23. Ibarra OH, Kim CE. Heuristic algorithms for scheduling independent tasks on nonidentical processors. Journal of the ACM. 1977;24(2):280–289.
Available:https://dl.acm.org/doi/abs/10.1145/322003.322011
24. Canabe M, Nesmachnow S. Parallel implementations of the MinMin heterogeneous computing scheduler in GPU. CLEI Electronic Journal. 2012;15(3):8–8.
Available:http://www.scielo.edu.uy/scielo.php?pid=S0717-50002012000300009&script=sci_arttext&tlng=pt
25. Etminani K, Naghibzadeh M. A min-min max-min selective algorihtm for grid task scheduling. In 2007 3rd IEEE/IFIP International Conference in Central Asia on Internet, 2007;1–7.
Available:https://ieeexplore.ieee.org/abstract/document/4401694/
26. Freund RF, Gherrity M, Ambrosius S, Campbell M, Halderman M, Hensgen D, Siegel HJ. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In Proceedings Seventh Heterogeneous Computing Workshop IEEE(HCW 98). 1998;184–199.
Available:https://ieeexplore.ieee.org/abstract/document/666558/
27. Lavanya M, Shanthi B, Saravanan S. Multi objective task scheduling algorithm based on SLA and processing time suitable for cloud environment. Computer Communications. 2020;151:183–195.
Available:https://www.sciencedirect.com/science/article/pii/S014036641930492X
28. Maheswaran M, Ali S, Siegel H, Hensgen D, Freund RF. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. Journal of Parallel and Distributed Computing. 1999;59(2):107–131.
Available:https://www.sciencedirect.com/science/article/pii/S0743731599915812
29. Mishra SK, Sahoo B. Load balancing in cloud computing: A big picture. Journal of King Saud University - Computer and Information Sciences. 2020;32(2):149–158.
Available:https://www.sciencedirect.com/science/article/pii/S1319157817303361
30. Zhou Z, Li F, Zhu H, Xie H, Jemal HA, Morshed UC. An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments. Neural Computing and Applications, 2020;32(6):1531–1541.
Available:https://link.springer.com/article/10.1007/s00521-019-04119-7
31. Iosup A, Li H, Jan M, Anoep S, Dumitrescu C, Wolters L, Epema DHJ. The Grid Workloads Archive. Future Generation Computer Systems. 2008;24:672–686.

Available:https://doi.org/10.1016/j.future.2008.02.00Bell G.

32. Bell's law for the birth and death of computer classes: A theory of the computer's evolution. IEEE Solid-State Circuits Society Newsletter. 2008;13 (4):8–19.

Peer-review history:
The peer review history for this paper can be accessed here:
http://www.sdiarticle4.com/review-history/67791