ISSN: 0975-833X

Available online at http://www.journalcra.com

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

International Journal of Current Research
Vol. 13, Issue, 06, pp.17736-17744, June, 2021

DOI: https://doi.org/10.24941/ijcr.41616.06.2021

REVIEW ARTICLE **OPEN ACCESS**

PARALLEL SCHEDULING OF GRID JOBS ON DUOCORE SYSTEMS USING GROUPING METHOD

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

Computer Science Department, Faculty of Science, Niger Delta University, Yenagoa, Nigeria[1*]
Mathematics Department, Faculty of Science, Niger Delta University, Yenagoa, Nigeria[2]

ARTICLE INFO

Article History:
Received 14th March, 2021
Received in revised form
25th April, 2021
Accepted 28th May, 2021
Published online 26th June, 2021

Key Words:
Multicore, Parallelism,
Multi-scheduling, Machine Grouping, Job Grouping, Parallel-Scheduling.

Corresponding author:
Dr. Abraham, T. Goodhead

ABSTRACT

Grid computing has continued to gain applicability in various spheres of computing while multicore computers are also becoming ubiquitous. Most grid scheduling algorithms remain sequential while several attempts at parallelizing grid scheduling rely on the underlying hardware. To leverage the grid to meet the growing global computing need, a method to increase the efficiency of grid scheduling on parallel systems is required. This work aims at enhancing the parallel scheduling of Grid jobs on a duocore system. An arbitrary method was employed to group machines, a summation of the total processing power of machines in each group was computed. Then, a size (of jobs) proportional to processing power (of machines) was used to allocate jobs to machine groups. The MinMin scheduling algorithm was implemented in parallel (multi-scheduling) within the groups and also implemented without the group method to schedule the same range of jobs on a single processor machine and on a duocore machine. A performance improvement of 16%, 49%, and 71% was recorded on the duocore system by the group method over the ordinary MinMin method using two, four, and eight groups respectively. We find that the group method increased the performance of the scheduling algorithm on duocre systems. Secondly, we find that the ordinary MinMin algorithm benefited from the underlying parallelism of the duocore but not as much as the group method. Thirdly, we find that performance of the scheduling algorithm increases as the number of groups increases. We conclude that job grouping and multi-scheduling enhance performance significantly on the ducore system.

Citation: Goodhead T. Abraham and Evans, F, Osaisai. "Parallel scheduling of grid jobs on duocore systems using grouping method", *2021. International Journal of Current Research, 13, (06), 17736-17744.*

INTRODUCTION

Multicore systems are increasingly becoming prevalent due to gains made in hardware computing technology and Grid computing has been acclaimed as the paradigm to solve the ever-increasing computing need of an ever-demanding world (1) and (2). This means more grid infrastructures will be comprised of multicore systems in the future and the need to exploit the gains of multicore systems to the benefit of the grid is necessary. Scheduling algorithms in grid computing have largely remained sequential; there by falling short of maximizing the gains associated with parallel systems.

Corresponding author:** ***Goodhead T. Abraham,
Computer Science Department, Faculty of Science, Niger Delta University, Yenagoa, Nigeria.

Secondly, most grid scheduling researches are concerned with scheduling parallel jobs rather than scheduling jobs in parallel, and thirdly, several attempts at parallelizing scheduling algorithms mainly rely on the underlying hardware, and also parallelizing sequential algorithms is not completely attainable due to bottlenecks from the non-parallelizable sections. Multicore systems provide the solution to the growing processing need of man, for the grid to achieved its numerous computing goal, a concerted effort aimed at exploiting the benefits of multicore systems is required. This paper seeks to harness the benefits of multi-core systems by employing grouping methods to improve parallelism in the scheduling of Grid jobs on a duocore machine. The method randomly selects machines into a group, then sums the total processing power of all machines in each group, a job balancing algorithm uses the ratio of the total processing power of the machine group to determine and allocate grid jobs to the machine groups. In

varied experiments, the MinMin algorithm is implemented using the group method and also implemented separately on a single processor machine and also on a duocore machine and the length of scheduling recorded. The remainder of the paper is organized as follows: The next section discusses related literature. The proposed method, experimental setup, results and analysis is discussed after the literature. This is followed by recommendation. Then the last section discusses conclusion and thoughts for future work.

Related Work

This section discusses related work.

The Multicore Era: The continued growth of computer hardware technology was predicted by (3) while (4) hinted that limiting factors on serial computing would impede the continuous production of serial computers. This led (5) and (6) to predict the death and 'level off' of the Moore's law. These predictions changed the direction of computer growth towards multicore technology which is seen as the dominant choice for modern computing platforms now and in the near future (6)–(11). Advances in computing technology come with improved performance, speed, efficiency, and increased throughput (12). However, it has been shown that sequential algorithms do not completely gain from parallel systems and secondly, execution of sequential algorithms impedes performance on parallel systems (13) and (14). To leverage the grid for the growing computing need, in tandem with the global goals for innovative and sustainable development, a method that exploits the efficiency of parallel systems in grid scheduling is required. Several researchers including (15), (16), (17), and (18) proposed a total change in programming models – from sequential to parallel programming. Therefore, multicore computing calls for a new way of programming and software development.

The Grid: The grid is composed of an aggregate of powerful super computers, multiple computing clusters, large scale distributed networks and the connectivity of several other heterogeneous resources operating on diverse owner policies. (19) opined that the Grid can change the way complex problems are solved, provide large-scale aggregation and sharing of computational data and other resources across institutional boundaries or virtual organizations. For these goals to be attained, the potentials of the grid must be properly harnessed and an efficient scheduling paradigm be exploited (20) and (21).

Some Parallel Scheduling Attempts: (22) noted that every major advance in computing technology comes with a change in programming model, the birth of the multicore has led to many researchers attempting to parallelize processing and scheduling algorithms by directly executing sequential algorithms on parallel systems. (23) implemented CPU and GPU multi-threaded parallel designs of the MinMin algorithm. (24) implemented a memetic algorithm that combines non-deterministic approaches to improve speed up in scheduling in a GPU environment. (25) presented an online algorithm with a competitive ratio that minimized the maximum machine load on three hierarchical machines. (26) implemented a parallel fixed-point algorithm to find the solution of the total variation model for phase demodulation on multi-core CPU and GPU using OpenMP and CUDA and achieved up to 9x on multi-core and 103x on GPU.

(27) implemented the affinity-based scheduling using Bayesian analysis model and groups or clusters of dependent tasks and achieved improvement of 5.57% to 9.05%. This work is an effort aimed at investigating further the effect of increasing the number of cores in relation to increasing groups and how this might affect the parallel scheduling of Grid jobs.

Group Scheduling: Group scheduling has been exploited differently in researches, (28) used gang scheduling to execute different parts of cooperating processes in a multiprocessor system. (29) grouped jobs before transmitting to Grid resources for computation. (30) explored the advantages of grouping *light-weight* or small jobs to *coarse-grain* jobs before scheduling to reduce the communication computation ratio (CCR). (31) exploited CPU-GPU computer architecture to speedup queries from high-dimensional holistic data cubes. (32) code transformation exposes data parallelism latent in task-parallel applications and enhance resource utilization and speedup on chips to facilitate parallelism on the hardware. (33) used the Bees Algorithm (BA) to minimize makespan on resource-constrained project scheduling problem (RCPSP) but the method was narrowed to RCPSPs. (34) applied Inter Linear Programming strategy to schedule Precedence-Constrained Task Graphs (PTGs) and attained significant improvement in scheduling. (35) implemented a recursive divide and conquer algorithm that improved probability and efficiency for a class of dynamic programming problems in multicore systems. (36) proposed the multi-functional based scheduling framework to solve the duplication-based scheduling problem, the work successfully improved the schedule length and optimized energy consumption in the multicore system.

Most of these works were executed on parallel systems but restricted to specific problem areas. However, the methods rely solely on the underlying hardware for parallelism as no known method was implemented to parallelise the scheduling. This research employs grouping not as a means of turning fine-grain jobs into coarse-grain jobs but to help create platforms to independently schedule jobs in parallel, taking advantage of the multicores. Parallelizing sequential codes is quite a difficult task as most algorithms are sequential, however, some attempts have been made to improve performance on parallel systems. (37) executed a method that improves parallelism by grouping machines and jobs before scheduling and achieved a good result in scheduling, but the group cardinality was not dynamic, and this affected the efficiency of the results. (38) then exploited a method that varied the number of groups and attained improved scheduling efficiency. They however recommended that the effect of increasing the (CPUs) was necessary to be investigated further. (39) also exploited various methods for grouping grid jobs and machines before multi-scheduling on HPC systems.

Parallel Scheduling of Grid jobs on a duocore system using grouping method: This research proposes a job and machine group methods aimed at enhancing Grid scheduling on a duocore machine. The proposed method allows for the simultaneous execution of several instances of the MinMin algorithm within independent groups (multischeduling). The method employs a random method to group Grid machines. The total processing power (or processing speed) of machines in each group is determined. A *size_proportional_to_speed*method is used to assign jobs to machine groups, this is done to ensure a balanced distribution of jobs to machine groups. Several instances of the MinMin

algorithm are then implemented to independently schedule jobs to machine groups in parallel. The simultaneous execution of several scheduling instances within each group is referred hereto as multi-scheduling.

Size proportional to speed Grouping method: The size proportional to speed grouping method (SpdRnd) uses the total processing power of all machines in each group to compute a ratio used to determine the size of jobs to be assigned to each machine group. Based on the computed ratio, the first N jobs making up the ratio for the first group are allocated, then the next N jobs making up the ratio for the next group are allocated. This is continued until all jobs are allocated. The algorithm for estimating the size of jobs to be allocated to a group is shown in Table 1.

Table 1. Speed proportional to Job Allocation Method

Step1: Start
Step2: get sum of processing power of machines in each group Sum_{Groupi}
Step3: get cumulative processing power of machines in all groups Sum_{Total}
Step4: sum total size of all jobs Sum_{Jobs}
Step5: Estimate size of jobs to be allocated to each group $Sum_{Groupi} / Sum_{Total}$
Step5i: Allocate first N jobs of ratio to group i
Step5ii: Increment group count
Step5iii: allocate next N jobs of ratio to group i+1
Step6: Repeat Step5i to Step 5iii until all jobs are assigned
Step7: Stop

Machine Grouping: Table 2 shows the algorithm to group machines and compute the total processing power.

Table 2. Random Method to group machines and sum the processing power

Step1: Start
Step2: GroupTotal=0
Step3: SumTotal=0
Step4: For Group 1 to g (g is the number of job groups)
Step5: Randomly
a.Select a machine
b.Insert to a group (*groupi*)
c.Increase*group*totalwith machine processing power: Sum_{Groupi} + Machine
d.Increase Total with machine processing power Sum_{Total} +Machine
Step6: iterate step 5until all machines are assigned
Step7: Stop

Grid Machines: Each Grid site is made of hundreds to thousands of computing machines with different attributes. These are the machine identification (MId), speed of processor (SP), number of processor cores (NPC), and RAM size in KB or GB.

Grid Site: The Grid is characterized by different attributes ranging from the number of processing machines, policies, network bandwidth, computing resources, and configuration. The computing resources are also characterized by different attributes that differentiate them from others – like size, speed, and the number of cores or processors. Table 3 shows the attribute of the Grid site used in the simulation experiment.

Table 2 Composition of Grid site used in the simulation experiment

Features	Characteristics	Attributes
Network Bandwidth	The network bandwidth determines the speed of connection of the Grid site.	Network bandwidth is categorized into 4, 3, 2 and 1. Representing Super-Fast (SF), Very Fast (VF), Medium Fast (MF) and Not Fast (NF)
Number of Machines	This refers to the number of machines contained in the Grid site.	The number of machines was not categorized because they vary from time to time
Grid ID	This refers to the Grid site, the features that identify them. It can be a uniquely given number or a serial number.	Name or number or combination of both

Simulation of Grid, CPU Speed and Number of Cores: The Grid was simulated to compose: CPU; RAM; Bandwidth. A grid represented as {A; 2000; 5000000; 3000} translates to Grid site A, with 2000 CPUs, RAM size5000000, and 3000 Bandwidth. Table 4 shows the composition and characteristics of the machines simulated in each Grid site.

Table 4. Composition of machines in the Grid site

Grid Site/ No of Machines	Characteristics: No of machines	Speed of CPU	No of CPU/ Cores	Grid Site/ No of machines	Characteristics: No of machines	Speed of CPU	CPU/ Cores
A	30	4GHz	1	C	40	1.5GHz	2
240	20	3GHz	1	480	40	2GHz	2
	30	2GHz	1		50	3.5GHz	2
	20	1GHz	1		50	4GHz	2
	10	4GHz	2		70	1.5GHz	4
	10	3GHz	2		70	2GHz	4
	10	2GHz	2		80	3.5GHz	4
	10	1GHz	2		80	4MHz	4
B	60	3.5GHz	2	D	60	1.5GHz	2
400	60	4GHz	2	600	60	2GHz	2
	60	1.5GHz	2		60	3.5GHz	2
	60	2GHz	2		60	4GHz	2
	40	3.5GHz	4		60	1.5GHz	4
	40	4GHz	4		60	2GHz	4
	40	1.5GHz	4		40	3.5GHz	4
	40	2GHz	4		40	4GHz	4
					40	1.5GHz	8
					40	2GHz	8
					40	3.5GHz	8
					40	4GHz	8

Computing machine was simulated as: CORES; CPU; RAM. For instance {4; 3000; 4000000} refers to a Grid machine with 4 CPUs, 2000 MHz or 2GHz and 2000000B or 2MB.

Source of data: Jobs for the experiment was downloaded from the Grid workload archive provided by (40) for researchers and developers.

Experimental Design: A total of four experiments were carried out across the two computing platforms (a single processor machine and a duocore machine). In the first experiment, the ordinary MinMin algorithm was executed on a single processor system to schedule a range of jobs (from 1000 to 10000 in steps of 1000). The second experiment was executed on a duocore system.

It implemented the ordinary MinMin algorithm to schedule the same range of jobs as in the first experiment. The third experiment was executed on a single processor system. It used the random method to group machines and 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. The fourth experiment was executed on a duocore system, it used the random method to group machines and 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, several runs were made using 2, 4, and 8 groups in turn. For each group, the number of threads used was varied between 1, 2, 4, and 8. For each of the combinations, the time taken to schedule and the makespan for each variation was recorded.

System Properties: The configuration of the single processor and the duocore machines used for the experiment is shown in Table 5.

Table 3. Properties of the systems

Single processor system:	Duocore:
Processor:Intel(R) Pentium(R) 4 CPU 3.00GHZ3.00GHz RAM: 1.50 GB Operating System: Windows XP Professional Version 2002	Processor: Intel(R) Core(TM) i5-320M CPU @2.50 GHz 2.50 GHz RAM: 4.00 GB System Type:64-bit Operating System Operating System:Windows 7

RESULTS AND DATA ANALYSIS

This analysis compares the results obtained using the ordinary MinMin against the result from the group method. In the analysis; grp refers to Group, thrds refers to threads, Duo refers to duocore and proc refers to processor. The comparison is based mainly on improvement of the group method on the ordinary MinMin which is computed in multiple as:

Improvement in multiple (X) =

The improvement in percent is computed as:

Improvement in percent (%) =

((Min Min-Group)/Group)*100 ……. Equation 1

Performance of ordinary MinMin on the two computing platforms: This section discusses result of the ordinary MinMin (without the group method) on the single processor and the duocore machines. This experiment is like most other experiments where the sequential algorithm is executed on a parallel system. Table 6 shows the result and performance of the ordinary MinMin on the single processor machine and the duocore machine using two threads. The MinMin algorithm recorded a total scheduling time of 1235755Ms on the single processor machine and 220726 Ms on the duocore machine. This represents a performance improvement of 5.6 times or 82% by the duocore system over the single processor system. This improvement indicates that the MinMin algorithm is scalable to some extent and gained from the parallelism in the duocore system. the MinMin relied on the underlying hardware for improved performance on the duocore, but this does not represent the best performance of the MinMin on duocore as the analysis in the next two sections show.

Table 4. Result and performance of MinMin on a single processor machine and on a duocore machine

Jobs Limit	MinMin2Thrds(SingleCPU)	MinMin2Thrds (Duocore)
1000	4500	690
2000	17672	2800
3000	40250	6410
4000	66922	10674
5000	90407	26432
6000	116922	19275
7000	154781	24914
8000	196768	32446
9000	244679	44057
10000	302854	53028
Total	1235755	220726
Average	123575.5	22072.6
Improvement over Single processor system in multiples (X)		5.598593
Improvement over Single processor system in percent (%)		82.13837

Performance of the ordinary MinMin on the duocore against group method on the single processor machine: This analysis compares the best ordinary MinMin result obtained on the duocore machine against group method result on the single processor machine. The result is shown in Table 7 which also shows the formulas used in computing improvement while Table 8 shows the computed improvement and average improvements in multiples and percent. Using two groups and one, two, and four threads; the ordinary MinMin on duocore performed better than the group method on single processor by 4.37, 4.41, and 4.43 times or 77.13%, 77.32%, and 77.44% respectively. Using four groups, and one, two or four threads, the ordinary MinMin on duocore performed better than the group method on single processor by 2.36, 2.39, and 2.38 times or 57.56%, 58.10%, and 57.93% respectively. While using eight groups, there was an improvement of 1.53, 1.49, and 1.51 times or 34.80%, 32.94%, and 34.1% by the MinMin on duocore over the group method on the single processor. Figure 1 and Figure 2 show the performance and aggregate performance of the ordinary MinMin implemented on a duocore machine over the grouping method executed on a single processor machine. On average, there was an aggregate improvement of 77.30%, 57.86% and 33.95% (or 4.40, 2.37 and 1.57 times) by the ordinary MinMin over the group method-using two, four and eight groups respectively. The ordinary MinMin performed better on the duocore machine compared to the group method on a single processor system because of the parallelism on the duocore and lack of support for parallelism on the single processor machine. The trendline fitted through the aggregate improvement graph yielded:

$Y = -1.445x + 5.65$……Equation 2

The equation of the trendline indicates that even though the ordinary MinMin on the duocore machine performed better than the group method on the single processor machine, the group method was improving as the number of groups increases. The correlation (of 0.99) between the MinMin values and the group method values indicates a strong correlation and the R-squared value of 0.9482 (on the trendline) explains the reliability and repeatability of the results using the method.

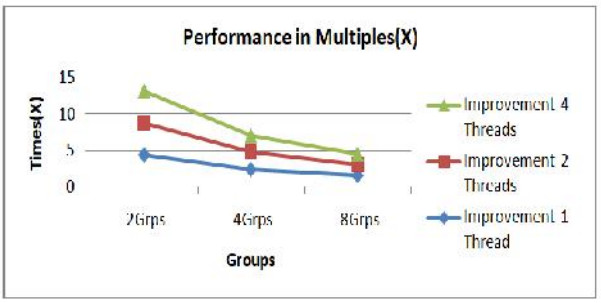

Figure 1. Performance of MinMin on duocore vs Group method on a single processor in multiples (X)

Figure 2. Aggregate Improvement of MinMin on Duocore against Group method on single processor

Analysis of Results on the Duocore Machine: This section presents the results and analysis of the experiment executed on the duocore machine. Table 9 shows the result and Table 10 shows the computation of improvement in multiples and in percent. With two groups, the group method performed better than the MinMin by 1.07 to 1.29 times or 6.8 to 22.26 percent. Using four groups, the group method performed better than the MinMin by 1.6 to 2.28 times representing 37.7 to 56 percent. While using eight groups, the group method performed better than the MinMin by 2.7 to approximately 4 .0 times which represents 63 to 75 percent. The improvement graph in Figure 3 shows that as the number of groups increases on the duocore, performance over the ordinary MinMin also increases. The trendline fitted through the aggregate improvement graph in Figure 4 yielded equation:

Y = 1.175x - 0.08 …… Equation 3

This indicates that the group method increases scheduling efficiency on the duocore machine as the number of groups increases. There is a strong correlation of 0.99 between pairs of results and the R-squared value of 0.97 on the trendline indicates that the equation represents a good fit and the result is reliable and repeatable.

Combination of results: This section shows the analysis between the best result of the ordinary MinMin on the duocore against the group method on the two platforms. Table 11 and Table 12 show the combination of aggregate performance in multiples and percent respectively, Figure 5 shows the total scheduling time using MinMin on the two platforms, while Figure 6 and Figure 7 show the combined aggregate performance between the single processor system and the duocore system.

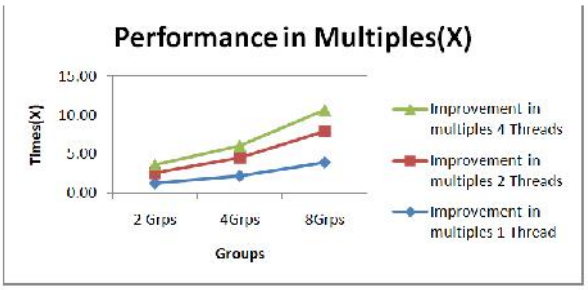

Figure 1. Improvement over MinMin (X)

Figure 4. Aggregate performance improvement on duocore

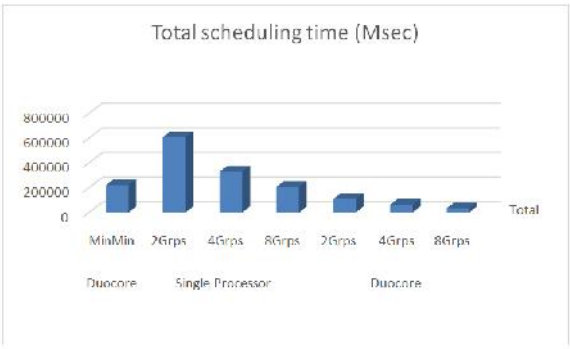

Figure 5. Total scheduling time of MinMin and group method on the two platforms

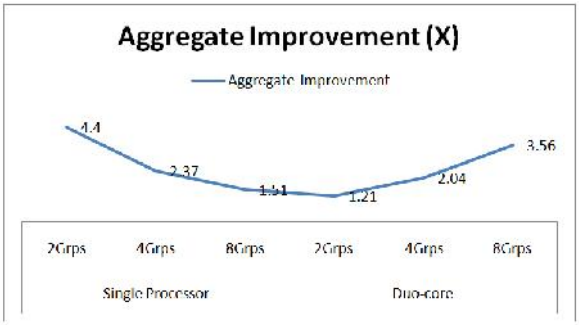

Figure 2. Aggregate performance

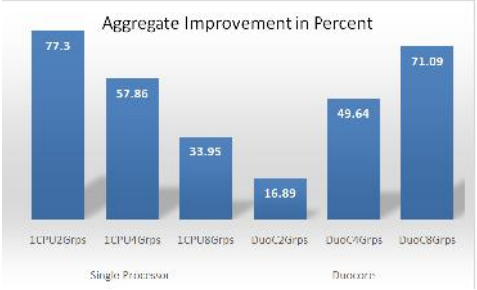

Figure 7. Aggregate improvement in percent

Table 7. Result from single processor system

No of jobs	One thread				Two threads			
		PropSpdRnd (Times Msec)				PropSpdRnd (Times Msec)		
	MinMin	2Grps	4Grps	8Grps	MinMin	2Grps	4Grps	8Grps
1000	452	2516	1062	547	403	2360	1094	594
2000	1924	7500	4110	2062	1775	7907	3876	2031
3000	4178	17265	7484	4406	4203	17031	9063	4360
4000	7118	27313	13453	7890	7431	28468	13656	9359
5000	10013	39406	20454	13516	9903	40453	22672	12985
6000	12825	57610	27937	21453	12984	59781	32578	19062
7000	16685	79407	42266	25218	16872	79499	42047	26234
8000	21633	94140	53375	37125	21956	94781	54172	34453
9000	27624	123121	66657	42843	27778	119609	67438	41250
10000	34406	150262	85688	54860	34570	157970	82423	55282
Total	136858	598540	322486	209920	137875	607859	329019	205610
Average	13685.8	59854	32248.6	20992	13787.5	60785.9	32901.9	20561
	Four threads							
		PropSpdRnd (Times Msec)						
	MinMin	2Grps	4Grps	8Grps				
1000	399	2234	1047	562				
2000	1795	9125	3969	2016				
3000	4222	15922	8140	4296				
4000	7188	28437	14906	8516				
5000	9994	41250	20438	13406				
6000	12700	60438	29188	19860				
7000	16690	79031	41157	25125				
8000	21719	95032	53329	35891				
9000	27989	126173	66907	44157				
10000	34122	148689	86110	53781				
Total	136818	606331	325191	207610				
Average	13681.8	60633.1	32519.1	20761				

(1) Performance improvement in multiple (X) = $\frac{Group\ \text{Re}\ sult}{MinMin\ \text{Re}\ sult}$

(2) Performance Improvement in percent (%) $\frac{Group\,\text{Re}\,sult - MinMin\,\text{Re}\,sult}{Group\,\text{Re}\,sult} x100$

Table 5. Performance on the Single processor system

		Spd Rnd						
		Performance in Percentage				Performance in Multiples		
	Improvement	2Grps	4Grps	8Grps		2Grps	4Grps	8Grps
1	1 Thread	77.13	57.56	34.8		4.37	2.36	1.53
2	2 Threads	77.32	58.10	32.94		4.41	2.39	1.49
3	4 Threads	77.44	57.93	34.1		4.43	2.38	1.51
	Average Improvement	77.30	57.86	33.95		4.40	2.37	1.51

Table 9. Result from Duo-core system

Number of Jobs	One Thread				Two Threads				Four Threads			
	MinMin	SpdRnd			MinMin	SpdRnd			MinMin	SpdRnd		
		2Grps	4Grps	8Grps		2Grps	4Grps	8Grps	MinMin	2Grps	4Grps	8Grps
1000	452	394	277	106	403	375	319	95	399	447	232	126
2000	1924	1418	811	245	1775	1360	617	331	1795	1390	1971	447
3000	4178	2809	1544	720	4203	2628	1334	609	4222	2782	1926	1114
4000	7118	4766	2540	1232	7431	4665	2138	1575	7188	5209	3310	2232
5000	10013	6824	4035	1847	9903	7680	3832	2323	9994	8136	5626	3311
6000	12825	9963	5864	2944	12984	9607	5481	2927	12700	10064	7779	4729
7000	16685	13424	8039	4772	16872	14322	7888	4864	16690	14411	11354	6867
8000	21633	17362	10120	5567	21956	17156	9967	5672	21719	19857	14663	8138
9000	27624	22578	12738	7609	27778	22976	12796	7273	27989	27558	18172	11557
10000	34406	26853	15792	9590	34570	27406	15921	9080	34122	37564	20116	11039
Total	136858	106391	61760	34632	137875	108175	60293	34749	136818	127418	85149	49560
Average	13685.8	10639.1	6176	3463.2	13787.5	10817.5	6029.3	3474.9	13681.8	12741.8	8514.9	4956

Table 10. Performance analysis on the Duo-core system

		Speed Proportional and Randomly						
		Improvement in Percentage				Improvement in Multiples		
	Improvement	2Grps	4Grps	8Grps		2Grps	4Grps	8Grps
1	1 Thread	22.26	54.87	74.69		1.29	2.22	3.95
2	2 Threads	21.54	56.27	74.80		1.27	2.29	3.97
3	4 Threads	6.87	37.76	63.78		1.07	1.61	2.76
	Average Improvement	16.89	49.64	71.09		1.21	2.04	3.56

Table 6. Combined Improvement Analysis in multiples

Combined Improvement Analysis in Multiples						
	Single Processor			Duo-core		
	2Grps	4Grps	8Grps	2Grps	4Grps	8Grps
1 Threads	4.37	2.36	1.53	1.29	2.22	3.95
2 Threads	4.41	2.39	1.49	1.27	2.29	3.97
4 Threads	4.43	2.38	1.52	1.07	1.61	2.76
Aggregate Improvement over MinMin	4.4	2.37	1.51	1.21	2.04	3.56

Table 7. Combined Improvement Analysis in percentage

Combined Improvement Analysis in Percentage						
	Single Processor			Duo-core		
	2Grps	4Grps	8Grps	2Grps	4Grps	8Grps
1 Threads	77.13	57.56	34.8	22.26	54.87	74.69
2 Threads	77.32	58.1	32.94	21.54	56.27	74.8
4 Threads	77.44	57.93	34.1	6.87	37.76	63.78
Aggregate Improvement	77.3	57.86	33.95	16.89	49.64	71.09

On the single processor machine, the MinMin performed better than the group method by 4.4 times using two groups, 2.37 times using four groups and 1.51 times using eight groups. The performance level decreased as the number of groups increases, this is shown in the falling part of the graph labelled single process 2 grps to 8 grps in Figure 6 and the part labelled Single processor in Figure 7. On the Duocore, the group method performed better than the MinMin by 1.21, 2.04 and 3.56 times (or 16%, 49% and 71%) using two, four and eight groups respectively. This caused the aggregate graph to rise from duocore (two groups) to duocore (eight groups) in Figure 6 and Figure 7.

General Discussion on the results

There was a general performance improvement over the ordinary MinMin using the group method. However, the group method underperformed on the single processor machine, this is because of the lack of parallelism support needed to execute multiple scheduling instances within the groups. The ordinary MinMin executed on a duocore performed better than the group method executed on a single processor system, this indicates that the MinMin algorithm is also scalable to some extent while the group method is parallel-centric. On the duo-core system, the group method performed better than the ordinary MinMin, the performance over the MinMin increased as the number of groups increases. Both the MinMin and group method exploited the parallelism on the duocore but the group method enhances parallelism far more than the MinMin, ensuring a significant performance improvement over the MinMin. Although, all the machines are not timed at the same speed, it can be deduced that the grouping method expands the realm of parallelism as scheduling throughput increases as the number of groups increases. The correlation of 0.99 between the sets of results indicates that the results are reliable and repeatable. The R-squared value of 0.97 shows that the trend generated by the result and analysis represents the best fit.

Recommendations

It has been proven severally that sequential algorithms do not fully utilize the underlying hardware. To achieve better performance, we recommend that grouping methods be integrated with grid schedulers.

CONCLUSION

This work explored a job group and machine group method to enhance parallelism in the scheduling of Grid jobs on a duocore machine. The group methods improved performance by 1.21 to 3.56 times (or by 16 to 71%). The improvement was directly related to increasing groups. It has been proven that sequential algorithms do not scale with multicore systems due to performance-limiting sections. Multicore systems on the other hand do not automatically guarantee optimal performance with sequential algorithms. The dawn of the multicore era, therefore, calls for a different approach to programming models. To leverage the Grid for future goals and benefit from improvements in hardware technology, we propose the use of job and machine grouping methods for increased performance.

Future thoughts: The experiments were conducted on a single processor system and a duocore system and the pattern of improvement was investigated. Though different factors characterized the performance of a system, the overall result cannot be 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 recorded significant improvement over the MinMin on a duocore system, it will be interesting to test the efficacy of the group method on computers with more cores (probably a quadcore) and see the percentage of improvement; this will reveal if the improvement pattern is continuous and by how much.

Acknowledgement

I acknowledge the Faculty of Engineering and Computing, Coventry University, United Kingdom for award of research grant in 2015.

REFERENCES

(1) Chervenak, A. I. Foster, C. Kesselman, C. and Salisbury, and S. Tueke, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets," *Journal of Network and Computer Applications*, vol. 23, no. 3, pp. 187–200, 2000, Accessed: Mar. 04, 2020. (Online). Available: https://www.sciencedirect.com/science/article/pii/S1084804500901103.

(2) Klusáček, D. H. Rudová, R. Baraglia, M. Pasquali, and G. Capannini, "Comparison Of Multi-Criteria Scheduling Techniques," in *Grid Computing*, Springer US, 2008, pp. 173–184.
(3) Moore, G. "Cramming more components onto integrated circuits," pp. 114–117, 1965.
(4) W. Knight, "Two heads are better than one (dual-core processors)," *IEE Review*, vol. 51, no. 9, pp. 32–35, 2005, Accessed: Mar. 05, 2020. (Online). Available: https://digital-library.theiet.org/content/journals/10.1049/ir_20050903.
(5) Eck, D. J. "Introduction to Programming Using Java," 2006. Accessed: Mar. 05, 2020. (Online). Available: http://math.hws.edu/javanotes/.Thewebsiteincludes.
(6) M. Kaku, "Tweaking Moores's law: Computers of the post-silicon era," 2013, Accessed: Mar. 05, 2020. (Online).
(7) Gepner P. and Michal. F. Kowalik, "Multi-core processors: New way to achieve high system performance," in *International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)*, 2006, pp. 9–13, Accessed: Mar. 05, 2020. (Online). Available: https://ieeexplore.ieee.org/abstract/document/1698630/.
(8) Meyer, R. "Emerging Multi-core Realities," *Scientific Computing*, 2006, Accessed: Mar. 23, 2020. (Online).
(9) Lin, J. C. Yu, L. Wenlong, J. and Aamer, and T. Zhizhong, "Understanding the memory behavior of emerging multi-core workloads," in *2009 International Symposium on Parallel and Distributed Computing*, 2009, pp. 153–160, Accessed: Mar. 05, 2020. (Online). Available: https://ieeexplore.ieee.org/abstract/document/5284359/.
(10) Schauer, B. "Discovery Guides Multicore Processors-A Necessity," 2008. Accessed: Mar. 05, 2020. (Online). Available: http://www.netrino.com/node/91.
(11) Zhuravlev, S. J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of scheduling techniques for addressing shared in multicore processors," *ACM Reference Format*, vol. 45, no. 4, Nov. 2012, doi: 10.1145/2379776.2379780.
(12) Kessler, C. U. Dastgeer, and L. Li, "Optimized Composition: Generating Efficient Code for Heterogeneous Systems from Multi-Variant Components, Skeletons and Containers," *Skeletons and Containers. arXiv preprint arXiv*, vol. 1405.2915, May 2014, Accessed: Mar. 06, 2020. (Online). Available: http://arxiv.org/abs/1405.2915.
(13) Amdahl, G. M. "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS Conference Proceedings - 1967 Spring Joint Computer Conference, AFIPS 1967*, Apr. 1967, pp. 483–485, doi: 10.1145/1465482.1465560.
(14) Adams, J. C. D. J. Ernst, T. Murphy, and A. Ortiz, "Multicore education: Pieces of the parallel puzzle," in *SIGCSE'10 - Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 2010, pp. 194–195, doi: 10.1145/1734263.1734329.
(15) Tendulkar, P. "Mapping and Scheduling on Multi-core Processors using SMT Solvers," 2014.
(16) Jin, H. D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, and B. Chapman, "High performance computing using MPI and OpenMP on multi-core parallel systems," *Parallel Computing*, vol. 37, no. 9, pp. 562–575, 2011, doi: 10.1016/j.parco.2011.02.002.
(17) Asanovic K. *et al.*, "A view of the parallel computing landscape," *Communications of the ACM*, vol. 52, no. 10, pp. 56–67, Oct. 2009, doi: 10.1145/1562764.1562783.
(18) Chaiken R. *et al.*, "SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1265–1276, 2008, Accessed: Mar. 06, 2020. (Online). Available: https://dl.acm.org/citation.cfm?id=1454166.
(19) Foster I. and Kesselman, C. "*The Grid: Blueprint for a new computing infrastructure.*" San Francisco: Morgan Kaufmann, 1999.
(20) Foster, I. "Internet computing and the emerging grid," *Nature web matters*, vol. 7, 2000.
(21) Sajedi H. and M. Rabiee, "A metaheuristic algorithm for job scheduling in grid computing," *International Journal of Modern Education and Computer Science*, vol. 6, no. 5, p. 52, 2014.
(22) Bell, G. "Bell's law for the birth and death of computer classes: A theory of the computer's evolution," *IEEE Solid-State Circuits Society Newsletter*, vol. 13, no. 4, pp. 8–19, 2008.
(23) Nesmachnow S. and M. Canabé, "GPU implementations of scheduling heuristics for heterogeneous computing environments."
(24) Pinel, F. B. Dorronsoro, and P. Bouvry, "Solving very large instances of the scheduling of independent tasks problem on the GPU," *Journal of Parallel and Distributed Computing*, vol. 73, no. 1, pp. 101–110, 2012, doi: 10.1016/j.jpdc.2012.02.018.
(25) Xiao, M. L. Ding, S. Zhao, and W. Li, "Semi-online Algorithms for Hierarchical Scheduling on Three Parallel Machines with a Buffer Size of 1," in *Communications in Computer and Information Science*, Nov. 2021, vol. 1352 CCIS, pp. 47–56, doi: 10.1007/978-981-16-1877-2_4.
(26) Hernandez-Lopez, F. J. R. Legarda-Sáenz, and C. Brito-Loeza, "Parallel algorithm for fringe pattern demodulation," *Journal of Real-Time Image Processing*, pp. 1–11, Jun. 2021, doi: 10.1007/s11554-021-01129-4.
(27) Abbasi S. I. and S. Kamal, "Affinity Based Scheduling Using Bayesian Model and Load Balancing in Multicore Systems," in *2021 International Conference on Digital Futures and Transformative Technologies (ICoDT2)*, May 2021, pp. 1–7, doi: 10.1109/ICoDT252288.2021.9441513.
(28) Ousterhout, J. "Scheduling Techniques for Concurrent Systems," *ICDCS*, vol. 82, pp. 22–30, 1982.
(29) Buyya, R. S. Date, Y. Mizuno-Matsumoto, S. Venugopal, and D. Abramson, "Neuroscience instrumentation and distributed analysis of brain activity data: A case for eScience on global Grids," *Concurrency Computation Practice and Experience*, vol. 17, no. 15, pp. 1783–1798, Dec. 2005, doi: 10.1002/cpe.888.
(30) Soni, V. K. R. and Sharma, and K. Mishra, Manoj, *Grouping-based job scheduling model in grid computing*, vol. 41. 2010.
(31) Silva, L. "Computing data cubes over GPU clusters.," 2018, Accessed: Mar. 04, 2020. (Online). Available: https://www.monografias.ufop.br/handle/35400000/1527.
(32) Bin, R. E. N. S. Balakrishna, Y. Jo, S. Krishnamoorthy, K. Agrawal, and M. Kulkarni, "Extracting SIMD parallelism from recursive task-parallel programs," *ACM Transactions on Parallel Computing*, vol. 6, no. 4, pp. 1–37, Dec. 2019, doi: 10.1145/3365663.
(33) Nemmich, M. A. F. Debbat, and M. Slimane, "(on-line) ©IBERAMIA and the authors An Enhanced Discrete Bees Algorithms for Resource Constrained Optimization Problems," *Inteligencia Artificial*, vol. 22, no. 64, pp. 123–134, 2019, doi: 10.4114/intartf.vol22iss64pp123-134.
(34) Roy, S. K. R. Devaraj, A. Sarkar, K. Maji, and S. Sinha, "Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous

distributed systems," *Journal of Systems Architecture*, vol. 105, p. 101706, May 2020, doi: 10.1016/j.sysarc.2019.101706.

(35)M. M. Javanmard, Z. Ahmad, M. Kong, L.-N. Pouchet, R. Chowdhury, and R. Harrison, "Deriving parametric multi-way recursive divide-and-conquer dynamic programming algorithms using polyhedral compilers," in *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization*, Feb. 2020, pp. 317–329, doi: 10.1145/3368826.3377916.

(36)Q. Tang, L.-H. Zhu, J. Lian, L. Zhou, and J.-B. Wei, "An efficient multi-functional duplication-based scheduling framework for multiprocessor systems," *The Journal of Supercomputing*, pp. 1–26, Feb. 2020, doi: 10.1007/s11227-020-03208-y.

(37)G. T. Abraham, A. and James, and N. Yaacob, "Priority-grouping method for parallel multi-scheduling in Grid," *Journal of Computer and System Sciences*, vol. 81, no. 6, pp. 943–957, 2015, doi: 10.1016/j.jcss.2014.12.009.

(38)G. T. Abraham, A. and James, and N. Yaacob, "Group-based Parallel Multi-scheduler for Grid computing," *Future Generation Computer Systems*, vol. 50, pp. 140–153, 2015, doi: 10.1016/j.future.2015.01.012.

(39) Abraham, G. T. "Group-based parallel multi-scheduling methods for grid computing," Coventry University, 2016.

(40) Iosup A. *et al.*, "The Grid Workloads Archive," *Future Generation Computer Systems*, vol. 24, pp. 672–686, 2008, doi: 10.1016/j.future.2008.02.003.

****