Load Balancing of Parallel Simulated Annealing on a Temporally Heterogeneous Cluster of Workstations

Sourabh Moharil and Soo-Young Lee
Department of Electrical and Computer Engineering
Auburn University, Auburn, AL 36849
leesooy@eng.auburn.edu

Abstract

Simulated annealing (SA) is a general-purpose optimization technique widely used in various combinatorial optimization problems. However, the main drawback of this technique is a long computation time required to obtain a good quality of solution. Clusters have emerged as a feasible and popular platform for parallel computing in many applications. Computing nodes on many of the clusters available today are temporally *heterogeneous. In this study, multiple Markov chain (MMC) parallel simulated annealing (PSA) algorithms have been implemented on a temporally heterogeneous cluster of workstations to solve the graph partitioning problem and their performance has been analyzed in detail. Temporal heterogeneity of a cluster of workstations is harnessed by employing static and dynamic load balancing techniques to further improve efficiency and scalability of the MMC PSA algorithms.*

1. Introduction

Simulated annealing (SA) [1] is a stochastic optimization algorithm to search a solution space of a combinatorial problem with the objective to minimize a cost function. Its generality and applicability stem from its foundation in thermodynamics and statistical mechanics, and its hill-climbing capability enables escape from local minima. However, the main drawback of SA is that it takes a long time to find a global or acceptable solution. There have been many efforts to develop parallel SA (PSA) algorithms. The PSA algorithms developed so far can be classified into two groups, i.e., single Markov chain (SMC) PSA and multiple Markov chain PSA [2]-[9].

It has been observed that performance of the MMC PSA is more consistent in terms of speedup achieved as compared to the SMC PSA. This approach of MMC PSA can exploit large-scale parallelism to achieve a high speedup with no loss in quality of solution [3, 5, 7]. Also, performance of the MMC PSA is much less problem-dependent, compared to the SMC PSA. However, there is no implementation of MMC PSA (to solve the graph partitioning problem) on a coarse-grain temporally heterogeneous cluster of workstations inspite of the fact that such a system has emerged as a feasible alternative for parallel computing in many applications. Hence, in this study, it is attempted to improve efficiency and scalability of PSA on a temporally heterogeneous cluster of workstations by incorporating simple but effective static and dynamic load balancing techniques that exploit the *random nature* of the problem (SA).

This paper is organized as follows. In Section 2, SA algorithm is described. In Section 3, the graph partitioning problem is reviewed. In Section 4, the implementation details and load balancing schemes for non-interacting, interacting and asynchronous MMC PSA's are described. In Section 5, the experimental results obtained on a Sun MPI cluster are provided In Section 6, a summary is provided.

2. SA Algorithm

A typical SA algorithm consists of two nested loops. The annealing process starts with "heating" the initial solution, s_0, to a melting point or an initial temperature, T_{init}. The outer loop decrements the annealing temperature depending on a factor α where $0 < \alpha < 1$. At a given temperature of SA, a number of perturbations are carried out until the inner loop break conditions are met. The inner loop break conditions are set depending on the type of combinatorial problem. The *cooling schedule* defines quantitatively how the temperature is lowered during the course of SA. Two parameters which determine the length of an outer loop are *maximum inner loop iterations*, ILI_{max}, (the maximum number of perturbations allowed at each temperature), and *maximum inner loop rejections*, ILR_{max}, (the maximum number of consecutive rejections of candidate solutions allowed at each temperature). Outer loop iterations continue until the annealing temperature reaches the freezing point T_{frz}. A typical high-level construct of SA is as follows:

1-4244-0910-1/07/$20.00

$s \leftarrow s_0$;
$T \leftarrow T_{init}$;
while($T \geq T_{frz}$) {
  while(inner loop break condition is not met){
    Perturb the current configuration s to s^*;
    Evaluate cost function $C(\ )$ and $\Delta C = C(s^*) - C(s)$;
    Accept the new configuration with the probability of
    $\min(1, e^{-\Delta C/T})$;
  }
  $T \leftarrow \alpha \times T$;
}

3. Graph Partitioning

The combinatorial optimization problem chosen in this study is *graph partitioning*. Many scientific and engineering applications can be modeled as graph partitioning problems [5]. Obtaining an optimal partitioning is NP-complete. In its most general form, the graph partitioning problem is to divide vertices in a graph into a specified number of subgraphs such that a certain cost function is minimized. An example of typical cost functions is the weighted sum of the maximum number of nodes in a subgraph and the number of edges between subgraphs.

4. MMC PSA

In the multiple Markov chain PSA (MMC PSA), all processes follow their own search paths. Each process, independent of other processes, perturbs the current solution, evaluates the perturbed solution and decides on its acceptance or rejection. Three versions of MMC PSA have been implemented in this study, i.e., non-interacting, interacting (synchronous) and asynchronous MMC PSA's. Depending on the load balancing scheme (SLB: static load balancing, DLB: dynamic load balancing) incorporated, each version has two or three variations as summarized in Table 1.

	Scheme
SQ	Sequential SA
NI	Non-interacting MMC PSA with no LB
NS	Non-interacting MMC PSA with SLB
IN	Interacting MMC PSA with no LB
IS	Interacting MMC PSA with SLB
IA	Interacting MMC PSA with DLB
AY	Asynchronous MMC PSA with no LB
AS	Asynchronous MMC PSA with SLB
AA	Asynchronous MMC PSA with DLB

Table 1. MMC PSA schemes implemented.

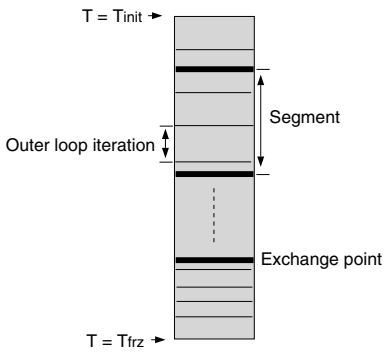


Figure 1. Segments in interacting and asynchronous MMC PSA's.

For the interacting and asynchronous MMC PSA's, a *segment* is defined as the duration between two successive solution exchanges (interacting MMC PSA) or global state accesses (asynchronous MMC PSA) as illustrated in Figure 1, which may be quantified in terms of the number of temperature decrements or outer loop iterations. The segment length is controlled such that it decreases as temperature decreases. This control causes the processes to interact more frequently at a higher temperature where the solution is highly unstable and less frequently at a lower temperature where the solution tends to converge. For non-interacting MMC PSA, a segment corresponds to the duration of an outer loop iteration (between two successive temperature decrements).

A cluster of workstations may include heteregenous computing elements. Also, the workload varies with time and workstation since a cluster is usually shared by multiple users and a user may submit task(s) at any time. Such spatial and temporal heterogeneity necessitates load balancing in order to utilize such a cluster optimally.

In this study, ILI_{max} (the maximum number of inner loop iterations) is used as the *load balancing parameter* to be controlled for static and dynamic load balancing since ILI_{max} determines an outer loop break condition at *most* of the temperature decrements. On the other hand, ILR_{max} (the maximum number of inner loop rejections) determines an outer loop break condition only when temperature is close to the freezing point (T_{frz}) or when the solution is converging to an optimal solution and, therefore, is not referred to in most cases.

4.1. Non-interacting MMC PSA

In non-interacting MMC PSA illustrated in Figure 2, each process independently follows its own Markov chain to find a solution. At the end, the best of all the solutions - the one with minimum cost, is retained and the others are

discarded. The processes do not interact with each other during the annealing process except for exchanging local solutions at the end when faster processes wait for slower processes, leading to idle or wait times. In Figure 2, Process 2 is the slowest of all the processes while Process 4 is the fastest.

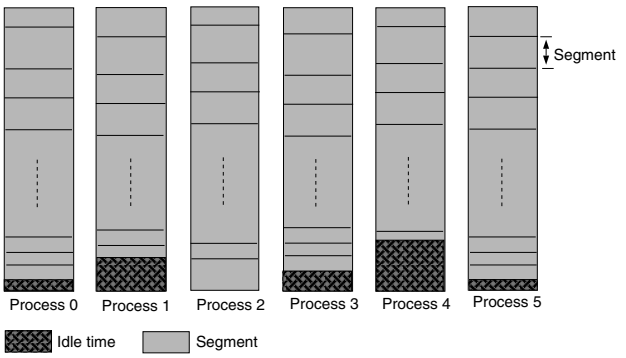


Figure 2. Non-interacting MMC PSA.

4.1.1. Implementation Process 0 reads in a graph to be partitioned, carries out the initial random partitioning and broadcasts it to all other processes. Then, all processes simultaneously follow their own Markov chains for optimizing the partitioning of the graph. In order to achieve a significant speedup, ILI_{max} and ILR_{max} are controlled as a function of the number of processes, n, such that :

$$ILI_{max_n} = \frac{ILI_{max\_seq}}{n} \tag{1}$$

and

$$ILR_{max_n} = \frac{ILR_{max\_seq}}{n} \tag{2}$$

where ILI_{max_seq} and ILR_{max_seq} are for the sequential SA, and ILI_{max_n} and ILR_{max_n} for the PSA. The other parameters, T_{init}, T_{frz} and α are set the same as for the sequential SA.

4.1.2. Static Load Balancing In this version, static load balancing is performed at the end of the first segment. The average execution time per inner loop iteration, measured by each process (Process j) in the first segment, t_{1j}, is used for static load balancing. Let l_{ij} and m_{ij} denote the execution time and the total number of inner loop iterations in segment i for Process j, respectively. Then,

$$t_{1j} = \frac{l_{1j}}{m_{1j}} \tag{3}$$

All processes send t_{1j} to Process 0 which in turn broadcasts $t_{min} = min_j\{t_{1j}\}$. On receiving t_{min}, each process (Process j) estimates its computing power S_j and that of the fastest process S_J as $S_j = \frac{1}{t_{1j}}$ and $S_J = \frac{1}{t_{min}}$, respectively. Then, it adjusts its ILI_{max_n} as follows.

$$ILI'_{max_j} = ILI_{max_n} \times \frac{S_j}{S_J} \tag{4}$$

The adjusted value, ILI'_{max_j}, dictates the annealing process in Process j. The reason for using t_{min} is to prevent non-interacting Markov processes from getting too lengthy.

4.2. Interacting MMC PSA

In the interacting MMC PSA, all processes interact with each other to exchange their intermediate solutions at the end of each segment. After each exchange, the intermediate solution with the least cost is retained by all processes. Thus, a process with an inferior solution can update its solution by the minimum-cost solution to accelerate the annealing process or to escape a local minimum. Hence, the solution quality is expected to be higher for the interacting MMC PSA than for the non-interacting MMC PSA. Compared to the non-interacting version, communication overhead is significantly higher because of periodic exchanges of solutions. However, the information required for load balancing can be piggybacked on the solutions so that communication is minimized. Also, in order to reduce the communication overhead further, a segment consists of multiple outer loop iterations, i.e., processes do not exchange their solutions in every outer loop iteration.

4.2.1. Implementation In addition to reading in a graph file, performing the initial partitioning on it and broadcasting it to the rest of the processes, Process 0 is also responsible for load balancing. On receiving the initially partitioned graph from Process 0, all processes, including Process 0, execute the PSA routine. In order to achieve a significant speedup, ILI_{max} and ILR_{max} are controlled according to Equations 1 and 2.

4.2.2. Static Load Balancing The static load balancing routine is executed once at the end of the first segment. The average execution time per inner loop iteration in the first segment, t_{1j}, is used for load balancing. Let l_{ijh} and m_{ijh}denote the outer loop execution time and the total number of inner loop iterations, respectively, in an outer loop h of segment i for Process j where $h = 1, \cdots, k_{ij}$, and k_{ij} the total number of outer loop iterations in segment i for Process j. Then,

$$t_{1j} = \frac{\sum_{h=1}^{k_{1j}} \frac{l_{1jh}}{m_{1jh}}}{k_{1j}} \tag{5}$$

All processes send the cost ($cost_j$) and t_{1j} to Process 0 in a single packet. Process 0 responds with a packet contain-

ing the process rank (BR or the Best Rank) with $cost_{min}$ = $min_j\{cost_j\}$ and t_{avg}, where $t_{avg} = \frac{1}{n}\sum_{j=1}^{n} t_{1j}$. Subsequently, they receive the best intermediate solution from Process BR. Then, each process computes the estimated computing power $S_j = \frac{1}{t_{1j}}$ and $S_J = \frac{1}{t_{avg}}$, and modifies ILI_{max} according to Equation 4.

4.2.3. Dynamic Load Balancing For the interacting MMC PSA with DLB, load balancing is carried out at the end of each segment. The average execution time per inner loop iteration (t_{ij} for segment i of Process j) from the previous segment is used in load balancing for the current segment on that process. Then, t_{ij} can be computed as follows:

$$t_{ij} = \frac{\sum_{h=1}^{k_{ij}} \frac{l_{ijh}}{m_{ijh}}}{k_{ij}} \quad (6)$$

At the end of a segment, all processes send the cost ($cost_j$) and t_{ij} to Process 0 in one packet. Process 0 responds with a packet containing $t_{avg} = \frac{1}{n}\sum_{j=1}^{n} t_{ij}$ and the rank (BR) of process with $cost_{min}$ = $min_j\{cost_j\}$. On receiving this packet, Process BR broadcasts its solution to the rest of the processes. Then, each process (j) estimates its computing power, $S_j = \frac{1}{t_{ij}}$ and $S_J = \frac{1}{t_{avg}}$. The modification of ILI_{max} is carried out the same way as in the SLB version, but at the end of every segment.

4.3. Asynchronous MMC PSA

A parent process spawns child processes and locally stores the global solution (the best solution) through asynchronous interactions with the child processes as illustrated in Figure 3. Whenever each child process completes a segment, it accesses the global solution at the parent process and compares it with its local solution. If the global solution is better than the local solution, the child process copies the global solution over its local solution. Otherwise, the child process informs the parent process of the local solution. On receiving this notification, the parent process updates its own solution with the local solution. Since access to the global solution is asynchronous, the communication overhead is reduced, compared to the (synchronous) interacting MMC PSA, and the idle time is eliminated. Furthermore, the information required for load balancing can be piggybacked on these exchanges of solutions between the parent and child processes to minimize communication overhead. Once a child process is spawned, it sets ILI_{max} and ILR_{max} according to Equations 1 and 2 and starts to execute the PSA routine.

4.3.1. Static Load Balancing The SLB routine is executed once right after the first segment. The average execution time per inner loop iteration (t_{1j} for Process j) for the first segment is computed by each child process (j) according to Equation 5. An active child process sends its local cost, $cost_j$, and t_{1j} to the parent process. Then, the parent process computes $t_{avg} = \frac{t'_{avg}+t_{1j}}{2}$ where t'_{avg} is the average execution time per inner loop iteration before the parent and active child process interaction, and sends t_{avg} and the global solution cost to the child process. If $cost_j$ is lower than the global solution cost, the child process sends its local solution to the parent process. Otherwise, it receives the global solution from the parent process. Once the exchange of solution between the parent process and the active child process takes place, the SLB routine is executed on that child process. Each child process (j) estimates its computing power, $S_j = \frac{1}{t_{1j}}$ and $S_J = \frac{1}{t_{avg}}$ and modifies ILI_{max} according to Equation 4.

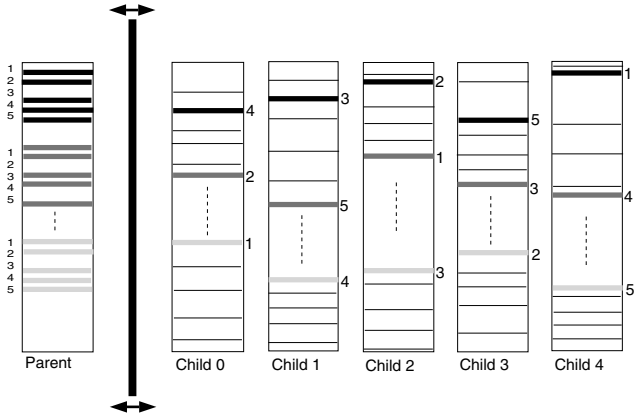


Figure 3. Asynchronous MMC PSA.

4.3.2. Dynamic Load Balancing The asynchronous MMC PSA with DLB is similar with the SLB version. The main differences are (i) the load balancing procedure (described in Section 4.3.1) is performed at the end of each segment (as opposed to only once at the end of the first segment in the SLB version); (ii) the average execution time per inner loop iteration is computed by each child process using Equation 6 (as opposed to Equation 5 in the SLB version).

5. Results and Discussion

5.1. Sun MPI Cluster and Test Graphs

The MMC PSA's described in Section 4 have been implemented on a Sun MPI Cluster [10], shared by multiple users, for performance analysis. The cluster consists of 18 UltraSparc-IIi workstations, each with 128 MB RAM. Only one PSA process is spawned on each workstation.

The three graphs employed for performance analysis are g150, g600 and g1200 with 150, 600 and 1200 nodes, respectively. These test graphs were randomly generated. The specifications of the test graphs are provided in Table 2.

Graph	Number of Edges	Number of Nodes
g150	324	150
g600	1692	600
g1200	3204	1200

Table 2. Graph specifications

In each case, cost and execution time are measured by taking their averages from 3 to 5 runs.

5.2. Cost, Speedup and Efficiency

In Figure 4, the MMC PSA's (refer to Table 1) are compared for different graphs in terms of solution quality (cost), speedup, and efficiency. The efficiency is defined as the ratio of the speedup achieved to the number of workstations used. The "background loads" on workstations in the cluster were not "controlled" for these results.

It can be seen that the interacting MMC PSA schemes perform better than the non-interacting MMC PSA schemes in most of the cases considered. That is, the interacting schemes find an equivalent quality of solution, taking less time than the non-interacting schemes. The processes in the interacting schemes guide each other to converge to an acceptable solution faster, such that the communication overhead for interaction is easily offset by an even larger reduction in the annealing time.

Another clear trend is that the asynchronous MMC PSA schemes achieve a higher speedup for the similar quality of solution than the synchronous counterparts (the interacting MMC PSA's). In the asynchronous schemes, the child processes are not synchronized for the global state access and, therefore, no synchronization overhead, particularly the idle time, is incurred. On the other hand, a significant synchronization overhead is paid at the end of each segment in the case of the synchronous schemes. It is also observed that a super-linear speedup was achieved in some cases of the asynchronous schemes (refer to Figure4-(b)). This is mainly due to the fact that the total length of the Markov chains followed by the child processes can be shorter than that of the sequential Markov chain since the length of a Markov chain is a random process.

5.3. Dependency on Number of Workstations

For the analysis of dependency on the number of workstations, g1200 with 8 subgraphs was used for the NS, IA and AA schemes. In Figure 5, cost, speedup and % efficiency are plotted as a function of the number of workstations.

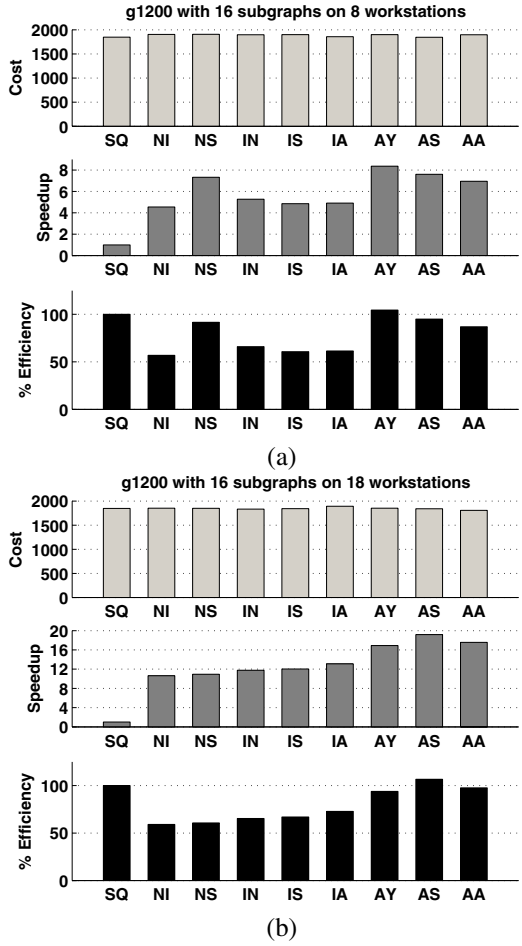
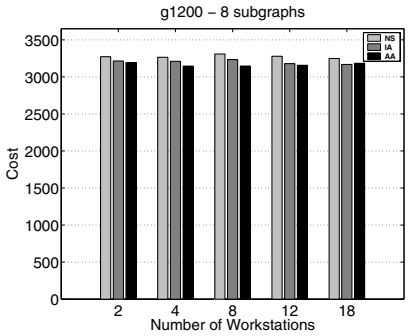
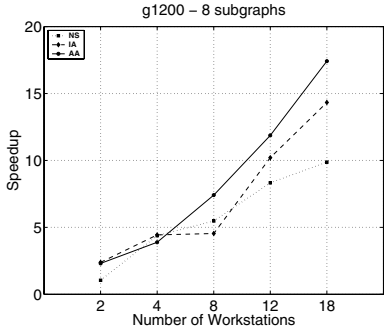


Figure 4. Cost, speedup and % efficiency for g1200 with 16 subgraphs under unperturbed load conditions for (a) 8 workstations and (b) 18 workstations.

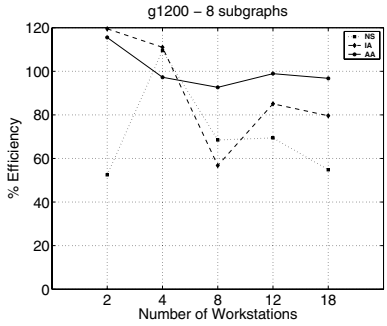
As shown in Figure 5-(a), the solution quality is almost independent of the number of workstations. The length of the Markov chain followed by each process becomes shorter as the number of workstations (processes) increases. However, the number of Markov chains increases such that the total amount of annealing remains equivalent. This makes the quality (cost) of solutions obtained by the MMC PSA schemes similar to that of the sequential solutions. In Figures 5-(b), it is seen that, as expected, as the number of workstations increases, speedup increases. In particular, the speedup achieved by the scheme AA is almost equal to the number of workstations, i.e., the efficiency is close to 100%. Such a high efficiency is mainly due to the low overhead required by the asynchronous scheme AA.



(a)



(b)

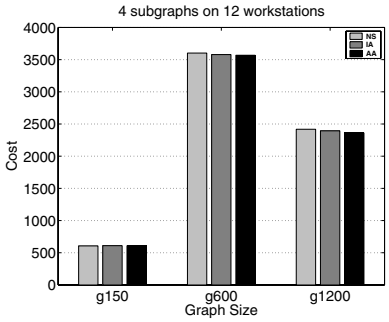


(c)

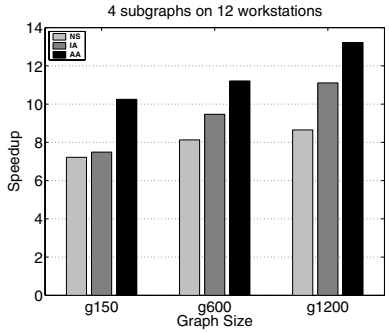
Figure 5. (a) Cost, (b) speedup and (c) % efficiency for g1200 with 8 subgraphs under unperturbed load conditions for 2, 4, 8, 12 and 18 workstations.

5.4. Dependency on Problem Size

In Figure 6, effects of the graph size on performance of the three schemes, NS, IA and AA, are analyzed for partitioning a graph into 4 subgraphs on 12 workstations. For a larger graph, the computational requirement is higher in general, which makes the relative communication (and other) overhead lower, leading to a higher speedup and efficiency. This can be observed in Figures 6-(b).



(a)



(b)

Figure 6. (a) Cost and (b) speedup on 12 workstations under unperturbed load conditions for g150, g600 and g1200 with 4 subgraphs.

5.5. Time Components

In Figure 7, the distribution of computation time and overhead (the communication and idle times) among the workstations is shown for the three schemes, NS, IA and AA. The test graph is g600, the number of subgraphs is 4 and the number of workstations employed is 12.

As can be seen in Figures 7-(a), the individual execution times of NS on 12 workstations are quite different since the processes are completely independent of each other without any synchronization. Note that most of the overhead is the idle time at the end of the annealing. In IA, the processes interact with each other in each segment through synchronous communication. Therefore, their execution times are similar as shown in Figures 7-(b). However, it is evident that the overhead is much higher than that for the other schemes. In AA, the child processes indirectly interact with each other by asynchronously accessing the global state. In other words, they are more loosely coupled than in IA. Therefore, variation of execution time among the child processes is larger compared to IA as shown in in Figures 7-(c). But, the overhead is lowest among the three schemes, which leads to the highest speedup.

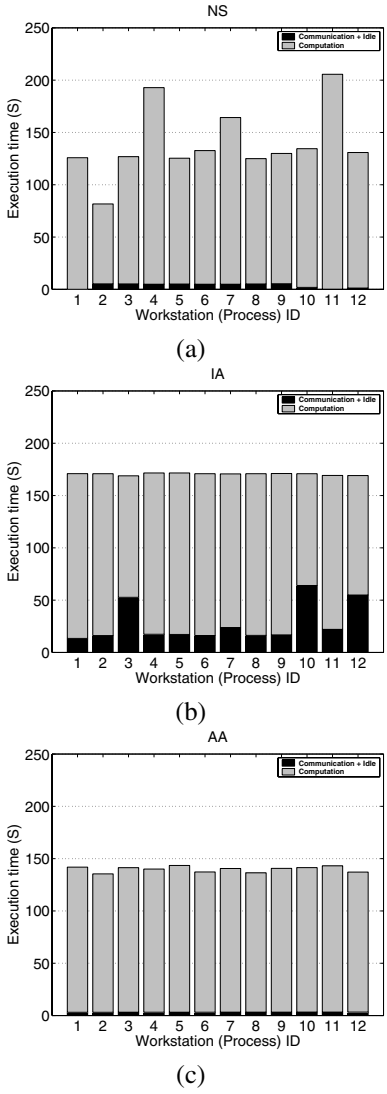


Figure 7. Execution time of 12 workstations for g600 (4 subgraphs) under unperturbed load conditions for (a) NS, (b) IA and (c) AA.

5.6. SLB and DLB Under Static and Dynamic Load Conditions

In order to test the efficiency of SLB and DLB schemes, two load environments were simulated over 12 workstations, i.e., static and dynamic load conditions.

Static load condition: Some of the available workstations running MMC PSA are pre-loaded with background jobs (viz. Matlab, Maple, etc.) that share the memory and CPU cycles on those workstations. It is ensured that these load conditions remain unchanged during the execution of PSA. This load condition is introduced to simulate *spatial heterogeneity* on the cluster.

Dynamic load condition: Some of the available workstations running MMC PSA are intermittently loaded with background jobs over time. This is carried out by repeatedly starting these jobs and terminating them after some time. The purpose of this load perturbation is to simulate *temporal heterogeneity* on the cluster.

Figures 8-(a) and (b) compare performances of the MMC PSA's with no load balancing (NI, IN and AY) and with SLB (NS, IS and AS) under the static and dynamic load conditions. Figures 8-(c) and (d) compare the MMC PSA's with SLB (IS and AS) and MMC PSA's with DLB (IA and AA) under both load conditions.

From Figure 8-(a), it is clear that, under the static load condition, the SLB versions outperform the versions with no load balancing, i.e., a higher speedup. Since the spatially varying load conditions remain unchanged throughout the entire PSA execution, adjusting the workload once in the beginning according to the estimated computing power, as in NS, IS and AS, is sufficient. On the other hand, in NI, IN and AY, the spatial heterogeneity of load is not taken into account in determining the workload of each workstation and, therefore, a significant load imbalance among the workstations is incurred, leading to a low speedup.

From Figure 8-(b), it is evident that, under the dynamic load condition, performance of NS, IS and AS is deteriorated slightly in terms of speedup. This is mainly because the load distribution does not remain the same as the initial load condition On the other hand, the speedup achieved by the no-load balancing schemes, except for NI, is higher in the dynamic load condition than in the static load condition. When the load on each workstation varies with time, the overall load distribution among the workstations tends to be averaged out, which benefits the no load balancing schemes (compared to the case when the load remains unchanged on each workstation).

From Figure 8-(c), it is clear that under the static load condition, the SLB schemes perform better than the DLB schemes in general. The speedup achieved by the SLB schemes is higher since the spatially varying load conditions remain unchanged throughout the entire PSA execution and, therefore, adjusting the workload once in the beginning according to the estimated computing power is effective. Also, the DLB schemes incur the load balancing overhead during the execution of PSA leading to a lower speedup.

As shown in Figure 8-(d), under the dynamic load condition, the speedup achieved by the DLB versions is higher than that by the SLB versions with a marginal difference in quality of their solutions. Performance of the SLB versions is deteriorated because they are not adaptive to time-varying loads on the workstations.

6. Summary

In this paper, several versions of MMC PSA for a temporally heterogeneous cluster of workstations have been developed. In order to harness heterogeneity on a cluster, static and dynamic load balancing schemes have been incorporated into the MMC PSA schemes to improve the efficiency of the PSA algorithms. The MMC PSA schemes developed in this study achieve high speedup and efficiency on a loosely coupled and temporally heterogeneous cluster of workstations, harness temporal heterogeneity by incorporating the simple yet effective load balancing algorithms, obtain the solution quality close to and in some cases better than that by the sequential SA, and scale well, particularly in the asynchronous MMC PSA case, with the increasing number of workstations.

References

[1] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi, "Optimization by Simulated Annealing," *Science,* pp. 671-680, vol. 220, 1983.

[2] R. Azencott, "Simulated Annealing: Parallelization Techniques," John Wiley & Sons, USA, pp. 200, 1992.

[3] J.A. Chandy and P. Banerjee, "Parallel Simulated Annealing Strategies for VLSI Cell Placement," *IEEE Proc. 9 th Intl Conf. VLSI Design,* pp 37, Jan 03-06, 1996.

[4] H. Chen, N.S. Flann, and D.W. Watson, "Parallel genetic simulated annealing: a massively parallel SIMD algorithm," *IEEE Trans. Parallel and Distributed Systems,* pp. 126-136, Vol.9, No. 2, 1998.

[5] S.Y. Lee and K.G. Lee, "Synchronous and Asynchronous Parallel Simulated Annealing with Multiple Markov Chains," *IEEE Trans. Parallel and Distributed Systems,* pp. 993-1008, vol.7, 1996.

[6] Georg Kliewer and Stefan Tschoke, "A General Parallel Simulated Annealing Library and its Application in Airline Industry," *Proc. Intl Conf Parallel Processing, IPDPS'00,* pp. 55 - 62, May 1 - 5, 2000.

[7] John A. Chandy, Sungho Kim, Balkrishna Ramkumar, Steven Parkes and Prithviraj Banerjee, "An Evaluation of Parallel Simulated Annealing Strategies with Application to Standard Cell Placement, " *IEEE Trans. CAD OF Integrated Circuits and Systems,* pp. 398 - 410 , VOL. 16, NO. 4, April 1997.

[8] J. Knopman and Z.S. Aude, "Parallel Simulated Annealing: An Adaptive Approach," *11th IEEE Intl Parallel Processing Symposium (IPPS '97),* pp. 552, April 01 - 05, 1997.

[9] A. Sohn, "Parallel N-ary Speculative Computation of Simulated Snnealing," *IEEE Trans. Parallel and Distributed systems,* pp. 997-1005, Vol. 6, No. 10, 1995.

[10] Sun Microsystems, "SUN MPI User's Guide."

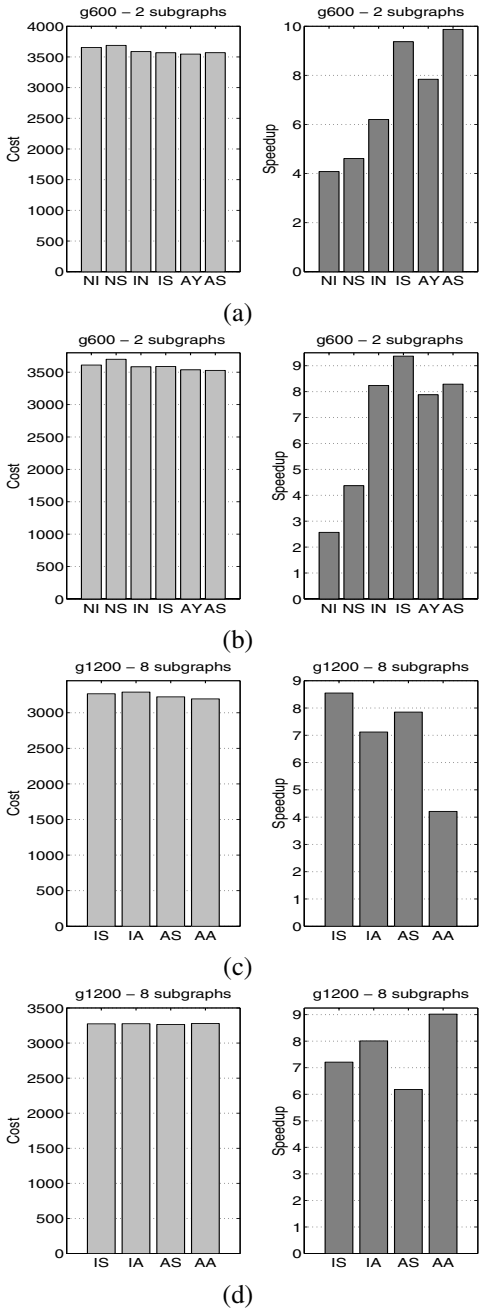


Figure 8. Comparison between no load balancing and SLB for g600 with 2 subgrahs under (a) static and (b) dynamic load conditions, and comparison between SLB and DLB for g1200 with 8 subgraphs under the (c) static and (d) dynamic load conditions. 12 workstations are employed.