The Journal of Engineering Research Vol. 2, No. 1 (2005) 77-89

A Multi-Step Approach for Scheduling Tasks with Synchronization on Clusters of Computers

B.R. Arafeh*

Department of Computer Science, College of Science, Sultan Qaboos University, P.O. Box 36. Al Khod 123, Muscat, Sultanate of Oman

Received 3 March 2004; accepted 10 October 2004

طريقة متعددة الخطوات لجدولة مهمات المعالجة المتزامنة على التجمعات الحاسوبية

المستخلص: هذا البحث يتبنى أسلوب مكون من خطوتين لجدولةمهمات المعالجة ذات الاتصال المتزامن. لخدمة هذا الهدف نعرض منهاج خوارزمي فعال، يطلق علية اسم GLB-Synch ، لوضع تخطيط توزيع مجموعات من مهمات المعالجة على المعالجات الحاسوبية وترتيبها. تستخدم الخوارزمية المعلومات التي تم الحصول عليها خلال خطوة تجميع المهمات لاختيار مجموعة مهمات وربطها بالمعالج الاقل حملا. لقد اجريت دراسة على اداء الخوارزمية GLB-Synch باستخدام أسلوب المحاكاة. ولقد تم تطوير نظام جدولة متعدد الخطوات بناءا على خوارزمية مطورة سابقا لعملية تجميع المخططات الموجهة واللادائرية (DAG) ، والتي تاخذ في الاعتبار الاتصال المتزامن بين مهمات المعالجة. حيث استخدمت الدراسة اسلوب تركيبي لتكوين مخططات مهمات المعالجة. أظهرت الدراسة بواسطة التحليل والتجريب بأن الخوارزمية GLB-Synch تحتفظ بمستوى متدني من كفاءة التعقيد مماثل لذلك الذي تم الحصول علية في الخطوة الاولى الخاصة بعملية التجميع. كما تبين نتائج دراسة الاداء الجوانب السلبية لوجود عمليات اتصال متزامنة بين مهمات المعالجة على امكانية رفع قدرة التوسع في مجال تسارع مهمات المعالجة.

المفردات المفتاحية: جدولة متعددة الخطوات، عملية التجميع، الترسيم والترتيب، تناقل الرسائل بطريقة متزامنة، النظم ذات الذاكرة الموزعة.

Abstract: In this work, a two-step approach is adopted for scheduling tasks with synchronous inter-task communication. To that end, an efficient algorithm, called GLB-Synch, is introduced for mapping clusters and ordering tasks on processors. The algorithm used the information obtained during the clustering step for selecting a cluster to be mapped on the least loaded processor. A performance study has been conducted on the GLB-Synch algorithm by simulation. A multi-step scheduling setup has been performed based on a previously developed algorithm for clustering DAGs with synchronous communication, called NLC-SynchCom, and using synthesized DAGs. We have shown by analysis and experimentation that the GLB-Synch algorithm retains the same low complexity cost of the first step for clustering. The performance results highlight the drawback of synchronization on speedup scalability.

Keywords: Multi-step scheduling, Clustering, Mapping and ordering, Synchronous message passing, Duistributed-memory systems

1. Introduction

With the era of low cost commodity hardware, clusters of workstations (COW) are emerging as platforms for parallel and distributed computing environments. Computers connected through LAN or WAN form an infrastructure Grid Computing Foster and Kesselman, (1997). They form a high computing power for massive parallel processing that can be accessed by wide spectrum of application programmers. But the communication cost over the available networking facilities is still very high compared to computing cost. Besides, it lacks the necessary reliability found in typical multi-processor interconnection networks. To overcome this deficiency, synchronous message-passing communication may need to be enforced for many parallel applications or middleware software. In general, synchronous communication adds overhead to the already high cost of communication. Furthermore, it may develop deadlock problems among the communicating tasks. Regardless of any constraints, parallel programs must be efficiently partitioned and scheduled on a COW to achieve any perceivable gain in performance.

Scheduling tasks, efficiently, on distributed memory architectures is still a challenging problem. A multi-step scheduling approach has been proposed by many researchers, such as (Sarkar, 1989; Liou and Palis, 1997), in order to reduce the complexity of the scheduling problem by using low cost heuristics. In this work, we adopt a

*Corresponding author E-mail: arafeh@squ.edu.om

The Journal of Engineering Research Vol. 2, No. 1 (2005) 77-89

two-step approach for scheduling synchronous parallel programs on distributed-memory architectures. In the first step, tasks of the program are clustered to reduce the communication cost and to avoid deadlocks, assuming unbounded number of processors. In the second step, clusters are mapped and their tasks are ordered for execution on the available number of processors.

The main contribution of this work is an algorithm, called Guided Load Balancing with Synchronization (GLB-Synch). It is a basis for mapping and ordering tasks on processors with synchronous communication. The GLB-Synch algorithm is based on the GLB algorithm for cluster mapping that was introduced by Radulescu (2001). However, the GLB-Synch algorithm performs both cluster-mapping and task-ordering, and retains the low complexity of the unbounded number of processors (UNC) schedule applied in the first step.

The rest of the paper is organized as follows. The next section introduces a background and some definitions related to this paper and the related work. Section 3 presents the GLB-Synch algorithm, while section 4 presents the performance study. Finally, section 5 is the conclusion.

2. Background and Related Work

2.1 Preliminaries

In this work, a parallel program is modeled as a weighted directed acyclic graph (DAG), $G = (V, E, \omega, \lambda)$, where V is the set of task nodes, E is the set of communication edges, ω is the set of task computation weights, and λ is the set of edge communi cation costs. An edge $e_{ij} = (n_i, n_j) \in E$ represents a data dependence constraint between the two tasks n_i and n_j, where the execution of n_j must start after receiving all input from n_i. The communication cost of message passing along an edge e_{ij} is denoted by $c_{ij} = \lambda(e_{ij})$, and the computation weight of a task n_i is denoted by $\omega(n_i)$. We will refer to the source and destination nodes of an edge by the parent node and the child node, respectively. A node that does not have any parent is called an entry node, while a node, which does not have any child, is called an exit node. $Pred(n_i)$ is the set of immediate predecessors of n_i, and $Succ(n_i)$ is the set of immediate successors of n_i. The length of a path is defined as the sum of all computation weights of nodes and all communication costs of edges along the path. The critical path of a DAG is the path from an entry node to an exit node that has the maximum length. The computation to communication ratio of a parallel program (*PCCR*) is defined as its average computation weight divided by its average communication cost.

The execution behavior of the program DAG is the macro-dataflow model. However, the execution of each task consists of three phases: receive, compute and send. The receive phase includes receiving all messages required by the task for its execution to start. The compute phase is the phase in which the instructions of the task are executed without interruption. We assume a synchronous communication protocol. The send phase includes sending all messages to all dependent tasks in parallel. However, the sender task is blocked, waiting for acknowledgements, until all receiving tasks actually receive the messages.

2.2 Multi-Step Scheduling

The multi-step scheduling process can be achieved by two or three steps. First, clustering of tasks without duplication can be performed, assuming unbounded number of processors. Second, clusters are mapped on the available processors. Third, the tasks of the mapped clusters are ordered for execution on the processors. The clustering problem has been shown to be NP-complete (Papadimitriou and Yannakakis, 1999; Sarkar, 1989). Polynomial-time heuristic algorithms have been proposed for the clustering problem based on the critical path analysis (Sarkar, 1989; Wu and Gajski, 1990; Gerlasoulis and Yang, 1993; Kwok and Ahmad, 1999; Kadamuddi and Tsai, 2000; Shirazi *et al*. 1990; Lee *et al*. 2003). An overview of cluster-mapping is given next.

2.2.1 Mapping Clusters to Processors

Cluster-mapping is needed when the number of available processors is less than the number of clusters. There are a number of approaches that are proposed in literature. However, the issue of mapping clusters to processors has not been given enough attention in the literature, and there is much room to explore on this topic. In the next paragraphs, we discuss some of the approaches reported in the literature.

Sarkar (1989) used a list-scheduling based method to map the clusters to processors, called List Cluster Assignment (LCA). It is an incremental algorithm that performs both cluster-mapping and task-ordering in a single step. Kim and Browne (1988) proposed a mapping scheme for clusters based on their linear clustering algorithm. The clusters are first merged in order to reduce their number to be at most equal to the number of processors. Then the process is followed by heuristics to optimize the mapping step. Mainly, the heuristics would choose a processor which has the most appropriate number of channels among currently unallocated processors. Wu and Gajski (1990) proposed a mapping scheme for clusters based on a dedicated traffic scheduling algorithm that balances the network traffic. The algorithm used generates an initial assignment by a constructive method; then, the assignment is iteratively improved to obtain a better mapping. The heuristic is based on minimizing the total communication traffic. Yang and Gerasoulis (1994) employed a work profiling method for merging clusters, called Wrap Cluster Merging (WCM) algorithm. First, clusters are sorted in an increasing order of aggregate computational load. Then, a load balancing algorithm is invoked to map the clusters to the processors, so that every processor has about the same load.

The work of Liou and Palis (1997) investigated the

The Journal of Engineering Research Vol. 2, No. 1 (2005) 77-89

problem of mapping clusters to processors. They have shown the effectiveness of using the two-phase scheduling approach, in which the task clustering is followed by the cluster-mapping step, over the one-phase scheduling. They proposed a clustering algorithm, called CASS-II (Clustering and Scheduling System II), and introduced three algorithms for cluster-mapping schemes, namely, the LB (Load-Balancing) algorithm, CTM (Communication Traffic Minimizing) algorithm, and the RAND (Random) algorithm. They applied randomly generated task graphs in an experimental study using their clustering algorithm and cluster-mapping schemes. Their work shows that, when task clustering is performed before cluster-mapping, load balancing is the preferred approach for merging clusters. Compared to CTM, LB is fast, easy to implement and produces significantly better schedules.

Radulescu (2001) proposed two algorithms for mapping clusters to processors in a multi-step scheduling approach. Both algorithms aim at achieving a better cost-performance ratio. The first algorithm, called Guided Load Balancing (GLB), exploits knowledge about the task start times that were computed in the clustering step. Accordingly, clusters are mapped in the order of their start times to the least loaded processor at that time. The second algorithm, called List Load Balancing (LLB), aims at improving the load balancing throughout the program execution time by performing cluster-mapping and task-ordering in one step. There are two benefits reported for this integration. First, it allows dynamic load balancing through the execution of the mapping process, because only the ready tasks are considered in the mapping process. Second, it considers communication costs, when selecting tasks for mapping, as opposed to other cluster-mapping algorithms, such as WCM and GLB, which do not.

The work by Lee *et al*. (2003), introduced a multi-step scheduling approach using a block dependency DAG, that represents the execution behavior of block sparse Cholesky factorization operation in a distributed-memory system. The proposed scheduling algorithm consists of two stages. In the first stage, a clustering algorithm, called Early-Start Clustering (ESC), is used to cluster tasks while preserving the earliest start time of a task without limiting the potential degree of parallelism, and without considering the number of available processors. In the second stage, a cluster mapping algorithm, called Affine Cluster Mapping (ACM), is used to allocate clusters to a given number of processors. The ACM algorithm attempts to reduce the communication overhead and balance the workloads among the processors based on two criteria. These are the affinity of a cluster with respect to a processor, in terms of the sum of communication costs required when the cluster is mapped to other processors, and the amount of workload required for a cluster. The work by Lee *et al*. (2003) shows by experiments the effectiveness of applying the proposed scheduling algorithm, compared to other processor mapping methods that are used for parallelizing the sparse Cholesky factorization operation. The experiments were conducted on a Myrinet cluster system and using benchmark sparse matrices

2.2.2 *Scheduling with Synchronous Communication*

Most scheduling algorithms for distributed-memory parallel architectures assume the use of an asynchronous communication protocol for a message-passing system. However, parallel computing on a network of workstations or over the Internet is not as reliable as that performed on parallel machines. Therefore, the requirement for synchronization at the application level becomes eminent for many software systems.

In a synchronous communication, the sender is blocked until an acknowledgement is received from the receiver. This waiting time is called the blocking delay. A deadlock occurs when a sender gets blocked indefinitely, waiting for an acknowledgment from a receiver task in another cluster. At the same time, the receiver task cannot start execution, because one of its predecessor tasks has been blocked indefinitely, waiting for an acknowledgment from a task in some other cluster. A direct deadlock situation between two clusters occurs due to a cyclic dependency relation between them. In general, a deadlock situation may arise due to a chain of dependency relations among a subset of tasks. In this work, we consider direct deadlock situations only. Based on the task execution phases, the following definitions of time parameters characterize the scheduling of a task node in a scheduled DAG with synchronous communication.

- $s\_receive(n_i)$: Start time for receiving messages by task node n_i
- $e\_receive(n_i)$: End time for receiving messages by task node n_i
- $s\_compute(n_i)$: Start time for computation by task node n_i
- $e\_compute(n_i)$: End time for computation by task node n_i
- $s\_send(n_i)$: Start time for sending messages by task node n_i.
- $e\_send(n_i)$: End time for sending messages by task node n_i.

The issue of scheduling on distributed-memory parallel architectures with synchronous communication has not been given enough attention in literature. However, the works of Kadamuddi and Tsia (2000) and Arafeh (2003) address this issue, assuming a multi-step scheduling approach. Both propose clustering algorithms for tasks with synchronous communication, in which deadlocks are detected and avoided as part of the task clustering step.

The work presented in this paper uses the clustering algorithm, called NLC-SynchCom, by Arafeh (2003) for the task-clustering step. The algorithm proceeds in one pass in the forward direction from entry nodes to exit nodes, one level at a time. The algorithm starts assuming

The Journal of Engineering Research Vol. 2, No. 1 (2005) 77-89

each task node is in a cluster by itself. Therefore, there are $|V|$ clusters at the beginning of the algorithm. Each node in a cluster is designated by its status as a Head, Tail, Regular or Singleton. A node in a cluster by itself is given the status of a Singleton node. The Head task of a cluster is the one that must be scheduled first due to its precedence with respect to all other tasks in the cluster. Similarly, the tail task of a cluster is the one that must be scheduled last due to its precedence with respect to all other tasks in the cluster. A Regular task in a cluster is one that is not a Singleton, Head or Tail. The selection of a parent node, n_i, at level l that will be merged with one of its child nodes is determined using two priority schemes. The first scheme is used to determine the priority of a parent node at level l for merging. It is defined by the parent's completion time, *e_send(parent)*, in descending order. The second scheme is used to determine the priority of merging a child node n_j for a parent n_i. This priority depends on the maximum remaining time left to the completion of execution from a parent node, n_i, to an exit node, excluding n_i. Since a merging step would zero a (*parent, child*) edge, then the parent node and all its descendant nodes may have their completion times changed accordingly. Thus, the priorities of the parent nodes at each level are found dynamically before the nodes of that level are scanned for merging. On the other hand, the remaining time to the completion of execution for each node is computed at the initialization time only.

A priority is given to merge the parent node with a child node which leads to the highest remaining time to completion. A selected child node n_j has to pass two tests before merging can be finally applied. These are the deadlock detection and the merging check tests. First, the deadlock detection test ensures that a merging step of a child with its parent's cluster would not cause a deadlock case for the parent's cluster with any other existing cluster in the DAG. Second, the merging check test ensures that a merging step of a child node with its parent's cluster would not cause an increase in the application's execution time. This execution time is referred to in this paper by the DAG Parallel time, *PT*. The (*parent, child*) edge is zeroed and merging is performed, only, if both tests are passed successfully. The complexity cost of the NLC-SynchCom algorithm is $O(v(\log v + e^2))$, where v is the number of nodes and e is the number of edges in a DAG. For further details, see the paper by Arafeh (2003).

3. Cluster-Mapping and Scheduling with Synchronous Communication

3.1 Description of the GLB-Synch Algorithm

The GLB-Synch algorithm is an extension of Radulescu GLB algorithm for mapping clusters to processors (Radulescu, 2001). However, the GLB-Synch algorithm performs both cluster-mapping and task-ordering in the context of synchronous communication. The algorithm uses the information obtained during the clustering step, based on the NLC-SynchCom algorithm, for mapping clusters to a distributed-memory system of unbounded number of homogeneous processors that are fully connected. Eventually, the NLC-SynchCom algorithm schedules the tasks on the virtual processors, assuming each cluster is allocated to one virtual processor and there is Unbounded Number of Processors or Clusters (UNC). Therefore, we will refer to the DAG and schedule generated by the NLC-SynchCom algorithm by the clustered DAG and the UNC schedule, respectively. Each node of the clustered DAG represents a cluster or virtual processor, and each directed edge between two clusters represents a communication link connecting them. The GLB-Synch algorithm uses the cluster start time, $T_s(C)$, to represent the priority of a cluster, C, for mapping. The start time of a cluster is the start time for a computation phase of the header task in the cluster, and it is given by

$$T_s(C) = \underset{t \in C}{Min}\{s\_compute(t)\}, \tag{1}$$

where t is a task of cluster C.

Similar to GLB, the cluster, which has the earliest start time, is mapped first. In case of a tie, the cluster with the highest workload is mapped first. If there is still a tie, a cluster is selected randomly. Based on the execution phases of a task, the UNC schedule allocates time slots for all the phases, with no consideration for overlapping computation with communication. At this stage, it is more natural to consider the existence of some overlapping between the computation and the communication phases, as clusters are mapped to physical processors. In this work, we assume that the duration of the receive phase is implicitly handled by a communication processor, and the acknowledgment of a received message is handled by the message-passing system. Each scheduled task should have all its expected messages to be received declared to the message-passing system ahead of the start time of its receive phase. Only when all the needed messages have arrived, the task can start the computation phase. However, the duration of the sending phase is still considered explicitly as part of the task's schedule on a processor to enforce synchronization. The GLB-Synch algorithm maps clusters to a distributed-memory system of bounded number of homogeneous processors that are fully connected. Since a complete interconnection network is assumed, there is no consideration for bandwidth contention. Furthermore, each processor is assumed to have unlimited number of communication ports and unlimited memory space.

Since the workload of a task, t, includes the duration starting from *s_compute(t)* till *e_send(t)*, then the workload of a cluster is defined as

$$T_w(C) = \sum_{t \in C} (e\_send(t) - s\_compute(t)) \tag{2}$$

Because a cluster is not a schedulable unit, the GLB-Synch algorithm maps a cluster to the least loaded proces-

The Journal of Engineering Research Vol. 2, No. 1 (2005) 77-89

sor as in the GLB algorithm. It is expected that clusters mapped to the same processor to be interleaved due to the task-ordering step. The workload of a processor, *p*, is defined as

$$T_w(p) = \sum_{C \in \Psi(p)} T_w(C) \quad (3)$$

where, $\psi(p)$ is the subset of clusters mapped to processor *p*. As a consequence, all inter-cluster communication costs among the mapped subset of clusters must become zero.

The algorithm achieves the objective of scheduling tasks on the processors in three steps. In the first step, it assumes that the process of cluster-mapping is generating super-clusters, constructed as aggregates of the mapped clusters to the target processors. In this step, all the time parameters of all tasks are recomputed, due to the zeroing of the inter-cluster communication costs, without performing the task-ordering (*i.e*. sequentialization) process. It may look as if each aggregate of clusters (*i.e*. virtual processors) is mapped to a shared-memory multiprocessor.

In the second step, the GLB-Synch algorithm performs task-ordering, based on start times of tasks for the computation phase. All tasks allocated to the same processor are sorted topologically in an increasing order of their start computation time, *s_compute*. If two tasks have the same *s_compute* time, the task with the highest blocking delay is scheduled first. The blocking delay of a task, *t*, is defined as the time it is waiting for acknowledgments for all messages sent by that task, and it is computed by *(e_send(t) - s_send(t))*. If there is still a tie, a task is selected randomly. In the third step, the mapped tasks to the same processor are scheduled using their precedence order. Let $T^i_r(p)$ denote the processor ready time on a partial schedule. It is initialized by zero, and it is defined as the end time of the last task, t_i, scheduled on that processor. Accordingly, a task t_{i+1}, is scheduled for execution at $T^i_r(p)$, if the current start time for the computation phase, $s\_compute(t_{i+1})$ is less than or equal to $T^i_r(p)$. Otherwise, the task t_{i+1} is scheduled to start execution at its designated *s_compute* time. The GLB-Synch algorithm is described text.

GLB-Synch Algorithm

Input:
1. A clustered DAG
2. The Table of time parameters (*i.e*. UNC schedule)
3. Number of processors.

Output:

1. A mapping of the clusters to the processors
2. Tasks schedule on the processors.

Algorithm Steps:

1. Compute the start time, $T_s(C)$, and the workload, $T_w(C)$ for each cluster, *C*.
2. Sort the clusters in an increasing order based on T_s, breaking ties by choosing the cluster with the highest workload. If there is still a tie, select one randomly.
3. For each cluster, *C*, do
   * Map *C* to a processor, *p*, with the least workload.
   * Zero the inter-cluster communication cost between any currently mapped clusters to *p* and *C*.
   * Update the workload of processor *p*:

$$T_w(p) = T_w(p) + T_w(C) \quad (4)$$

4. Update the UNC schedule due to the mapping step.
5. For each processor, *p*, do
   * Perform task-ordering for all tasks mapped to *p* based on *s_compute* time, breaking ties by choosing the task with the highest blocking delay. If there is still a tie, select one randomly.
   * For each task t_i mapped to a processor

If $(T_r^{i-1}(p) \le s\_compute(t_i))$ then
Schedule t_i to start at $s\_compute(t_i)$;

$$T_r^i(p) = s\_compute(t_i) + T_w(t_i); \quad (5)$$

Else

Schedule t_i *to start at* $T_r^{i-1}(p)$;

$$T_r^i(p) = T_r^{i-1}(p) + T_w(t_i) \quad (6)$$

3.2 Complexity Analysis

The following notations are used to characterize the time complexity of the GLB-Synch algorithm.

c: The number of clusters.
e: The number of DAG edges, $|E|$.
v: The number of DAG vertices, $|V|$.
m: The number of processors.

The GLB-Synch algorithm performs cluster-mapping, task-ordering and scheduling on the processors, based on the clustering step for synchronous communication. The GLB-Synch algorithm assumes that the formulation of clusters, along with the computation of their workloads, have been performed in the clustering step. Besides, it assumes that task clusters are deadlock-free, since all generated clusters by the clustering step had passed the deadlock detection test successfully.

Theorem 1*. The time complexity of the GLB-Synch algorithm is O(mc + mv logv + ev).*

Proof*. The start time of a cluster, $T_s(C)$, is the s_compute*

The Journal of Engineering Research Vol. 2, No. 1 (2005) 77-89

time of the Head task of C. Then, the cost of the first step is O(c). In step 2, clusters are sorted in O(c logc) time. The determination of the least loaded processor, p_j, takes O(m) steps. While, the process of zeroing the inter-cluster communication costs between the already mapped clusters to p_j and the current one to be mapped needs O(e). Accordingly, step 3 of the algorithm takes O(c(m+e)) time.

The updating of the UNC schedule at step 4 takes O(ev) steps, due to the need to find the e_receive(t_i) and the e_send(t_i) time for each task t_i. Finally, step 5 of the algorithm takes O(mv logv) steps. Because, the algorithm orders all tasks mapped to a processor in O(v logv) steps, then schedules tasks on a processor in O(v) steps. Since m < c and c < v, the total time complexity of the GLB-Synch algorithm is O(pc + mv logv + ev).

4. Performance Study

A performance study has been conducted on the multi-step approach for scheduling tasks with synchronous inter-task communication. The study is based on the simulation of the multi-step scheduling approach using the NLC-SynchCom algorithm for the clustering step, and the GLB-Synch algorithm for the cluster-mapping and task-ordering steps. The performance study had adopted randomly generated DAGs for experimentation. Synthesized random DAGs are generated so that the results would not be biased towards regular graph structures or certain graph shapes, allowing various DAG characteristics to be considered. The objectives of the performance study include assessing the cost of the multi-step scheduling approach in the context of synchronous communication, evaluating the outcome of the multi-step scheduler, and discovering the points of deficiencies and limitations. In this section, the definitions of the chosen performance metrics in the study are given next. Then, the simulation set-up for experimentation is described. Finally, the performance results are presented and discussed.

4.1 Performance Metrics

The performance metric considered for assessing the cost of each step of the multi-step scheduler is the execution time. The performance metrics considered for evaluating the outcome of the scheduling steps are based on the Schedule Length (*SL*). We will refer to SL_o as the original schedule length of an application DAG. It is the parallel time for executing a DAG on an unbounded number of processors. SL_o is equal to the length of the critical path of the DAG. The schedule length obtained from executing a DAG on a uniprocessor is referred to by SL_1, and it is defined as

$$SL_1 = \sum_{v_i \in V} \omega(v_i) \tag{7}$$

The schedule length obtained by the clustering step (*i.e.* the UNC schedule) is referred to by SL_c. It is also based on an unbounded number of processors. The schedule length obtained by mapping and scheduling the tasks on bounded number of processors, *m*, is referred to by SL_m. The speedup factor, SP_m, is defined as the ratio of executing a parallel program on a uniprocessor to its execution on *m* processors, and it is given by

$$SP_m = \frac{SL_1}{SL_m} \tag{8}$$

Definition 1 Normalized Schedule Length *NSL(m)*. *In this work, we define the Normalized Schedule Length, NSL(m), as the ratio of the schedule length of a DAG on m processors, SL_m, to its original schedule length, SL_o. That is,*

$$NSL(m) = \frac{SL_m}{SL_o} \tag{9}$$

Definition 2 Utilization *U(m)*. *The utilization of a system of m processors, U(m), is defined as the percentage of the m processors' time that is kept busy during the execution of a parallel program due to a certain allocation Hwang, (1993). Let $b(v_i, p_j)$ be the blocking time of task v_i due to synchronization, when it is executed on p_j. Hence,*

$$U(m) = \frac{\sum_{v_i \in V} \sum_{p_j \in P} x(v_i, p_j)(\omega(v_i) + b(v_i, p_j))}{m \times SL_m} \tag{10}$$

where, $x(v_i, p_j)$ is a *0-1* function defined as follows:

$$x(v_i, p_j) = 1 \text{ if } v_i \text{ is allocated to processor } p_j \\ = 0 \text{ otherwise} \tag{11}$$

and

$$b(v_i, p_j) = e\_send(v_i, p_j) - s\_send(v_i, p_j) \tag{12}$$

Definition 3 Efficiency *E(m)*. *The Computing Efficiency, E(m), or the Efficiency for short, is defined as the actual percentage of the computing time performed by the m processors during the execution of a parallel program. The Efficiency indicates the actual degree of speedup achieved on m processors compared with the maximum value Hwang, (1993). It is given by,*

$$E(m) = \frac{\sum_{v_i \in V} \omega(v_i)}{m \times SL_m} = \frac{SL_1}{m \times SL_m} = \frac{SP_m}{m} \tag{13}$$

Definition 4 Synchronization Overhead Ratio *SOR(m)*. *Let B_m be the average blocking (i.e. synchronization time) for scheduling tasks of a parallel program on bounded number of processors, m. Accordingly, the Synchronization Overhead Ratio, SOR(m), is defined as the ratio of the average blocking time per processor to the schedule length, SL_m, where,*

$$B_m = \frac{\sum_{p_j \in P} \sum_{v_i \in V} b(v_i, p_j)}{m}, \text{ and} \quad (14)$$

$$SOR(m) = \frac{B_m}{SL_m} \quad (15)$$

Definition 5 Load Imbalance Factor *LIF(p)*. *Let $T_w(p_j)$ refer to the workload of processor p_j. Therefore, the Load Imbalance Factor, LIF(p), is defined as the percentage of the average processor's time that is kept idle during the execution of a parallel program. Actually, it indicates the percentage of non-utilized processor's time, and it is given by*

$$LIF(m) = \frac{\sum_{p_j \in P} (SL_m - T_w(p_j))}{m \times SL_m} = 1 - U(m) \quad (16)$$

From definitions 2, 3 and 4, the utilization is characterized by the following Lemma.

Lemma 1. *The utilization U(m) is characterized in terms of the efficiency, E(m), and the synchronization overhead ratio, SOR(m), for scheduling a parallel program on m processors as*

$$U(m) = E(m) + SOR(m) \quad (17)$$

Proof. *Let b_{ave} be the average blocking time in a DAG, and B_m the average blocking per processor for scheduling the DAG on m processors. Accordingly, the utilization can be rewritten as*

$$\begin{aligned} U(m) &= \frac{\sum_{v_i \in V} \sum_{p_j \in P} x(v_i, p_j)(\omega(v_i) + b(v_i, p_j))}{m \times SL_m} \\ &= \frac{\sum_{v_i \in V} \sum_{p_j \in P} x(v_i, p_j)\omega(v_i)}{m \times SL_m} \\ &\quad + \frac{\sum_{v_i \in V} \sum_{p_j \in P} x(v_i, p_j) b(v_i, p_j)}{m \times SL_m} \end{aligned} \quad (18)$$

$$\begin{aligned} U(m) &= E(m) + \frac{v \times b_{ave}}{m \times SL_m} = E(m) + \frac{B_m}{SL_m} \\ &= E(m) + SOR(m) \end{aligned} \quad (19)$$

4.2 Simulation

The use of randomly generated Directed Acyclic Graphs (DAGs) to model parallel applications is a common practice in the evaluation of proposed scheduling heuristics for parallel and distributed computing systems. The use of simulation provides a basis to evaluate the scheduling algorithm independent of the hardware implementation and its organization. Many approaches have been proposed in the literature on how to generate synthesized DAGs randomly (Kasahara Laboratory, Japan, 2004). In this work, a random graph generator is implemented to generate weighted DAGs, as defined in this paper, with various characteristics based on a method that uses the following factors:

1. The number of tasks, *v*.
2. The shape factor, α., of a DAG: We assume the height (*i.e*. number of levels) of a DAG is randomly generated from a uniform distribution with a mean value, L_{mean}, equal to $v^{1/2}/\alpha$. Similarly, the width for each level in the DAG is randomly generated from a uniform distribution with a mean value, W_{mean}, equal to $\alpha v^{1/2}$.
3. The maximum out-degree of a node in a DAG.
4. The maximum span of an edge in a DAG.
5. The Computation to Communication Ratio, *CCR*: It is taken as the ratio of the average computation weight to the average communication cost. Values of *CCR* in the range 0.1-0.7 represent fine granularity, values in the range 0.8-1.4 represent medium granularity, and values greater that 1.4 represent coarse granularity.
6. The mean computation weight, ω_{mean}: The computation weight of each node is determined randomly from a uniform distribution with a mean ω_{mean}.
7. The mean communication cost, λ_{mean}: The mean communication cost of a DAG is equal to ω_{mean}/CCR. Each communication cost of an edge is determined randomly from a uniform distribution with a mean λ_{mean}.

For the purpose of generating random DAGs in this study, we have arbitrarily chosen ω_{mean} to be 20. Three values of *CCR* are considered. 0.5, 1.0 and 5.0, to represent fine, medium and coarse granularity respectively. To control the structure of the DAG, we have limited the outdegree of a node to be within the range 1-W_{mean}. Also, the span of an edge is limited to be within 1-$\lceil 0.25 \times \text{height} \rceil$. Three values for the shape factor, α, are considered. These are 0.5, 1.0, and 2.0. An $\alpha < 1.0$ represents a DAG with a long height and low degree of parallelism; while an $\alpha > 1.0$ represents a shorter DAG with high degree of parallelism. The width of a DAG level *l* indicates the degree of parallelism at that level. Therefore, the mean width of a DAG, *Wmean*, is adopted as a measure of the potential for the degree of parallelism in the DAG. The relationship between the mean width, W_{mean} of a DAG and the number of nodes, *v*, of the DAG for constant values of α are shown in Fig. 1. The plots shown in Fig. 1 indicate the potential speedup expected by scheduling a DAG on *m* processors. Accordingly, values of $m >> W_{mean}$ are not expected to have any significant speedup improvements in the *PT* of a DAG.

The Journal of Engineering Research Vol. 2, No. 1 (2005) 77-89

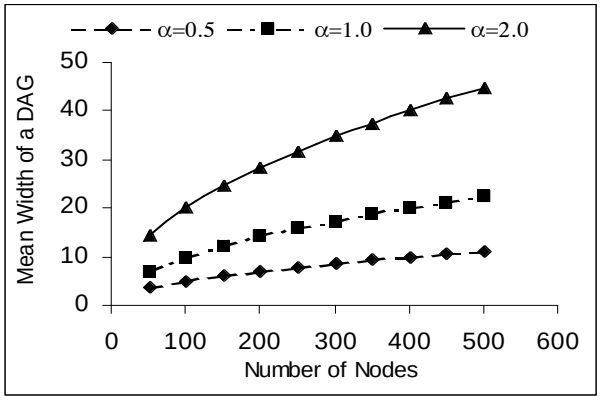


Figure 1. Relationship between the mean width and the number of nodes in DAG

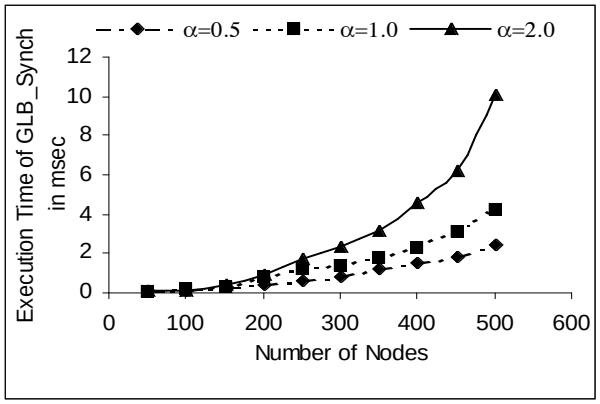


Figure 3. The average execution time of the mapping and ordering step versus the number of nodes

Simulation experiments were conducted to measure the cost of each step of the multi-step scheduling techniques used in this work. Ten groups of synthesized DAG sizes were generated. These range from 50 to 500 nodes with an increment of 50. For each DAG size, *v*, and shape factor, α, we have generated 100 random DAGs. The NLC-SynchCom algorithm is applied on each generated DAG for clustering tasks with synchronous communication. The resultant clustered DAG is taken as an input to the GLB-Synch algorithm for mapping and ordering the tasks on the processors.

4.3 Performance Results

The cost of each step in our multi-step scheduling scheme is measured against the number of nodes in a DAG. Figures 2 and 3 show the average execution time for the clustering step, and the mapping and ordering step, respectively, versus the number of nodes. The average execution time for the mapping and ordering step is taken over all the number of processors considered in the simulation runs. Three cases are considered for each scheduling step, based on the shape factor of the DAG. For both steps, the execution time increases as the DAG size and the degree of parallelism (*i.e.* α) in the DAG increase. From the plots, it can be deduced that the cost of the mapping and ordering step is much less than the cost of clustering, in general. The cost of mapping and ordering does not exceed 25% of the cost of clustering for DAG sizes greater than 100 nodes.

In order to assess the performance of the GLB-Synch algorithm, we have to focus a little on the main performance features of the NLC-SynchCom algorithm. The main objective of performing the clustering step is to reduce the communication cost in the DAG by merging tasks onto clusters, where each cluster can be assigned to a processor. In this way, the clustered DAG would have better *PCCR* relative to the initial value of the DAG's *PCCR*. The results, shown in Figs. 4 and 5, indicate the role of the

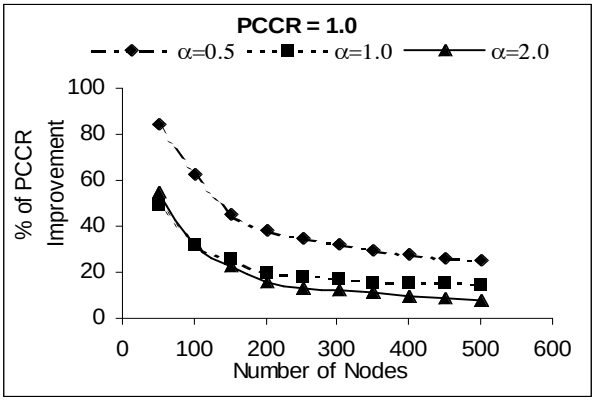


Figure 4. The percentage improvement in *PCCR* due to the clustering step versus the number of nodes

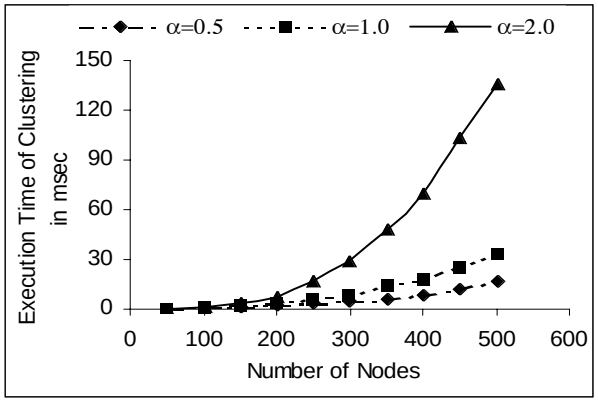


Figure 2. The average execution time of the clustering step versus the number of nodes

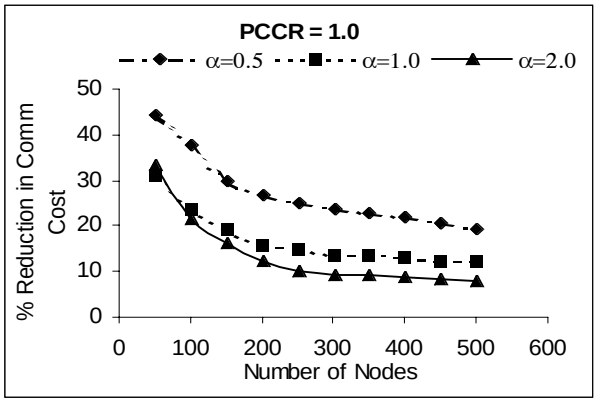
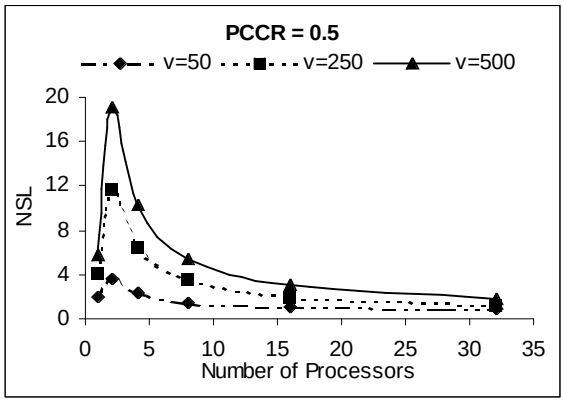
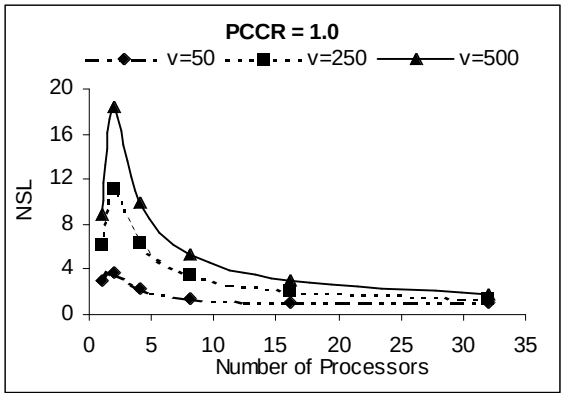


Figure 5. The percentage reduction in communication cost due to the clustering step versus the number of nodes

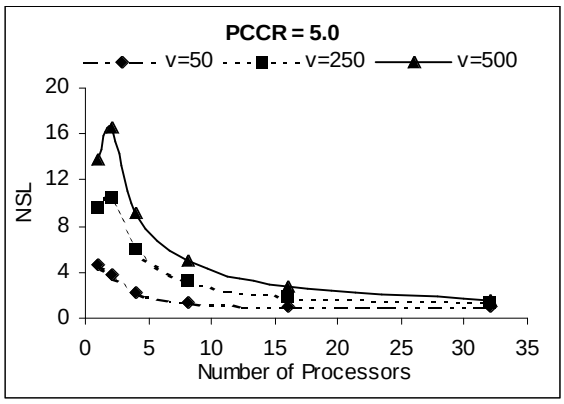
The Journal of Engineering Research Vol. 2, No. 1 (2005) 77-89



(a)

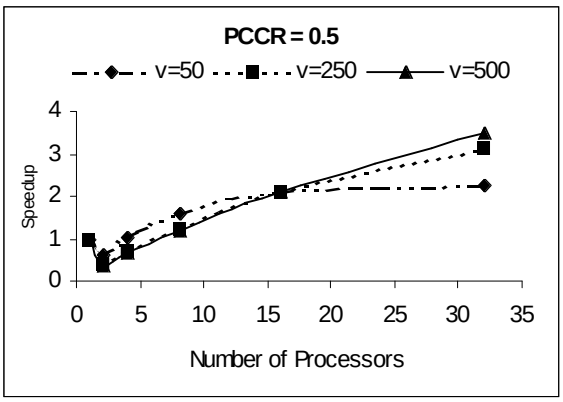


(b)

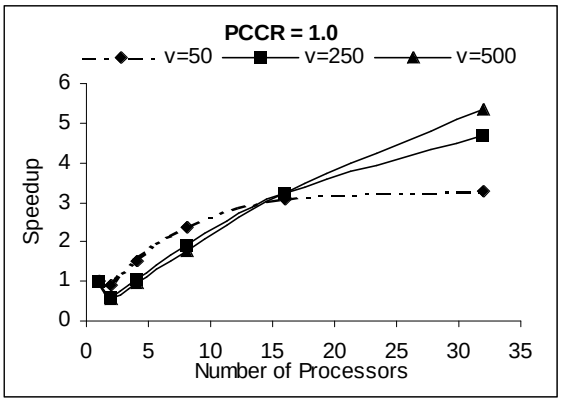


(c)

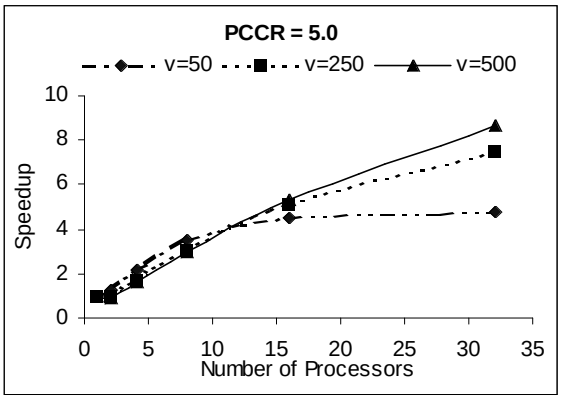
Figure 6. The Normalized Scheduling Length of DAGs versus the number of processors



(a)



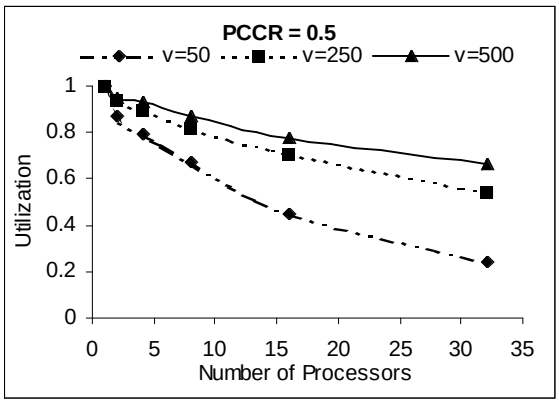
(b)



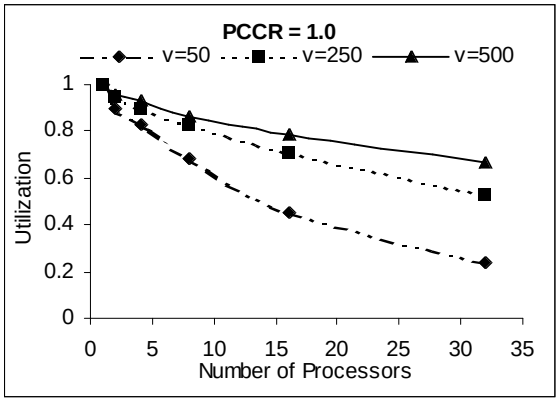
(c)

Figure 7. The average Speedup factor versus the number of processors

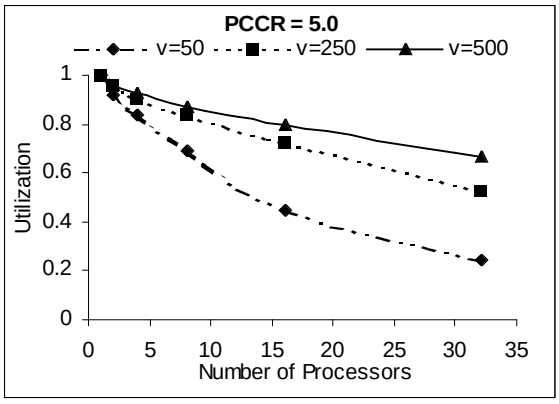
The Journal of Engineering Research Vol. 2, No. 1 (2005) 77-89



(a)

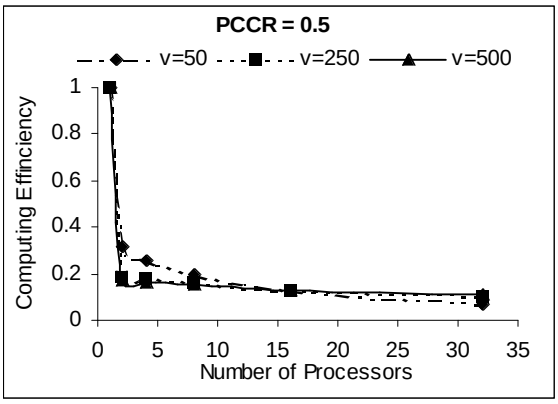


(b)

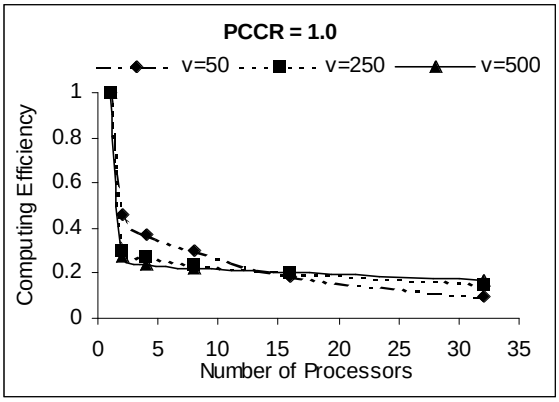


(c)

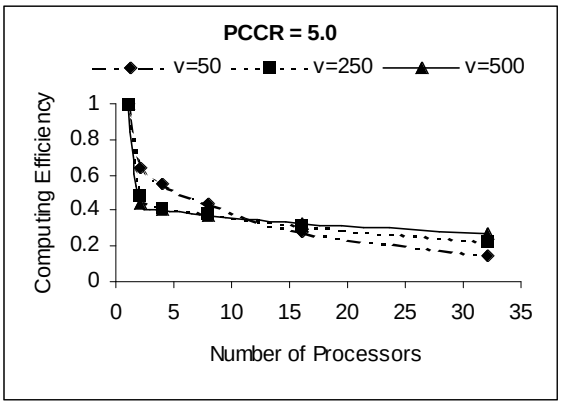
Figure 8. The Utilization versus the number of processors



(a)



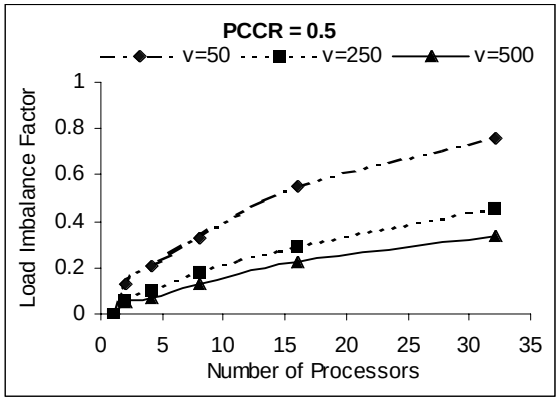
(b)



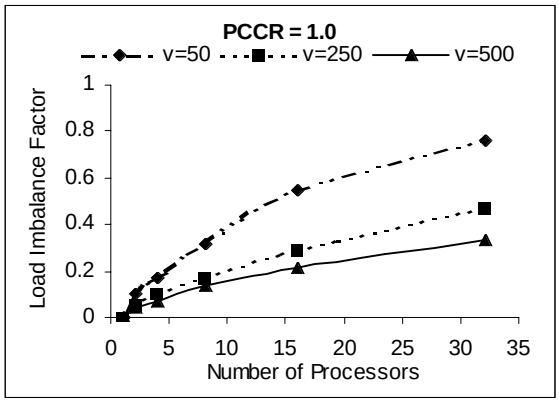
(c)

Figure 9. The Computing Efficiency versus the number of processors

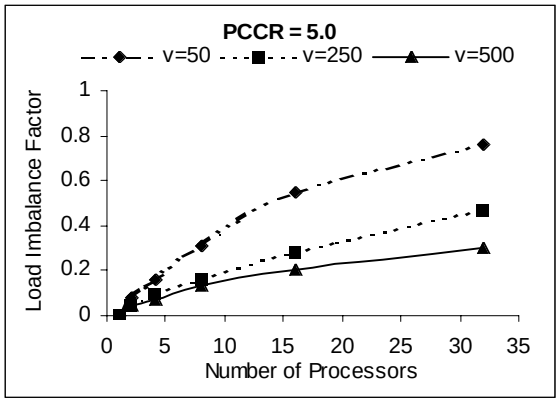
The Journal of Engineering Research Vol. 2, No. 1 (2005) 77-89



(a)



(b)



(c)

Figure 10. The Load Imbalance Factor versus the number of processors

clustering step in improving *PCCR* and reducing the communication cost, respectively, for DAGs with *PCCR*=1.0. Both measures tend to degrade at higher DAG sizes and shape factors, as shown in Figs. 4 and 5. However, the NLC-SynchCom algorithm imposes restrictions on merging nodes to clusters that may cause an increase in the DAG's *PT*. Therefore, the UNC schedule length, SL_c, is always less than or equal to SL_o. For example, the parallel time is reduced by about 9% for DAGs with a size of 50 nodes, *PCCR*=1 and α = 0.5. The ability of the algorithm for *PT* reduction degrades, generally, with higher DAG sizes. For example, the *PT* reduction is about 1%, for DAGs with a size of 500 nodes, *PCCR* = 1.0 and α = 0.5. This brief characterization of the clustering step should support understanding the outcomes of the mapping and ordering step (for more details see Arafeh, 2003).

The performance results are shown in Figs. 6-10. They are taken for DAGs with α =1.0 and sizes of 50, 250 and 500 nodes. This should provide us with bases of uniformity in our comparisons, analyses and assessments. This does not mean that the effect of the shape factor has been ignored, or that it does not have a role on the type of results generated. On the contrary, all performance results in this work scale proportionally with the value of α. Figures 6-10 show the relationships of the normalized scheduling length, *NSL*, the speedup, *SP*, the utilization, *U*, the efficiency, *E*, and the load imbalance factor, *LIF*, against the number of processors, respectively. In particular, simulation results were collected for number of processors equal to 1, 2, 4, 8, 16 and 32. However, curve fitting techniques were applied in order to obtain smooth curves in those figures.

The results shown in Fig. 6(a), (b) and (c) depict the relationship between the *NSL(m)* of DAGs with *PCCR* valus equal to 0.5, 1.0 and 5.0, respectively, versus the number of processors, *m*. The *NSL* results are high with small number of processors, but they approach optimal values at high number of processors. The *NSL* values are optimal in the cases of DAG size of 50 nodes, and close to optimal for higher sizes, when $m > 16$. The next set of results is for the average speedup factor against the number of processors. They are shown in Fig. 7(a), (b) and (c) for *PCCR* values equal to 0.5, 1.0 and 5.0, respectively. Before the crossover points, the speedup factor is higher for lower DAG sizes. While after the crossover points, the speedup factor becomes higher for higher DAG sizes.

The crossover points in the speedup curves are expected. Because, the potential degree of parallelism (represented by the average DAG width) for smaller DAG sizes would have higher opportunity to match the available number of processors, before the crossover points, than large DAG sizes. Accordingly, SL_m of smaller DAG sizes on low number of processors would be close to SL_o, as shown in Fig. 6 by the relationship between the *NSL* and the number of processors, *m*. However, the high potential degree of parallelism found in large DAG sizes would not have an opportunity for exploitation with a limited number of processors. Accordingly, their SL_m values would be much larger than SL_o due to processing tasks allocated to the same processor sequentially. But the situation changes after the crossover points, as more processors become available, since they can match the potential degree of parallelism found in high DAG sizes. The discrepancies in the crossover points among the three curves is attributed to the average *PCCR* of the DAGs.

It is very clear that the speedup does not scale linearly with the number of processors. Definitely, synchronous communication has a serious drawback on limiting the

degree of speedup that can be achieved. To clarify this point, we compare between the results of the utilization, *U(m)*, and the efficiency, *E(m)*. Figure 8(a), (b) and (c) shows the *U(m)* versus the number of processors, *m*, for *PCCR* values equal to 0.5, 1.0 and 5.0, respectively, using three DAG sizes of 50, 250 and 500 nodes. Similarly, Fig. 9(a), (b) and (c) shows *E(m)* versus the number of processors using the same parameters. Both measures would have $U(m) = E(m) = 1$ at $m=1$. However, at $m = 2$ the utilization is high, it is greater than 87%. While the efficiency drops sharply to a value as low as 17%. The differences between the values of *U(m)* and *E(m)* are attributed to the Synchronization Overhead Ratio, *SOR(m)*, as shown previously. This is the blocking time due to synchronous communication, during which a sender is waiting for an acknowledgement from a receiver. The difference between *U(m)* and *E(m)* is very wide in the case of fine granularity tasks, but it becomes narrower as the granularity increases. The results of the Load Imbalance Factor, *LIF(m)*, are shown in Fig. 10(a), (b) and (c). They are taken for $\alpha = 1.0$ and *PCCR* values equal to 0.5, 1.0 and 5.0, respectively. In general, the behavior of the *LIF(m)* results is the complement of the *U(m)*.

5. Conclusions

This work has introduced a low cost algorithm, called GLB-Synch. It is intended to perform task mapping and ordering, in the context of synchronous communication as a part of a multi-step scheduling approach. We have shown by analysis that the complexity of the GLB-Synch algorithm is $O(mc + mv \log v + ev)$. The simulation results show that a multi-step scheduling using the NLC_SynchCom and the GLB_Synch algorithms for clustering, and mapping and ordering, respectively, retain the same low complexity cost for both steps. The performance study has shown limited speedup gain over different DAG shape factors. The limitations in the achieved speedups are mainly attributed to the synchronization overhead. Further improvements in clustering and mapping techniques are needed to achieve high performance, unless the objective for synchronous parallel and distributed computing is otherwise.

Acknowledgments

I gratefully acknowledge the support provided by Sultan Qaboos University through Research Grant No. IG/SCI/COMP/02/02 in facilitating and encouraging an environment conducive to research and academic excellence in the Sultanate of Oman.

References

Arafeh, B., 2003, "Non-linear Clustering Algorithm for Scheduling Parallel Programs with Synchronous Communication on NOWs," *Int. J. of Computers and Their Applications*, Vol. 10(2), pp. 103-114.

Foster, I. and Kesselman, C., 1997, "Globus: A Metacomputing Infrastructure Toolkit," *The Int. J. of Supercomputing Applications and High Performance Computing*, Vol. 11(2), pp. 115-128.

Gerasoulis, A. and Yang, T., 1993, "On the Granularity and Clustering of Directed Acyclic Task Graphs," IEEE TPDS, Vol. 4(6), pp. 686-701.

Hwang, K., 1993, "Advanced Computer Architecture: Parallelism, Scalability, Programmability". MacGraw-Hill, Inc., New York.

Kadamuddi, D. and Tsai, J., 2000, "Clustering Algorithm for Parallelizing Software Systems in Multiprocessors," IEEE Transactions on Software Engineering, Vol. 26(4), pp. 340-361.

Kasahara Lab., [http://www.kasahara.elec.waseda.ac.jp/] (viewed in July 2004), Japan.

Kim, S. J. and Browne, J.C., 1988, "A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures," *Proceedings of Int. Conf. on Parallel Processing*, Vol. II, pp. 1-8.

Kowk, Y. K. and Ahmad, I., 1999, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," ACM Computing Surveys, Vol. 31(4), pp. 406-471.

Kowk, Y. K. and Ahmad, I., 1996, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs onto Multiprocessors," IEEE TPDS, Vol. 7(5), pp. 506-521.

Lee, H., Kim, J., Hong, S. and Lee, S., 2003, "Task Scheduling Using a Block Dependency DAG for Block-Oriented Sparse Cholesky Factorization," Parallel Computing, Vol. 29(1), pp. 135-159.

Liou, J. C. and Palis, M. A., 1997, "A Comparison of General Approaches to Multiprocessor Scheduling," Proceedings of 11th *Int. Parallel Processing Symposium*, pp. 152-156.

Papadimitriou, C. and Yannakakis, M., 1999, "Towards an Architecture-Independent Analysis of Parallel Algorithms," *SIAM J. on Computing*, Vol. 19(2), pp. 322-328.

Radulescu, A., 2001, "Compile-Time Scheduling for Distributed-Memory Systems". Ph.D. Thesis, Faculty of Information Technology and Systems, Delft University of Technology, Delft, The Netherlands.

Sarkar, V., 1989, "Partitioning and Scheduling Programs for Execution on Multiprocessors". Cambridge, MA:MIT press.

Shirazi, B., Wang, M. and Pathak, G., 1990, "Analysis and Evaluation of Heuristic Methods for Static Scheduling," J. *of Parallel and Distributed Computing*, Vol. 10, pp. 222-232.

Wu, M. Y. and Gajski, D.D., 1990, "Hypertool: A Programming Aid for Message-Passing Systems," IEEE TPDS, Vol. 1(3), pp. 330-343.

Yang, T. and Gerasoulis, A., 1994, "DSC: Scheduling

Parallel Tasks on Unbounded Number of Processors," IEEE TPDS, Vol. 5(9), pp. 951-967.

Yang, T. and Gerasoulis, A., 1992, "List Scheduling with or without Communication Delays," Parallel Computing, Vol. 19, pp. 1321-1344.