A Cluster-based Strategy for Scheduling Task on Heterogeneous Processors

Cristina Boeres*, José Viterbo Filho and Vinod E. F. Rebello*
Instituto de Computação, Universidade Federal Fluminense (UFF)
Niterói, RJ, Brazil
e-mail:{boeres,jfilho,vinod}@ic.uff.br

Abstract

Efficient task scheduling is fundamental for parallel applications to achieve good performance on distributed systems. While extensive work exists for scheduling tasks on homogeneous processors, fewer algorithms exist for the more common problem of scheduling in heterogeneous processor environments. In this paper, we propose coupling a replication-based clustering heuristic for homogeneous processors, with a mechanism to map the generated clusters to the heterogeneous environment. Experimental results show that this strategy compares favourably in terms of the makespan with traditional list scheduling approaches to this problem, particularly when communication costs are high.

1. Introduction

Much effort has been applied to solving the problem of scheduling parallel applications efficiently on distributed memory systems. Scheduling aims to minimise the execution time of an application by balancing two objectives: maximising the parallelism that can be exploited given the resources available; while, at the same time, reducing the effects of communication. Although estimating computation and communication costs precisely may be difficult, scheduling techniques can still use approximate values to relieve the programmer of the burden of partitioning the application for each target system [5].

The problem of scheduling an application onto a set of *heterogeneous* resources considering inter-processor communication is NP-complete [11]. In this problem, processing rates of individual processors and communications delays between pairs of processors may differ. In recent years, a variety of static scheduling strategies have been proposed for heterogeneous processors [1]. Interestingly, the majority of these heuristics belong to the class of *list scheduling* algorithms [1, 17]. This is probably due to their relative simplicity and to the fact that algorithms in this class have been shown to have a lower complexity in comparison with other approaches such as clustering algorithms or metaheuristics and thus, favour faster execution times. Furthermore, list scheduling algorithms can easily be adopted when only a limited number of processors are available. It is known, however, that for the homogeneous processor scheduling problem the results obtained by this class of heuristics tend not to be as good as those produced by *clustering algorithms* [1, 5], particularly when communication costs are higher than the average computation costs [2].

A number of replication-based algorithms have been proposed in [4], based on a clustering algorithm design methodology [3]. These algorithms schedule applications, represented by task graphs, on an unbounded number of homogeneous processors under a LogP communication model [9] and generate schedules whose makespans compare favourably with those produced by other well known LogP and delay model scheduling algorithms [3, 4]. A theoretical analysis in [5] showed that those algorithms generate optimal schedules for the delay model under the small communication condition [7]. Clustering algorithms typically consist of two stages. During the first stage, tasks are clustered together with their predecessors to reduce communication costs. The second stage then maps the resulting clusters to the available processors of the target system.

Clustering algorithms tend to be successful because they adopt a more global view, while list scheduling algorithms, on the other hand, tend to only make local optimisation decisions. The work presented here aims to analyse the benefits of modifying the second phase of the methodology to map the clusters of tasks to a given set of heterogeneous processors. The proposed mapping algorithm, which uses list scheduling, explores characteristics of the clusters and the target system.

While the following section reviews the scheduling definitions used in this paper, Section 3 presents an overview of a few of the static schedulers for heterogeneous proces-

* The authors are partially funded by research grants from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil.

Proceedings of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'04)
1550-6533/04 $ 20.00 IEEE

sors published in the literature. The proposed two stage approach proposed is described in Section 4, with a brief summary of the initial clustering phase and details of the mapping algorithm. Section 5 presents our experimental environment, and a summary of the results obtained from an initial evaluation. Finally, some conclusions and on-going work are outlined in Section 6.

2. Definitions

This work considers the class of parallel applications in which each program can be represented by an directed acyclic graph (DAG). Tasks within these applications are indivisible processing units characterised by the input data received from, and the output data sent to, other tasks. In a *DAG* $G = (V, E, \varepsilon, \omega)$: the set of vertices, V, represent *tasks*; E, the precedence relation among them; $\varepsilon(v)$ is the amount of work associated to task $v \in V$; and $\omega(u, v)$ is the weight associated to the edge $(u, v) \in E$, representing the amount of data units transmitted from task u to v. The set of immediate predecessor and successor tasks of v is given by $pred(v)$ and $succ(v)$, respectively.

In the architectural model adopted, $P = \{p_0, \ldots, p_{m-1}\}$ is the set of m processors and $h(p_i)$ the heterogeneity factor of processor p_i such that the execution time of task v on p_i is given by $\varepsilon(v) \times h(p_i)$. We also consider that when two adjacent tasks, say, u and v, are allocated to distinct processors p_u and p_v, respectively, the cost associated with the communication of $\omega(u, v)$ data units is $L(p_u, p_v) \times \omega(u, v)$, where the latency $L(p_u, p_v)$ is the transmission time per byte incurred on the communication link between p_u and p_v.

As defined in the literature [17], the *bottom level* (also known as the $blevel$ or $level$) of task v is the longest path in DAG G from v to a sink task, taking into account the computation and communication costs associated with the input graph and the target system. In heterogeneous systems, scheduling algorithms often use the average cost values of unscheduled tasks and edges in $succ(v)$ to calculate $blevel(v)$ [14, 17]. We define the *scheduled bottom level* of a task v as the bottom level of v in a given schedule, i.e. $blevel_s(v, p_j)$ of task v allocated to processor p_j is

$$\varepsilon(v) \times h(p_j) + \max_{w \in succ(v)} \{comm(v, w) + blevel_s(w, p_w)\}$$

where the communication cost

$$comm(v, w) = \begin{cases} \omega(v, w) \times L(p_j, p_w), \text{ if } p_j \neq p_w \\ 0, \text{ otherwise} \end{cases} \quad (1)$$

and where p_j and p_w are the processors to which the tasks v and w are allocated in the given schedule.

3. Related Work

In the literature, list scheduling algorithms are hugely popular due to their low complexity [2, 13] and their easy applicability to heterogeneous networks of processors [1, 17]. The order in which tasks are chosen and to which processor they are scheduled are both determined by priorities. Differences between algorithms are principally due to the choice of priorities adopted and whether techniques such as task replication, are used to overcome the limitations of the list scheduling approach.

The algorithm *Heterogeneous Earliest Finish Time* (HEFT) is considered one of the best heuristics for scheduling tasks onto heterogeneous processors [17]. Prior to making scheduling decisions, HEFT orders all tasks in decreasing order of their *blevel*, breaking ties at random. During the scheduling phase, the algorithm selects the unscheduled task v with highest priority and looks for the earliest time that v can start on each processor p_j, *inserting* v in idle periods between two previously scheduled tasks if possible. The processor chosen is the one where task v will finish earliest. As shown in [17], HEFT outperforms four well known list scheduling algorithms in 86% of the cases, with average improvements on the makespans ranging from 7% to 52%.

The *Task Duplication based Scheduling* (TDS) [16] schedules a DAG on heterogeneous processors which are fully connected by homogeneous links. The heuristic is composed of the following steps. Initially, for each task $v \in G$, the earliest start and finish times are calculated in order to identify v's favourite predecessor and processor, i.e. the predecessor that should be scheduled together with v and the processor that minimises v's finishing time. The next step is to schedule the tasks (in order of their $blevel()$), based on the favourite predecessor, favourite processor and the latest start and finish times of the selected task. In the final step, a duplication procedure attempts to replicate the favourite predecessor task of v on its processor if the makespan decreases. The results were compared with the *Best Imaginary Level* (BIL) heuristic [15] and for coarse grained DAGs ($0.2 < ccr \leq 1$, where ccr is the communication to computation ratio), TDS outperformed BIL. For fine grained ($ccr > 1.0$), TDS still produced better results, but the improvements were less significant.

The *Level Duplication Based Scheduling* algorithm (LDBS) [10] is a list scheduling approach which uses replication to schedule a DAG onto a heterogeneous system (where processors and links are heterogeneous). LDBS schedules tasks of the same topological level[1] on the pro-

1 The task topological level only considers the precedence relationship and not the costs associated with computation and communication.

cessors that minimise their finishing times. Let $List_j$ be the list of tasks with the same topological level j. At each iteration, the task $v \in List_j$ with the highest $blevel$ is selected. As in HEFT, the algorithm attempts to schedule v within idle periods between previously scheduled tasks. The immediate predecessors are replicated if the finish time of v is reduced. The schedules produced by LDBS were on average 19% and 29% better than those of HEFT for a *ccr* of 1.0 and 10.0, respectively. For $ccr = 0.1$, however, the makespans of HEFT were 2% better than LDBS [10].

4. Clustering Tasks onto Heterogeneous Processors

This work aims to analyse the benefits, over the traditional list scheduling approach, of employing a cluster-based strategy to the heterogeneous scheduling problem. The two stage cluster-based strategy adopted in this initial investigation assumes a virtual homogeneous environment (an unlimited number of the slowest processor) while creating task clusters using the clustering algorithm design methodology [4]. In a second stage, a subset of the generated clusters are mapped to the heterogeneous environment (i.e. the fixed number of available processors), considering the execution cost characteristics of the tasks on the processors and the transmission characteristics on the communication links. In principle, the second stage mapping algorithm could also be used in conjunction with a variety of other homogeneous processor scheduling heuristics allowing them to be applied in heterogeneous environments.

Topologically for each task $v \in G$, the first stage of the clustering algorithm design methodology attempts to create a cluster $C(v)$ (v being known as the owner of $C(v)$), containing the set of ancestor tasks (not only immediate predecessors) which allows the *owner* to be scheduled as early as possible. Note that these ancestors are in fact copies, and thus it is possible for a task to exist in more than one cluster. The *scheduled time* of $v \in C(v)$ is given by $s(v)$. Once all $|V|$ clusters have being created, a schedule with makespan $\max_{v \in V}\{s(v) + \varepsilon(v) \times h(p_i)\}$, for a virtual homogeneous environment of slow processors ($h(p_i)$ being the heterogeneity factor of the slowest processor in P), can easily be constructed using (a subset of) the clusters. The objective of the first stage is to coarsen the granularity of the input DAG G in accordance with the communication aspects of the target system. The first stage attempts to not only reduce the communication costs, but also, the number of messages required to implement the schedule. A more indepth description of the methodology can be found in [3, 4].

During the second stage, the proposed *Cluster Mapping Algorithm* (CMA), based on a list scheduling framework, maps the set of clusters of the virtual schedule to the heterogeneous target environment. The mapping algorithm is summarised in Algorithm 1. Let LC be the list of clusters produced during the first stage, with each cluster $c \in LC$ being associated with to a virtual processor p'_c. Initially, all cluster in LC are considered unmapped, i.e. the set C_{umap} contains a tuple $\langle c, p'_c \rangle$ for all $c \in LC$ (line 2). During an iteration of the mapping process (lines 3 to 7), two phases are identified: the *cluster choice* phase, when the next cluster c is selected to be mapped to a processor of the target architecture in accordance with a given priority (line 4); and the *processor choice* phase, where another priority is used to select the processor p to which the chosen cluster will be assigned (line 5). The tuple $\langle c, p \rangle$ is then included in the list of mapped clusters C_{map} (line 7).

Algorithm 1 : *Cluster Mapping Algorithm* (LC)

```
1  C_map = ∅; C_umap = ∅;
2  for all c ∈ LC do C_umap = C_umap ∪ ⟨c, p'_c⟩;
3  while LC ≠ ∅ do
4      c = cluster choice(LC);
5      p = processor choice(c, P) ;
6      LC = LC − {c};
7      C_map = C_map ∪ ⟨c, p⟩;
```

The *Cluster Choice* Phase

One of the crucial issues to be tackled when scheduling a DAG is the order in which tasks are chosen to be assigned to a processor. The main conclusions found in the literature have lead heuristics to use priorities that are recalculated at each iteration of the scheduling process (so-called dynamic priorities) [13]. These priorities are related to the costs which impact the execution of the application on the target system. During the mapping process, the priority used to select the next cluster to be mapped is equally important. During this study, we also concluded that in order to tackle both the limitation on the number of processors and the minimisation of the final makespan, the priorities adopted should also be dynamic, i.e. updated after each mapping iteration, to take into consideration the influence of unmapped clusters on the partial schedule.

The *cluster choice* phase of CMA uses a dynamic variant of the bottom level priority to select the next cluster c to be mapped. Suppose that $c = \langle u_1, u_2, \ldots, v \rangle$. The *bottom level* of cluster c is $blevel_c(c) = blevel_s(u_1, p_c)$, where p_c is either the virtual processor to which cluster c was allocated in the first stage or the actual physical processor to which it has already been mapped during the second, and where u_{i+1} is treated as an immediate successor of u_i. The bottom level of the unmapped clusters are re-calculated and the one with the highest value is selected. In the case of ties, the cluster with the least *affinity* with those already sched-

uled will be chosen. We define the affinity of cluster c to be

$$aff(c) = \max_{\forall p \in P}\{\sum_{\forall u \in c \ \cap u \in p} \varepsilon(u)\}$$

Ties are then broken by choosing the cluster with the highest execution weight.

The *Processor Choice* Phase

Traditionally, list scheduling algorithms adopt a strategy to allocate each task to the processor that minimises its finish time. In the case of CMA, however, first whole clusters rather than individual tasks are being allocated to processors, and second, priority is given to critical path clusters which means that a cluster may be remapped before one of its predecessors. Therefore, it is more important to minimise the overall makespan of all mapped clusters than just the finish time of the cluster currently being mapped.

During the *processor choice* phase, the best processor p_i to host cluster c is chosen by evaluating the following criteria, as shown in Algorithm 2: p_i is the processor that minimises the makespan of the set of currently mapped clusters (lines 3 to 7); in the case of ties (line 8) (i.e. when more than one processor offers the same finish time), p_i is the processor that minimises the makespan of the current partial schedule at this current iteration (assumes that unmapped clusters will be allocated to distinct copies of the slowest processor) (lines 9 to 12). Finally, if there are still ties (line 13), then the processor that minimises the makespan of cluster c is chosen (lines 14 and 16).

In order to calculate the makespan when cluster c is inserted to processor p_i, the tasks in c have to be merged with those tasks already allocated to p_i and multiple copies of a given task be eliminated. The execution order of the tasks in p_i obeys the precedence relationship given in G and the original order established in the clusters when generated by the first stage. Let $ltask_c$ be the ordered list of tasks in cluster c and $ltask_p$, the ordered list of tasks already in p_i. Let v and v' be tasks in $ltask_c$ and $ltask_p$, respectively, and the *latest message arrival time* of a task t in p_i be

$$lmat(t, p_i) = \max_{u \in pred(t)}\{s(u) + \varepsilon(u) \times h(p_i) + comm(u, t)\}$$

where $comm(u, t)$ is defined in Equation 1. The resulting order of v and v' in p_i is determined by scheduling the task with the smallest $lmat()$, ties being broken by using the highest $blevel_s$, first.

5. Experimental Results

The complexity of the scheduling problem has lead researchers to attempt to tackle the problem for specific classes of DAGs, be it in accordance with the topology (e.g. join, forks, trees, diamond, or irregular), with the *granularity*[2] or both. Typically, the graph topologies chosen represent specific classes of applications, while varying the granularity can also account for a variety of target systems. Adopting such an approach not only allows theoretical results to be obtained (e.g. optimality bounds) for specific problem instances but also identifies the parameter space for which an experimental analysis of proposed mechanisms should focus. Heterogeneous environments however add further complexity to the problem by, for example, making concepts such as granularity and priorities (used extensively by list scheduling heuristics for the homogeneous processors problem) difficult to calculate or apply. Thus, it is hard to find a scheduling algorithm which performs well for both the homogeneous and the heterogeneous scheduling problems.

In this section, we compare the makespans produced by our two stage approach – Clustering for Heterogeneous Processors (CHP) – with two list scheduling algorithms, one which uses task replication, LDBS [10], and HEFT which does not [17]. The experimental analysis in this paper is based on a suite of unit execution time, unit data transfer graphs (UET-UDT) DAGs which includes both regular (various sizes of tree and diamond graphs) and irregular graphs (both randomly generated and graphs from the literature) [3]. Also considered are the Peer Set Graphs (PSG), irregular DAGs with arbitrary costs taken from the literature, proposed by Kwok and Ahmad to be used as part of a benchmark suite for comparing scheduling algorithms [13], as well as two real world applications: Gaussian Elimination (GE) [8] and a physics problem in molecular dynamics (MD) [12].

In the experiments with the UET-UDT DAGs, we defined a computation factor F, so that the execution weight of any task $v \in G$ would be F. We also varied the latency for different experiments so that the communication costs associated with each edge would be equal to L. In this way, we were able to define more accurately fine and coarse grained applications. We carried out a series of experiments with MD, in-tree, out-tree, diamond and random DAGs, considering the following parameter pairs (F, L) for each input DAG: $(1, 1)$, $(5, 1)$ and $(10, 1)$ which characterise coarse grained DAGs, and, for fine grained ones the parameters pairs were: $(1, 5)$ and $(1, 10)$. Since the GE DAGs are already coarse grained, only the parameters pairs $(1, 1)$, $(1, 5)$ and $(1, 10)$ were evaluated. We did not vary the granularity of the PSGs, thus the only parameter used was $(1, 1)$.

2 Although there exists a number of both formal and mathematical definitions for the granularity of a DAG [13], loosely speaking it is considered to be the relationship or ratio between the amount of communication and computation.

Algorithm 2 : *processor choice* (c, P)

```
1  C_umap = C_umap − ⟨c, p'_c⟩;  C_all = C_map ∪ C_umap;
2  for all p_i ∈ P
3      if mspan(C_map ∪ ⟨c, p_i⟩) < mspan* then {
4          mspan* = mspan(C_map ∪ ⟨c, p_i⟩);
5          mspan*_all = mspan(C_all ∪ ⟨c, p_i⟩);
6          mspan*_c = mspan(⟨c, p_i⟩);
7          proc* = p_i;
8      } else if (mspan(C_map ∪ ⟨c, p_i⟩) == mspan*) then
9          if (mspan(C_all ∪ ⟨c, p_i⟩) < mspan*_all) then {
10             mspan*_all = mspan(C_all ∪ ⟨c, p_i⟩);
11             mspan*_c = mspan(⟨c, p_i⟩);
12             proc* = p_i;
13         } else if (mspan(C_all ∪ ⟨c, p_i⟩) == mspan*_all) then
14             if (mspan(⟨c, p_i⟩) < mspan*_c) then {
15                 mspan*_c = mspan(⟨c, p_i⟩);
16                 proc* = p_i;
                   }
17 return (proc*);
```

In this initial work, we assumed that the heterogeneous processors were interconnected by a homogeneous network (a condition also adopted elsewhere [6, 16]) and which can be viewed as modeling a collection of different workstations or PCs interconnected via a switch. In this paper, we focus on investigating how the following aspects affect the proposed scheduling algorithm for each type of graph: first, the relative number of fast and slow processors in a dual speed environment; and second, the diversity or degree of heterogeneity of the environment. For each graph and parameter pair (F, L), a number of computing environments were considered. Depending on the DAG size, the total number of processors m in P was set to 20, 30 or 50. In the dual speed investigation, each computing environment consisted of x processors with $h() = 0.5$ and $m - x$ with $h() = 1.0$, with x varying from 0 to m in increments of 10% of m. In the diversity investigation, the m processors are assigned heterogeneity factors randomly with a standard deviation of 0.00 (homogeneous) 0.125, 0.25 and 0.5, for each environment, respectively. Tables 2, 4, 6 and 7 present, for each class of DAG, a pair-wise comparison between the three algorithms considered in terms of makespans of the schedules generated for all of the environments examined.

Overall Results

Tables 1(a) and (b) present the average improvement of CHP over HEFT and LDBS for all of the dual and diversity environment experiments. The columns $FP\%$ (fast processors) in Table 1(a) identify the percentage of processors with $h() = 0.5$. For the dual environments, CHP provides very reasonable improvements particularly when the number of fast processors is not restrictive. For the diversity environments, Table 1(b) shows, on the whole, that while CHP is clearly better for homogeneous environments, it still provides reasonable improvements for heterogeneous environments as well. The remainder of this section analyses the results in more details for each the class of DAG and granularity.

$FP\%$	HEFT	LDBS	$FP\%$	HEFT	LDBS
100%	25.21%	10.51%	50%	22.90%	9.28%
90%	25.21%	10.51%	40%	21.71%	8.32%
80%	25.03%	10.33%	30%	18.48%	5.72%
70%	24.56%	9.86%	20%	17.02%	4.85%
60%	23.50%	9.69%	0%	21.22%	9.08%

(a)

Std. Dev.	HEFT	LDBS
0.000	23.73%	12.06%
0.125	17.41%	6.00%
0.250	17.07%	4.97%
0.500	17.11%	3.79%

(b)

Table 1. Overall average percentage improvements over HEFT and LDBS for the (a) Dual and (b) Diversity Environments.

The UET-UCT DAGs

In-Trees Although the schedules for in-tree DAGs do not benefit from duplicating tasks (since each task has only one immediate successor), task replication does bring advantages to LDBS over HEFT. It allows LDBS to effectively undo scheduling decisions made in earlier iterations. For coarse grained in-trees with parameters $(10, 1)$, $(5, 1)$ and $(1, 1)$, CHP performs poorly compared to HEFT and LDBS particularly when the number of fast processors is low.

In the dual speed environments, CHP is worse than both HEFT and LDBS by 11.8% for $(10, 1)$ and 9.9% for $(5, 1)$. For $(1, 1)$, although CHP is 10.6% better than HEFT, it is still 14.8% worse than LDBS. In the diversity environments, results are even worse: CHP is worse than both HEFT and LDBS by 28.8% for $(10, 1)$ and 32.8% for $(5, 1)$. For $(1, 1)$, CHP is 7.3% worse than HEFT and 19.3% worse than LDBS. The reason for these results can be attributed to the size of the clusters generated by the first stage of CHP and to the fact that with a limited number of fast resources, the mapping becomes a load balancing problem. This is harder to solve efficiently with clusters of tasks rather than with individual tasks. CHP needs to either generate smaller clusters during the first stage or be able to remove tasks from clusters during the second stage (future work).

As the graphs become finer, CHP outperforms both HEFT and LDBS in both environments. For the dual speed (diversity) environments, the makespans were 48.6% (45.6%) better than HEFT for $(1, 5)$ and 66.6% (for the diversity, 63.7%) for $(1, 10)$. In relation to LDBS, the improvements were 9% (1.4%) for $(1, 5)$ and 37% (26.6%) for $(1, 10)$. For fine grained DAGs, CMA benefits greatly by the type of clusters generated in the first stage. Even-though an unlimited number of (slow) processors were assumed, tasks are clustered into *sub in-trees*, so that CMA actually only has to map a coarser graph onto the limited number of processors. The results for HEFT and LDBS are somewhat expected, since list scheduling tends to produce near optimal solutions for coarse grained graphs but relatively poor results for fine graned in-trees due to their greedy approach [2]. For the diversity experiments, while CHP is on average better than HEFT by 18.9% and LDBS by 2% for a homogeneous environment, it is only 4.9% better and 14.6% worse for the heterogeneous ones.

Out-Trees It has been shown that the first stage of CHP produces the optimal schedules for out-tree DAGs on homogeneous processors [5]. The tasks in each path of the out-tree are clustered together and thus no communication is necessary between clusters. At each execution of the *cluster choice* priority in CHP, all clusters in C_{umap} have the same bottom level values (and weights). Therefore, CHP uses the affinity priority to select clusters with the fewest tasks in common with those already scheduled. Initially, these clusters will be scheduled to idle fast processors. Once all the fast processors have been allocated a cluster, the algorithm attempts to minimise the makespan by mapping the next cluster to either an idle slow processor or to the fast processor with which the cluster has the most affinity (i.e. tasks in common). For out-trees, CHP produces schedules which are never worse than LDBS and HEFT. For both environments, CHP is, on average, better than HEFT and LDBS by 25.4% and 20.5%, respectively, for the parameter pair $(10, 1)$. As the graphs become more fine grained, the results improve further (32.8% better than HEFT and 23.9% better than LDBS, for $(5, 1)$, and 60.8% and 43.6%, respectively, for $(1, 1)$) to as much as 80% better than HEFT and 65% better than LDBS for both $(1, 5)$and $(1, 10)$. Even when the environments become more heterogeneous, CHP is still 52.3% and 39.5% better on average than HEFT and LDBS, respectively.

Diamond In the case of coarse grained diamond graphs, the first stage produces linear clusters (clusters without independent tasks) and the degree of task replication is usually greater than that of the schedules produced by LDBS. This is due to the fact that the cluster are constructed considering an unlimited number of processors. Due to the structure and number of clusters passed to the mapping stage, CHP ends up using, on average, more processors than LDBS and HEFT. Therefore, when there are few fast processors, CHP produces schedules with larger makespans. On the other hand, for fine grained DAGs, replication hides the high communication costs thus benefiting the schedules produced by CHP.

		HEFT	LDBS	CHP
HEFT	$>$		685	642
	$=$		132	80
	$<$		23	118
LDBS	$>$	23		500
	$=$	132		145
	$<$	685		195
CHP	$>$	118	195	
	$=$	80	145	
	$<$	642	500	

Table 2. A pair-wise comparison in terms of the number of worse, equal and better schedules for the UET-UCT DAGs

In the dual speed environments, while for $(10, 1)$, $(5, 1)$ and $(1, 1)$, CHP was, on average, 2.5%, 0.2% and 0.7% worse than LDBS, for HEFT it was 0.5%, 6.9% and 24% better, respectively. In the case of $(1, 5)$, CHP was better

than LDBS and HEFT by 6.7% and 29.1%, and for $(1, 10)$, was 12% and 19.6% better. In the diversity environments, the results followed a similar trend, ranging from 0.1% (for $(10, 1)$) to 33% (for $(1, 10)$) better than HEFT and 1.4% worse to 18.9% better than LDBS. As the environments become more heterogeneous, the overall improvements of CHP decline from 21.4% to 15.6% better than HEFT, and 7.5% to 1.6% better than LDBS.

Table 3 summarises the overall improvements over HEFT and LDBS for all of the UET-UCT DAGs.

	Dual Speed		Diversity	
(F, L)	HEFT	LDBS	HEFT	LDBS
$(10, 1)$	1.79%	0.63%	−6.72%	−7.87%
$(5, 1)$	6.06%	2.60%	−4.15%	−6.94%
$(1, 1)$	27.35%	5.53%	17.50%	2.62%
$(1, 5)$	47.34%	21.05%	47.83%	17.99%
$(1, 10)$	51.22%	32.12%	53.42%	29.82%

Table 3. Average percentage improvements over HEFT and LDBS for the UET-UCT DAGs.

Gaussian Elimination

For the GE DAGs, the results are similar to the diamond DAGs. For the dual speed environments, CHP was, on average, 1.6% and 0.1% worse than HEFT and LDBS for parameter pair $(1, 1)$, respectively, improving to 16.9% and 4% for $(1, 5)$, 22.3% and 6.5% for $(1, 10)$. In the case of diversity environments, the results were almost identical: CHP was worse than both HEFT and LDBS by an average of 1.2% for $(1, 1)$, improving to 11.4% and 5.1% for $(1, 5)$, 18.9% and 6.6% for $(1, 10)$. As the number of fast processors becomes smaller, the improvements over HEFT and LDBS diminish. Again, the CMA is hampered by the type of clusters generated by the first stage and would benefit from an optimisation phase to "decluster" or remove some of the tasks from a given cluster. CHP performs better in environments which presents less diversity, ranging from 11.8% to 6.6% better for HEFT and 7.8% better to 1.7% worse than LDBS.

Molecular Dynamics Code

The Molecular Dynamics Code is in fact an irregular UET-UDT DAG [12] for which CHP produces results, in both environments, similar to those for the random DAGs. CHP was slightly worse than LDBS in the dual speed environments for the $(1, 1)$ and $(10, 1)$ cases by 3.1% and 0.2%, respectively. For each of the other parameter pairs,

		HEFT	LDBS	CHP
HEFT	$>$		112	145
	$=$		51	9
	$<$		5	14
LDBS	$>$	5		132
	$=$	51		13
	$<$	112		23
CHP	$>$	14	23	
	$=$	9	13	
	$<$	145	132	

Table 4. A pair-wise comparison for GE DAGs.

	Dual Speed		Diversity	
(F, L)	HEFT	LDBS	HEFT	LDBS
$(10, 1)$	0.0%	−0.1%	−9.7%	−11.4%
$(5, 1)$	3.5%	0.7%	−2.2%	04.7%
$(1, 1)$	16.9%	−3.2%	8.9%	−5.5%
$(1, 5)$	22.8%	8.1%	34.6%	9.5%
$(1, 10)$	9.5%	10.5%	24.2%	14.7%

Table 5. Average percentage improvements over HEFT and LDBS with respect to the parameter pairs (F, L) for the Dual Speed and Diversity Environments for the MD DAG.

improvements were obtained particularly for fine granularity. Table 5 presents the average improvements while Table 6 shows the number of times that CHP schedules were better than the other two algorithms.

		HEFT	LDBS	CHP
HEFT	$>$		49	60
	$=$		11	1
	$<$		10	9
LDBS	$>$	10		44
	$=$	11		10
	$<$	49		16
CHP	$>$	9	16	
	$=$	1	10	
	$<$	60	44	

Table 6. A pair-wise comparison for MD.

Non UET-UCT irregular DAGs

CHP also produced good schedules for the irregular graphs in PSG (see Table 7). The improvements in the makespan were, on average, 22.4% better than HEFT, and

15.9% better than LDBS for the dual speed environments. For the diversity environments, the CHP was 19.6% and 14.2% better than HEFT and LDBS, respectively. These are quite encouraging results, since in addition to having an irregular topology, non-uniform execution and communication weights are associated to the vertices and edges.

		HEFT	LDBS	CHP
HEFT	>		98	132
	=		36	21
	<		20	1
LDBS	>	20		124
	=	36		21
	<	98		9
CHP	>	1	9	
	=	21	21	
	<	132	124	

Table 7. A pair-wise comparison for the PSGs.

6. Conclusions and on-going work

We presented an alternative to the traditional list scheduling approach adopted to address the problem of scheduling tasks onto heterogeneous processors. The CHP strategy consists of two stages: initially, clusters of tasks are formed assuming homogeneous slow processors which are then assigned to the available heterogeneous processors by a list scheduling based mapping algorithm. Based on an initial evaluation, the schedules produced compare favourably to well known list scheduling strategies for this problem. CHP outperforms HEFT and LDBS significantly for fine granularities; when the number of fast processors is not severely restricted; and when the degree of heterogeneity in the system is low. On-going work is investigating how the number and type of the clusters generated by the first stage affect the mapping stage; developing mechanisms to reduce the degree of replication and methods to modify clusters (decluster tasks) during the mapping phase.

References

[1] O. Beaumont, A. Legrand, and Y. Robert. Static scheduling strategies for heterogeneous systems. *Computing and Informatics*, 21:413–430, 2002.

[2] C. Boeres, G. Chochia, and P. Thanisch. On the scope of applicability of the ETF algorithm. In Afonso Ferreira and Jose Rolim, editors, *The 2nd International Workshop on Parallel Algorithms for Irregularly Structured Problems (IRREGULAR'95)*, LNCS 980, pages 159–164, Lyon, France, September 1995. Springer.

[3] C. Boeres, A. P. Nascimento, and V. E. F. Rebello. Cluster-based task scheduling for LogP model. *International Journal of Foundations of Computer Science*, 10(4):405–424, 1999.

[4] C. Boeres and V.E.F. Rebello. On the design of clustering-based scheduling algorithms for realistic machine models. In *The Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01)*, San Francisco, USA, April 2001. IEEE Computer Society Press.

[5] C Boeres and V.E.F. Rebello. Towards optimal task scheduling for realistic machine models: Theory and practice. *The International Journal of High Performance Applications*, 17(2):173–190, 2003.

[6] T-Y. Choe and C-I. Park. A task duplication based scheduling algorithm with optimality condition in heterogeneous systems. In *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'02)*, pages 531– 536, Vancouver, B.C., Canada, Aug 2002. IEEE Computer Soc.

[7] J.Y. Colin and P. Chrétienne. CPM scheduling with small communication time. *Operational Research*, 39:680–684, March 1991.

[8] M. Cosnard and D. Trystram. *Parallel Algorithms and Architectures*. Int. Thomson Computer Press, 1995.

[9] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, USA, May 1993.

[10] A. Dogan and F Ozguner. LDBS: A duplication based scheduling algorithm for heterogeneous computing systems. In *Proceedings of the International Conference on Parallel Processing (ICPP'02)*, page 352, Vancouver, B.C., Canada, Aug 2002.

[11] D. Fernández-Baca. Allocating modules to processors in a distributed system. *IEEE Trans. Software Eng.*, SE-15(11):1427–143, 1989.

[12] S.J. Kim and J.C. Browne. A general approach to mapping of parallel computations upon multiprocessor architectures. In *Proceedings of the 3rd International Conference on Parallel Processing*, pages 1–8, 1988.

[13] Y-K Kwok and I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422, Dec. 1999.

[14] Y-K Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4), Dec. 1999.

[15] H. Oh and S. Ha. A static heuristic for heterogeneous processors. In *Proceedings of the 2th International Euro-Par Conference on Parallel Processing (Euro-Par'96)*. Springer, 1996.

[16] S Ranaweera and D. Agrawal. A task duplication based scheduling algorithm for heterogeneous systems. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00)*, pages 445–450, Cancun, Mexico, May 2000. IEEE Computer Soc.

[17] H. Topcuoglu, S. Hariri, and M.Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, Mar 2002.