Parallel Computing 37 (2011) 428–438

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters

Jorge G. Barbosa [a,*], Belmiro Moreira [b]

[a] Universidade do Porto, Faculdade de Engenharia, Departamento de Engenharia Informática, Lab. de Int. Artificial e Ciência dos Computadores, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
[b] Universidade do Porto, Faculdade de Engenharia, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal

ARTICLE INFO

Article history:
Available online 9 January 2011

Keywords:
Parallel task scheduling
List scheduling
Image data
Multiple DAG
Non-deterministic job arrival

ABSTRACT

This paper addresses the problem of minimizing the scheduling length (make-span) of a batch of jobs with different arrival times. A job is described by a direct acyclic graph (DAG) of parallel tasks. The paper proposes a dynamic scheduling method that adapts the schedule when new jobs are submitted and that may change the processors assigned to a job during its execution. The scheduling method is divided into a scheduling strategy and a scheduling algorithm. We also propose an adaptation of the Heterogeneous Earliest-Finish-Time (HEFT) algorithm, called here P-HEFT, to handle parallel tasks in heterogeneous clusters with good efficiency without compromising the makespan. The results of a comparison of this algorithm with another DAG scheduler using a simulation of several machine configurations and job types shows that P-HEFT gives a shorter makespan for a single DAG but scores worse for multiple DAGs. Finally, the results of the dynamic scheduling of a batch of jobs using the proposed scheduler method showed significant improvements for more heavily loaded machines when compared to the alternative resource reservation approach.

1. Introduction

A heterogeneous computer cluster is defined in this paper as a set of processors connected by a high speed network, with different processing capacities. The cluster is a dedicated machine as in [1,9,22], as opposed to a geographically dispersed, loosely connected grid. Such clusters can be private clusters of an organization, but also they can be the nodes of a grid infrastructure, which can only provide good performance if at least the end clusters also have good performance. This paper concentrates on optimizing the makespan of a batch of jobs that may arrive at the heterogeneous cluster scheduler at different times.

The task scheduling problem is to allocate resources (processors) to the tasks and to establish an order for the tasks to be executed by the resources [9,22]. In this context, there are two different types of task scheduling: static and dynamic. Static strategies define a schedule at the compile or at launch time based on the knowledge of the processors' availability and the tasks to execute. The aim is usually to minimize the schedule length (or *makespan*) of the tasks [1,10,11,14,22].

Dynamic strategies, on the other hand, are applied when the arrival times of the tasks are not known a priori, so the system needs to schedule tasks as they arrive [9,13,21]. We consider non-preemptive tasks, so that only the tasks waiting to be executed and the newly arrived tasks are considered in the re-scheduling. In this paper, the aim of dynamic scheduling is to minimize the completion time of a batch of tasks, as in [13,21]. Real world applications where such a batch of jobs can occur are

* Corresponding author. Tel.: +351 22 508 18 88.
E-mail addresses: jbarbosa@fe.up.pt (J.G. Barbosa), belmiro.moreira@fe.up.pt (B. Moreira).

0167-8191/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2010.12.004

found in the processing of image sequences, where objects contained in the frames are identified and followed along frames. The objects' shapes may vary along frames, and the problem is therefore known as deformable object tracking [18]. This problem has applications in video surveillance, microscopic imaging processing, and biomechanical analysis. In this context, a multiple job (or DAG) environment may be, for example, the input from several cameras forming a batch of jobs whose processing time should be minimized. The jobs are therefore related, and only the collective result is meaningful.

Dynamic strategies can aim also to minimize some global performance measurements related to the system's quality of service (QoS) [9]. The optimization of performance measurements related to the service quality for a dynamic multi-user environment, where jobs are unrelated, is outside the scope of this paper. However, this is another important problem that will be considered in a future work.

The parallel task model, or mixed-parallelism, combines task and data parallelism at the same time. Data parallelism is common in many scientific applications that are based on linear algebra kernels, for example, and it can exploit heterogeneous clusters as reported in [2]. Also, many applications are described by DAGs, where the edges represent the task precedence as well as the communication load between tasks. However, DAGs with many branches implicitly have tasks that can be executed in parallel, thus resulting in a mixed-parallelism if a single task is assigned to more than one processor. Mixed-parallelism has been widely reported in the literature [1,5–8,12,23].

In this paper, we first present a new adaptation of HEFT [22] for parallel task DAGs on a single heterogeneous cluster, called P-HEFT (P stands for parallel tasks). We then compare the results of this algorithm with those of another mixed-parallel scheduler, the Heterogeneous Parallel Task Scheduler (HPTS) [1], using a simulation of several machine configurations and job types.

We present results that compare the resource reservation policy [15,25] to the one proposed here, which carries out a global management of the machine and allows a floating assignment of processors to the jobs, leading to better makespans when seeking to minimize the completion time of a batch of jobs.

The contributions of the present work are (a) a mixed-parallel version of HEFT [22] for heterogeneous clusters (P-HEFT), (b) a characterization of P-HEFT and HPTS through extended simulations, (c) a first approach to scheduling multiple mixed-parallel jobs dynamically and (d) an evaluation of the best approach, in terms of the makespan metric, to scheduling multiple DAGs that arrive at different times.

This paper is organized as follows: Section 2 defines the scheduling problem and revises related work. Section 3 presents the computational model used in this paper. Section 4 presents the dynamic scheduling method proposed in this paper. Section 5 presents the results, and finally, Section 6 presents the conclusions and future work.

2. Problem definition and related work

The problem addressed in this paper is the dynamic scheduling of a batch of jobs, or applications, represented by directed acyclic graphs (DAGs) on a distributed memory computer (i.e., a cluster). Prior works on dynamic scheduling for a single heterogeneous cluster consider independent tasks and assign one task to one processor (sequential tasks) [9,13,21]. In [21] a genetic algorithm is implemented that can schedule a batch of independent tasks with a short completion time. Here in this paper, each job is composed of a DAG with parallel tasks, which is a more complex problem. The use of a genetic algorithm is not considered here because it would need to be reformulated to schedule dependent tasks, which would then increase the time complexity of the algorithm.

In [13], the performance of an immediate versus a batch mode for a set of independent task scheduling heuristics was compared; the batch mode showed better performance when minimizing the total schedule length. In [9], a set of independent task scheduling heuristics was tested into maximize a quality of service (QoS) measurement defined as the value accrued of completed tasks in a given interval of time. The value of a task is calculated based on the priority of the task and the completion time of the task with respect to its deadlines.

For the problem of scheduling several DAGs on a heterogeneous cluster, i.e., multiple DAG scheduling, [25] presents solutions for DAGs made up of sequential tasks. This is a static approach because the DAGs are known at the beginning of the scheduling. The authors compare four variants of solutions that incorporate all DAGs into a super DAG to the fairness policy that considers the DAGs individually. The aim was to optimize the overall makespan and to achieve fairness among DAGs. Fairness is achieved if the slowdowns experienced by each DAG due to resource sharing are equal. Fairness is a QoS measurement that results in higher-efficiency runs and has shown similar results in terms of overall makespan.

In [15] the former concept was extended to schedule multiple parallel task DAGs. The aim was again to minimize the overall makespan using constrained resource allocations to obtain fairness among DAGs. The aim of the fairness measurement is to allow shorter jobs and longer jobs equal access to resources. The DAGs are all known beforehand and are considered individually. Three policies based on three DAG metrics were proposed to obtain fairness. We compare the proportional share based on DAG workload (PS-work), which has been called the best in terms of makespan, with our proposed solution for multiple DAG scheduling.

This paper addresses a similar problem, i.e., that of scheduling multiple DAGs simultaneously, but we allow the DAGs to arrive at different instants in time, which results in a more complex problem. The aim is to minimize the overall makespan without considering fairness because the jobs are related in the sense that only the collective result is meaningful, as mentioned previously.

Thus, the scheduling method proposed here has two parts, one related to the scheduling strategy and another to the scheduling algorithm that determines a schedule for a given instant of time. The DAGs are integrated in a master DAG as in [25], and in each scheduling instant the algorithm considers the tasks ready for execution from all jobs and applies DAG scheduling. Therefore, the scheduling problem here can be expressed using a classical formula [10,19,20,22] as summarized below.

A DAG is represented by $G = (V,E)$, where V is the set of v nodes of the graph, representing the tasks to be processed, and E is the set of e edges, representing precedence among tasks and communication costs. For each node v_i, a start time ($ST(v_i)$) and a finish time ($FT(v_i)$) are defined; the scheduling length is given by $max_i\{FT(v_i)\}$. Here, the DAG is the master DAG that represents all active jobs. Therefore, the goal in scheduling a batch of jobs at any scheduling instant is to minimize $max_i\{FT(v_i)\}$. This definition is valid for both homogeneous and heterogeneous machines and for both parallel tasks (executed on several processors) and non-parallel tasks (executed on one processor).

Many scheduling algorithms were developed for the dynamic scheduling of DAGs, or workflows, for GRID systems. In [24] a taxonomy is presented that classifies the GRID schedulers based on different aspects. These scheduler types are not suitable in the context of this paper because their concerns are related to high-level issues, such as minimizing the data movements due to the scheduling choice [16], rather than to the local cluster decisions of which the processors to use to execute each task.

Steady-state scheduling is another approach to scheduling on heterogeneous systems, which aims to maximize the total throughput by determining a periodic task allocation [3,4]. The targets were applications with very large numbers of atomic and independent tasks, and the optimization was the steady-state throughput of each application. Here we are considering a different problem, i.e., the minimization of the processing time of a batch of jobs (applications) represented by DAGs, where each task can run on more than one processor (non-atomic tasks).

3. Computational model

The computational model allows us to estimate the execution time of each task on a heterogeneous distributed memory machine. The processors used in a given task can be arranged in a grid (p,q) that optimizes its execution time. The processors are connected by a switched network that supports simultaneous communications between different pairs of machines.

The communications required to complete a task are included in the computation time as a function of the processors $P = p \times q$ used for that task [2]. The tasks in this work are linear algebra kernels that operate with matrix data, which are common in many engineering problems.

The parameters considered are the processing capacity S_i of the processor i ($i \in [1,P]$) measured in *Mflop/s*, the network latency T_L measured in μs and the bandwidth ω measured in *Mbit/s*. The total computation time is obtained by summing the time T_{comm} spent communicating and the time $T_{parallel}$ spent in parallel operations. The time required to transmit a message of b elements is $T_{comm} = T_L + b\omega^{-1}$. The time required to compute the pure parallel part of the code without any sequential part or synchronization time and with P processors is $T_{parallel} = f(n)/\sum_{i=1}^{P} S_i$. The numerator $f(n)$ is the cost function of the algorithm, measured in floating point operations, which depends on the problem size n. In summary, the computational model considers a non-overlapping model of computation–communication, a one-port constraint for sending and receiving, a homogeneous switched network and heterogeneous processors. Multi-core machines are modeled here as single nodes with a single processing capacity. We suppose that there are no memory constraints, so the tasks have the available memory required to run on whatever processor is selected.

The optimal processor layout can be computed by introducing the Lagrange multipliers as in the computation of the best processor grid for linear algebra kernels in [2].

For example, the tridiagonal reduction algorithm (TRD) on a grid (p,q) must transmit $2n(p-1) + 4n^2(pq-1)$ data elements and has $f(n) = 28n^3$. In our platform, the best layout is $(1,q)$, so that the total processing time is $T_t(n,p,q) = T_L + 4n^2(q-1)\omega^{-1} + \frac{28n^3}{\sum_{i=1}^{q} S_i}$. In Section 4.2, we use this computational model to obtain the optimal processing time and the corresponding processing capacity for each task.

4. Dynamic scheduling method

The scheduling method is divided into two parts: a scheduling strategy and a scheduling algorithm. The scheduling strategy defines the instants when the scheduling algorithm is called to produce a schedule based on the machine information and the tasks waiting in the queue at the time it is called.

4.1. Scheduling strategy

The scheduling strategy determines the type of dynamic scheduling, i.e., immediate or batch mode, and the time when the scheduling algorithm is called to produce a new schedule, which defines a scheduling cycle [9,21]. Here, we consider the batch mode, which consists of rescheduling all ready tasks not yet processed along with the ready tasks of a newly arrived job. The scheduling instants can be related to several scheduler characteristics, e.g., hardware availability changes,

Algorithm 1

```
1. while(1)
2.     Wait for new job OR other event
3.     Insert the job into the master DAG
4.     Call the scheduling algorithm with tasks from all jobs
```

Fig. 1. Batch mode scheduling strategy.

updates to end time of completed tasks if the deviation from the estimate exceeds a specified threshold. These events are expressed as 'other event'. The scheduling strategy is shown in Algorithm 1.

Here, a new scheduling cycle starts only when a new job arrives. Other events, such as changes to processor availability or the completion of a job, can be explored in the future to improve the scheduler. An incoming job is first inserted in a master DAG and connected to the zero time root and exit nodes, so that it has the same priority as any other. At this instant, the scheduler algorithm is called with the master DAG as input, and it reschedules all tasks from all jobs, including the tasks scheduled before but not yet processed.

The rescheduling of all tasks is feasible for low-complexity scheduling algorithms such as the ones used in this paper. In fact, only the tasks ready to start, with no precedence, need to be rescheduled. The information from all tasks is used to compute the critical path, but the scheduling algorithm can be implemented in such a way that it stops after a schedule is obtained for ready tasks, thus reducing the time complexity of each schedule.

Regardless of which scheduling algorithm is used in the second step, no static reservation of processing nodes is made per job. This is an important feature of the dynamic scheduler presented here because, as shown in the results section, it improves the cluster performance in terms of the total schedule length. The difference in the static reservation schedule is that if a job does not need that capacity at a given moment, the capacity will be made available for other jobs.

4.2. Scheduling algorithm

In the second part of the scheduling method, the scheduling algorithm is called to produce a schedule based on the information available at that instant concerning the ready tasks and the machine information. Here, two algorithms for scheduling DAGs on heterogeneous clusters are considered: an adaptation of HEFT [22] for parallel tasks, which we call P-HEFT, and HPTS [1]. These algorithms are distinct from the work presented in [7,5,8,12,23] because there, monotonic tasks were used, i.e., the processing time of a task does not increase if more processors are used. Our task model considers non-monotonic tasks because they are more realistic for our target applications. In this model, there is a number p of processors for which the execution time $t_{i,p}$ of task i is a minimum, i.e., $t_{i,p} < t_{i,p-1}$ and $t_{i,p} < t_{i,p+1}$. This is due to the communications inherent to the parallel task model, similar to the Amdahl law for parallel data applications.

HEFT was proposed for scheduling non-parallel task DAGs on a heterogeneous cluster, and it achieved the minimum makespan [22]. In [14], an adaptation of HEFT for parallel task DAGs to run on a heterogeneous multi-cluster platform of

Algorithm 2

```
1. Set the computation cost of tasks and
     comm. costs of edges with mean values
   Vector s stores processing capacities in decreasing order
2. Compute b-level for all tasks
3. while there are unscheduled tasks
4.     Compute the set of ready tasks
5.     For each ready task i
6.         Compute the optimal capacity S_i^*
7.     while ready tasks ≠ ∅
8.         i = task with maximum b-level
9.         tl_i = t-level of task i
10.        k = fastest proc. free at tl_i OR first proc. free after tl_i
           est'_i = est_i = max(tl_i, AvailabilityTime_k)
11.        p = 1 , S_i = 0
12.        while  S_i < S_i^* AND S_i < limit
13.            if p = 1 OR est_i + t_{i,p} < est'_i + t_{i,p-1}
                   v(p) = k      #add processor
                   est'_i = est_i , p = p + 1 , S_i = S_i + s(k)
14.            k = next fastest proc. free at tl_i OR
                       next proc. free after tl_i
15.            if k = null break     #all processors analyzed
16.            est_i = max(tl_i, AvailabilityTime_k)
```

Fig. 2. P-HEFT scheduling algorithm.

homogeneous clusters, called M-HEFT, was evaluated and compared to other scheduling algorithms. The conclusion was that M-HEFT achieved the minimum makespan, particularly if it limited a maximum of 50% of the processors to any given task (M-HEFT-MAX50). However, it only achieved low rates of efficiency, below 20%. Because the aim of this paper is to minimize the makespan independently of the efficiency reached, we proposed an adaptation of HEFT for our target machine, i.e., a single heterogeneous cluster.

The scheduling algorithms applied to the global DAG are based on the list scheduling technique [10] with the following steps: (a) determine the available tasks to schedule, (b) assign a priorities to them and (c) until all tasks are scheduled, select the task with the highest priority and assign it to the processor that allows the earliest start time. For parallel tasks, the last step selects not one processor but several processors that allow the earliest start times [1]. Note that in this step, we refer to the tasks that result from all jobs.

Two attributes frequently used to define the tasks priorities are the *t-level* (top-level) and the *b-level* (bottom-level). The *t-level* of a node n_i is defined as the length of the longest path from an entry node to n_i (excluding n_i). The *b-level* of a node n_i is the length of the longest path from n_i to an exit node. The nodes of the DAG with higher *b-level* values belong to the critical path.

The HEFT algorithm is a heuristic that schedules tasks by decreasing *b-level* values and assigns them to the processor that allows the earliest finish time (EFT). The adaptation for parallel task DAGs, P-HEFT as shown in Algorithm 2, initially also computes the average computation and communication costs, and it computes *b-level* once in step 2.

From lines 4 to 6, the algorithm determines the computational capacity that minimizes each ready task *i*, S_i^*, according to the computational model assuming that the fastest processors are used. The number of processors is not important here, and it is not registered. From steps 7 to 16, the algorithm schedules the ready tasks in decreasing order of *b-level* and assigns processors that can start at the earliest start time. At step 10, the fastest processor available at *t-level* of task *i* (tl_i) is selected or, in case there is no processor currently available, the first processor available after tl_i is selected. The last condition is required to allow for the case when a later start time with more processors results in an earlier finish time. In this case, some holes can appear in the schedule that are not filled by the algorithm. The value est_i is the earliest start time of task *i*; it is the

Algorithm 3

1. Vector s stores processing capacities in decreasing order
2. $S_{max} = \sum_{j=1}^{P} s(j)$ (machine with P processors)
3. while set of tasks $U \neq \oslash$
4. Compute the set of ready tasks
5. For each ready task i
6. Compute the optimal capacity S_i^*
7. if $\sum_i S_i^* > S_{max}$
   For each ready task i
   $S_i' = (S_{max} / \sum_j S_j^*) S_i^*$
8. else
   For each ready task i, $S_i' = S_i^*$
9. Compute actual t-level for ready tasks
10. while ready tasks $\neq \oslash$
11. i =task with minimum t-level, tl_i = t-level of i
12. k = fastest proc. free at tl_i OR first proc. free after tl_i
    $est_i' = est_i = max(tl_i, AvailabilityTime_k)$, $p = 1$, $S_i = 0$
13. while $S_i < S_i'$
14. if $p = 1$ OR $est_i + t_{i,p} < est_i' + t_{i,p-1}$ #add processor
    $v(p) = k$, $est_i' = est_i$, $p = p + 1$, $S_i = S_i + s(k)$
15. k = next fastest proc. free at tl_i OR
    next proc. free after tl_i
16. if $k = null$ $break$ #all processors analyzed
17. $est_i = max(tl_i, AvailabilityTime_k)$
18. Compute b-level of ready tasks, $w = 1$
19. while true
20. $hbl(w)$ = task with highest b-level
21. r = task with minimum b-level
22. if r exists in hbl $break$
23. Remove fastest processor assigned to task r
24. Assign it to task $hbl(w)$
25. Re-evaluate proc. time of r and $hbl(w)$
26. Re-evaluate b-level of r and $hbl(w)$
27. $w = w + 1$

Fig. 3. HPTS scheduling algorithm.

maximum of tl_i and the finish time of the last task assigned to the selected processor k ($AvailabilityTime_k$). At step 12, the parameter *limit* is introduced to limit the amount of capacity assigned to a single task, as in [14]. When *limit* is less than 100%, we refer to the algorithm as P-HEFT-*limit*. At step 13, $t_{i,p}$ is the processing time obtained with the p processors selected thus far. If the actual finishing time is less than that with the former solution, the algorithm stores the processor k in the vector v that at the end will store the processors assigned to the task. At step 15, the assignment for task i ends if there are no more processors to evaluate.

The processing capacity required to achieve the optimal processing time $t^*_{i,p}$, for task i and considering the p fastest processors, is defined as $S^*_i = \sum_{j=1}^{p} s(j)$. It is obvious that if slower processors are used, because not all concurrent tasks can be executed on the fastest processors, the capacity that achieves the minimum time for task i is less than S^*_i. Because more processors are required to obtain S^*_i, more communications and consequently more processing time would result; therefore, the minimum processing time achievable will be certainly higher than the estimated $t^*_{i,p}$.

In HPTS algorithm, shown in Fig. 3, the while cycle at step 3 refers to all tasks of the Master DAG. In step 4, the ready tasks are those whose predecessors have finished processing. From step 5 to 8, the algorithm determines the computational capacity that minimizes each ready task S^*_i according to the computational model and by assuming that the fastest processors are used. Again, the number of processors is not important here, and it is not registered. If the machine capacity S_{max} is exceeded, the capacity assigned to each task is limited to the relative weight of each task. From steps 9 to 17, the algorithm schedules all ready tasks and assigns them to the predetermined processing capacity. On step 12, the fastest processor available at *t-level* (tl_i) is selected for task i, but if there are no processors available to start at tl_i, the first processor available after tl_i is selected, as in Algorithm 2. From steps 18 to 27, the algorithm tries to correct the last schedule by assigning more processors to the tasks that have a higher *b-level*. The computation of the *b-level* on step 18 and 26 uses $t^*_{i,p}$ for the tasks on upcoming levels (not processed yet) and the time computed on steps 14 and 25 for tasks of the present level. The algorithm halts if the task with minimum *b-level* is in the highest b-level set (*hbl*). At step 25, the computation times of tasks r and $hbl(w)$ are re-evaluated considering the new set of processors made available by the transference of the fastest processor from the set assigned to task r to the set assigned to task $hbl(w)$. The time complexity of P-HEFT is the same as that of HEFT; therefore, Algorithms 2 and 3 are $O(u^2 \times P)$ [1,22] for a set U with u tasks and a machine with P processors.

5. Results and discussion

In this section, we evaluate and compare the performance of the scheduling algorithms P-HEFT and HPTS when scheduling a single parallel task DAG using an extensive simulation setup. Then, the simulation results of the dynamic scheduling method are presented and compared to the alternative of resource reservation per job [15]. Finally, we present the results of a real problem in a real system.

The metrics used for comparison are the schedule length (makespan) and the efficiency. The makespan is the finish time of the last task and is the most relevant metric in this work because the aim is to minimize it. Another important metric when comparing parallel implementations is the efficiency. In a heterogeneous cluster, the efficiency does not consider the number of processors used in the computation; instead it considers the power used. For a heterogeneous environment, the number of processors is ambiguous information. The efficiency is then computed as in [14], i.e., as the ratio of the total work in the sequential application execution on the fastest processor to the total work of the parallel application. The total work is computed as the sum of the products of the processing times used on each processor scaled by the ratio of the processor speed to the fastest one. Higher values of efficiency indicate better resource utilization.

5.1. Parallel machine

The clusters used in the simulations are generated with 10, 20, 30, 40 and 50 processors, and we assume a 100 Mb/s switched network and a latency of 50 μs. The main characteristic of the network is that it allows simultaneous communications between different pairs of machines. For parallel tasks, this is an important characteristic because to complete a task, the processors (group) involved have to exchange data. In general, when taking into account the amount of communication involved, we need to ensure that inside the group there are no communication conflicts. It would be very difficult to synchronize communications in different parallel tasks if there were conflicts to be avoided.

The processor speeds for a given cluster are given by a uniform probability distribution. The fastest processor is fixed to the value of 400 Mflop/s, and the minimum is defined by the heterogeneity of the cluster. We considered 5 values for the heterogeneity: 0, 0.2, 0.5, 1 and 2. For example, a heterogeneity of 1 will generate speeds from 200 Mflop/s to 400 Mflop/s uniformly. Although the dynamic algorithm was designed to work on heterogeneous machines, homogeneous cases will also be considered.

5.2. DAGs and tasks

The DAGs used in the simulation setup were randomly generated using the program in [17]. This DAG generator was also used in other related works, such as [7,14,15]. There are four parameters that define the DAG shape: width, regularity, density and jumps. The width is the number of tasks on the largest level. A small value results in a chain DAG. The regularity is

related to the uniformity of the number of tasks in each level. The density is related to the number of edges between two levels of the DAG. These parameters may vary between 0 and 1. The jump parameter indicates that an edge can go from level l to level $l + jump$. A jump of 1 is an ordinary connection between two consecutive levels.

In this paper, we consider applications with 10, 20, 30 and 50 tasks. The tasks constituting the DAGs are linear algebra kernels, i.e., tridiagonal factorization (TRD), matrix Q computation, QR iteration and correlation (C). The size of each task is randomly generated in the interval [200, 1200]. We recall that the tasks are non-monotonic, and to estimate the processing time of each task, the computational model described in Section 3 is used. Table 1 shows the intra-task communication and computational weights of the tasks, as well as the best processor layout. To each of the DAG edges is assigned an inter-task communication cost proportional to the size of the output data of the predecessor tasks, which is related to data redistribution [2]. All data are assumed to be double values of 8 bytes.

5.3. Single DAG scheduling

Here we present the results of the P-HEFT algorithm proposed in this paper and compare them to those of the HPTS algorithm [1]. The DAGs generated use the values of 0.1, 0.2 and 0.8 for the width, 0.2 and 0.8 for the regularity and density, and 1, 2, and 4 for the jump parameter. For each combination of these parameters, we generate 3 DAGs, giving $3^3 \times 2^2 = 108$ combinations. This number of DAGs is used for each DAG size of 10, 20, 30 and 50 tasks, giving $108 \times 4 = 432$ combinations. For the cluster, we consider 5 types of heterogeneity, i.e., 0, 0.2, 0.5, 1 and 2, with 5 different dimensions of 10, 20, 30, 40 and 50 processors, and we generate 5 machines for each combination, giving $5^3 = 125$ different clusters. Because we run each DAG in each cluster, we have $432 \times 125 = 54{,}000$ runs. Fig. 4 shows the average results, per machine size, obtained for the makespan and efficiency.

The label P-HEFT refers to Algorithm 2, and P-HEFT-50 denotes the results from P-HEFT when we limit the machine capacity to a maximum of 50% for each task. In fact, this simple limitation results in improved performance, as noted in [14]. The difference is not relevant above 30 processors and is zero above 40 because for these machine sizes no task needs more than 50% of the machine, so both give the same solution. On average, HPTS requires 4.9% more processing time. On the other hand, HPTS is, on average, 11% more efficient than P-HEFT-50, which means that HPTS has better resource usage. This is an important characteristic when scheduling multiple DAGs, as shown below.

The results reported for M-HEFT [14], an adaptation of HEFT for parallel task scheduling on a heterogeneous multi-cluster platform of homogeneous clusters, proved to be the best for the makespan, but with a significant decrease in efficiency. One important result of this paper is that the parallel task version of HEFT proposed here produces the best makespan for a single DAG without significantly penalizing the efficiency.

Table 1
Task communication and computational weight and corresponding best processor layout; here, for QR, the convergence rate γ is considered to be 1.

Task type	Comm. weight	Comp. weight ($f(n)$)	$Grid(p,q)$
TRD	$4n^2(q-1)$	$28n^3$	$(1,q)$
Q	$0.5n^2(q-1)$	$23n^3$	$(1,q)$
QR	$\gamma n^2(p-1)$	$56n^3$	$(p,1)$
C	$2n^2(q-1)$	$84n^3$	$(1,q)$

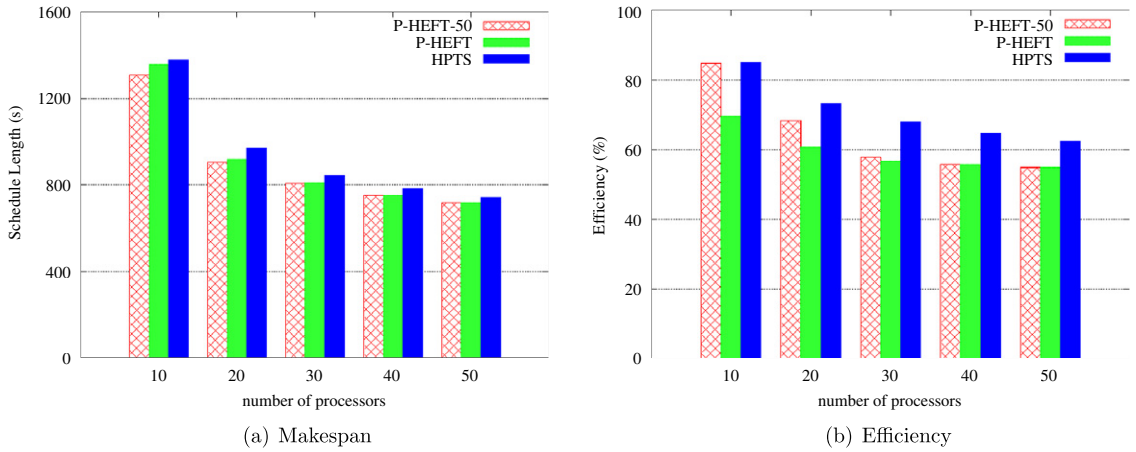

Fig. 4. Schedule length (makespan) and efficiency obtained for single DAG scheduling with P-HEFT and HPTS; for all machine sizes DAGs with 10, 20, 30 and 50 tasks were used.

5.4. Dynamic scheduling of multiple DAGs

Here we present the results of the scheduling method for the dynamic case where jobs arrive at different times. The three scheduling algorithms were tested for the same set of machine configurations and for the same set of 12 jobs. To generate random arrival, we insert a new job when 40% of the estimated optimal execution time (obtained when the job runs exclusively on the machine) of the last job has elapsed. For the first job, this will be true because it runs exclusively; all other jobs will take more time to complete than the optimal time estimated, but we still insert a new job after 40% of the optimal time of the last job has elapsed. Depending on the job length, there could be many concurrent jobs.

We used 3 sets of 12 jobs each. Each set was made up of DAGs with 10, 20 and 30 tasks. The 12 DAGs were selected from the set of 108 used in Section 5.3 so that the full range of values for width, regularity and density were represented. For the jump parameter, only the value 1 was used.

As before, we used 5 levels of heterogeneity, i.e., 0, 0.2, 0.5, 1 and 2, with 5 different dimensions of 10, 20, 30, 40 and 50 processors. Five machines for each combination were generated giving $5^3 = 125$ different clusters. For each of the 125 clusters, we ran the 3 sets of 12 DAGs, giving $3 \times 125 = 375$ runs. Fig. 5 shows the average results, for each machine size, for the makespan and efficiency.

The results for P-HEFT-50 and P-HEFT maintain the same relationship as for single DAG scheduling, i.e., P-HEFT-50 achieved a lower makespan but higher efficiency. This result is particularly visible for smaller and therefore saturated machines. For larger machines, where there are sufficient processors that no task will require more than 50% of the machine power, P-HEFT-50 is equivalent to P-HEFT.

For multiple DAG scheduling, Fig. 5 shows that HPTS has better performance than the other two algorithms. This was expected because HPTS has higher efficiency levels, which is favorable when the machine has higher load levels. In fact, this is consistent with the improvement achieved by P-HEFT-50 over P-HEFT, which is only due to the limitation of 50% of the machine capacity for a single task. The HPTS algorithm also implements a limitation of resources per task at steps 5 to 7 (Algorithm 3), but instead of a blind limitation of 50%, the limitation is proportional to the dimension of the tasks ready to be scheduled.

5.5. Resource reservation versus non-reservation

Here we present results of the comparison of the scheduling method proposed in this paper, corresponding to the non-resource reservation policy, with the alternative of resource reservation [15]. It is important to mention that the aim in [15] was different from our aim, which was to minimize the completion time of a batch of jobs. The comparison is justified to show that, for our aim, the method proposed here achieved better performance than the available alternative. We considered the HPTS algorithm for scheduling.

The method in [15] is static and all DAGs are available at time zero; therefore we applied our dynamic method also assuming this condition. In this case, only the non-reservation policy of our method that does not assign a fixed set of processors to a job will be different from the method of [15]. Results for the dynamic case, as described in Section 5.4, are also presented, similar to [9], which compared the dynamic method to the static methods available. In this case, the arrival times of all tasks are known a priori for the static method.

Several alternatives are presented in [15] that balance the fairness of resources among jobs against the average makespan. Because our interest is primarily in minimizing the makespan, we selected the version PS-work (proportional share) that shares resources proportionally to the work of each job and achieves the shortest average makespan.

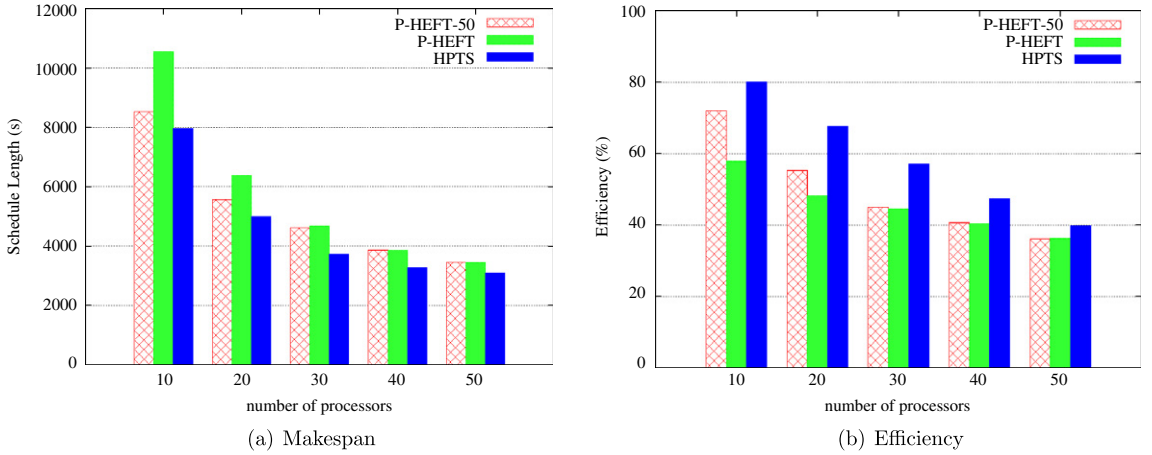

Fig. 5. Schedule length (makespan) and efficiency obtained for dynamic multiple DAG scheduling with P-HEFT and HPTS.

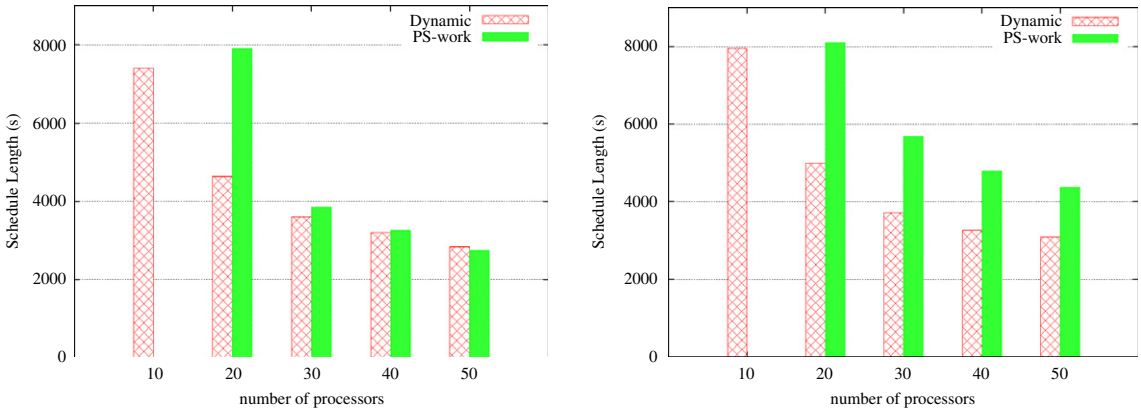

Fig. 6. Resource reservation versus non-reservation; schedule length (makespan) for (a) all jobs arriving at zero time and (b) jobs arriving at different instants.

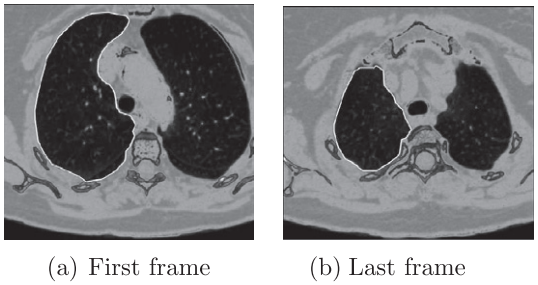

Fig. 7. First and last frames of an example image sequence from computed tomography (CT).

Fig. 6 shows the results for the same 3 sets of 12 DAGs and conditions described in the former Section 5.4. When all jobs start at time zero, there is a significant difference when the machine is heavily loaded, corresponding to up to 20 processors in this case. With 50 processors, the PS-Work version achieves better results as expected when more processors are available; also in these cases, because there are enough processors to use, the disadvantage of static reservation starts to become irrelevant. The results with 10 processors for PS-Work were not computed because the method assumed that there were more processors than jobs, which in fact was not true. For the dynamic case, Fig. 6(b), where tasks arrive at different times, shows that the dynamic algorithm achieves better results as expected.

5.6. Experimental results

In addition to the randomly generated DAGs here, we also present results from a real application, i.e., image registration for 3D object reconstruction, executed on a real cluster. The method consists of defining the target object by its border in each frame, as illustrated in Fig. 7, and between each two frames, the deformable object tracking algorithm [18] is computed to find the corresponding points in two consecutive frames. Fig. 7 shows the first and last frames of a sequence of 21 images taken from a CT scan of a human trunk. The object segmented is the left lung; we can see that the shape varies significantly along frames.

The deformable object tracking algorithm is based on finite element analysis, and it requires the computation of the eigenvectors of symmetric matrices. The aim is to obtain correspondences between the object points of images i and $i+n$. The algorithm is divided into eigenvector computation and matrix correlation. The eigenvector computation is subdivided into three operations: tridiagonalization, orthogonal matrix computation (matrix Q) and QR iteration. A job is created to represent the computation between each two consecutive frames. Fig. 8 shows the DAG and the tasks involved in a single job. Note that the DAG considered here does not include the task of contour definition because the algorithm is problem-dependent.

The proposed dynamic scheduling method is compared to the static processor allocation, i.e., PS-Work [14], when all jobs start at time zero. The cluster has 10 processors with the following distribution of capacities, in Gflop/s: 3 with 0.4, 4 with 0.3 and 3 with 0.2. To reduce the number of variables influencing the result, the number of points per object is 300 in all frames, so the matrices consist of 300 by 300 elements. The scheduling algorithm used is the HPTS.

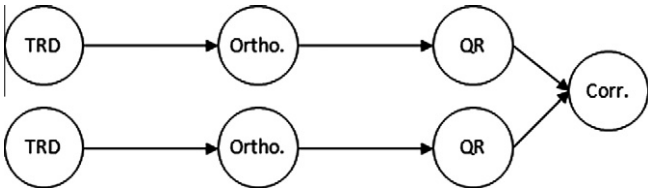

Fig. 8. DAG structure of the deformable object tracking algorithm; TRD: tridiagonalisation; Ortho: orthogonal matrix computation; Corr: correlation matrix computation.

Table 2
Performance results for the deformable object tracking algorithm.

Number of DAGs	t_{static}(s)	$t_{dynamic}$(s)	Improvement (%)
4	9.12	10.16	−11.4
5	12.07	12.47	−3.3
6	14.11	12.47	11.6
10	28.51	18.73	34.3

The experiment consists of executing 4, 5, 6 and 10 jobs simultaneously as described in Table 2.

These results are similar to the ones presented in Fig. 6(a), where all DAGs start at time zero. For 5 DAGs, the static approach can divide the cluster uniformly by the jobs with 2 machines each, totaling 0.6 Gflop/s and resulting in better performance. This was also observed in the simulation for more than 50 processors. For 4 jobs, there are enough processors per job, and the lower overhead is favorable for the static approach. When the machine is heavily loaded, the static reservation degrades the performance, and the improvement of the dynamic algorithm can be very high, in this case 34.3%. This behavior was also observed in Fig. 6(a) for 20 processors. Note that our target is not satisfied when all jobs start at time zero, and therefore, the static algorithm is not viable to solve the scheduling problem as required.

6. Conclusions

In this paper, we have proposed a dynamic method for scheduling a batch of mixed-parallel DAGs on heterogeneous systems that arrive at different times. It improves the response time by allowing variation in the computing power (number of processors) assigned to a job. It showed better performance than the resource reservation alternative for a wide set of machines, both homogeneous and heterogeneous, both for random generated DAGs and for a real world application.

To the best of our knowledge, ours is the first method to schedule multiple mixed-parallel jobs dynamically on heterogeneous clusters. A new version of HEFT for parallel task scheduling, called P-HEFT, was also proposed. This algorithm was the best for the makespan of single DAG scheduling, although it had lower efficiency. For multiple DAG scheduling, the efficiency of the scheduling algorithms reflects the capacity to share resources and, as it is the most efficient HPTS, it obtained the best makespan for multiple DAG scheduling.

The dynamic method proposed does not require a fixed subdivision of processors. When scheduling a set of ready tasks, the machine is viewed as a whole, independently of the groups of processors formed in the last level, thus allowing better use of the machine and consequently achieving improvements in processing time when the machine is more loaded.

The aim, as defined before, is to minimize the completion time of a batch of jobs, which corresponds to many real-world applications such as video surveillance, microscopic imaging processing, biomechanical analysis and image registration. Future research will extend the scheduling method for the optimization of a performance measurement related to the quality of service for a multi-user environment, where the objective function is the individual QoS measured by users rather than the global processing.

References

[1] J. Barbosa, C. Morais, R. Nobrega, A.P. Monteiro, Static scheduling of dependent parallel tasks on heterogeneous clusters, in: Heteropar'05, IEEE Computer Society, 2005, pp. 1–8.
[2] J. Barbosa, J. Tavares, A.J. Padilha, Linear algebra algorithms in a heterogeneous cluster of personal computers, in: Proceedings of 9th Heterogeneous Computing Workshop, IEEE CS Press, 2000, pp. 147–159.
[3] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, Y. Robert, Centralized versus distributed schedulers for bag-of-tasks applications, IEEE Transactions on Parallel and Distributed Systems 19 (5) (2008) 698–709.
[4] O. Beaumont, A. Legrand, L. Marchal, Y. Robert, Steady-state scheduling on heterogeneous clusters: why and how?, in: 18th International Parallel and Distributed Processing Symposium, IEEE Computer Society, 2004
[5] J. Blazewicz, M. Machowiak, J. Weglarz, M. Kovalyov, D. Trystram, Scheduling malleable tasks on parallel processors to minimize the makespan, Annals of Operations Research (129) (2004) 65–80.
[6] S. Chakrabarti, J. Demmel, K. Yelick, Modeling the benefits of mixed data and task parallelism, in: Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, 1995, pp. 74–83.

[7] P-F. Dutot, T. NTakpé, F. Suter, H. Casanova, Scheduling parallel task graphs on (almost) homogeneous multicluster platforms, IEEE Transactions on Parallel and Distributed Systems 20 (7) (2009) 940–952.
[8] K. Jansen, Scheduling malleable parallel tasks: an asymptotic fully polynomial time approximation scheme, Algorithmica 39 (2004) 59–81.
[9] J.-K. Kima, S. Shivleb, H.J. Siegel, A.A. Maciejewski, T.D. Braun, M. Schneider, S. Tidemanc, R. Chittac, R.B. Dilmaghani, R. Joshi, A. Kaul, A. Sharmab, S. Sripada, P. Vangari, S.S. Yellampalli, Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment, Journal of Parallel and Distributed Computing 67 (2007) 154–169.
[10] Y. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Computing Surveys 31 (4) (1999) 406–471.
[11] Y. Kwok, I. Ahmad, On multiprocessor task scheduling using efficient state space search approaches, Journal of Parallel and Distributed Computing 65 (2005) 1515–1532.
[12] R. Lepère, G. Mounié, D. Trystram, An approximation algorithm for scheduling trees of malleable tasks, European Journal of Operational Research (142) (2002) 242–249.
[13] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamically mapping of a class of independent tasks onto heterogeneous computing systems, Journal of Parallel and Distributed Computing 59 (1999) 107–131.
[14] T. N'Takpé, F. Suter, A comparison of scheduling approaches for mixed-parallel applications on heterogeneous platforms, in: 6th International Symposium on Parallel and Distributed Computing, IEEE Computer Society, 2007, pp. 1–8.
[15] T. N'Takpé, F. Suter, Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations, in: International Symposium on Parallel and Distributed Processing, IEEE Computer Society, 2009, pp. 1–8.
[16] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski, K. Kennedy, Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction, in: ACM/IEEE Conference on Supercomputing, ACM, 2006, pp. 1–10.
[17] DAG Generation Program, 2010. Available from: <http://www.loria.fr/~suter/dags.html>.
[18] S. Sclaroff, A. Pentland, Modal matching for correspondence and recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (6) (1995) 545–561.
[19] B. Shirazi, M. Wang, G. Pathak, Analysis and evaluation of heuristic methods for static task scheduling, Journal of Parallel and Distributing Computing 10 (1990) 222–232.
[20] O. Sinnen, L. Sousa, List scheduling: extension for contention awareness and evaluation of node priorities for heterogeneous cluster architectures, Parallel Computing (30) (2004) 81–101.
[21] W. Sun, Y. Zhang, Y. Inoguchi, Dynamic task flow scheduling for heterogeneous distributed computing: algorithm and strategy, IEICE Transactions on Information and Systems E90-D (4) (2007) 736–744.
[22] H. Topcuoglu, S. Hariri, M.-Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel and Distributed Systems 13 (3) (2002) 260–274.
[23] Denis Trystram, Scheduling parallel applications using malleable tasks on clusters, in: 15th International Conference on Parallel and Distributed Processing Symposium, 2001.
[24] J. Yu, R. Buyya, A taxonomy of scientific workflow systems for grid computing, ACM SIGMOD Record 34 (3) (2005) 44–49.
[25] H. Zhao, R. Sakellariou, Scheduling multiple dags onto heterogeneous systems, in: International Symposium on Parallel and Distributed Processing, IEEE Computer Society, 2006, pp. 1–14.