Task Scheduling: Considering the Processor Involvement in Communication

Oliver Sinnen and Leonel Sousa
INESC-ID, Rua Alves Redol 9
1000-029 Lisboa, Portugal
{oliver.sinnen, las}@inesc-id.pt

Abstract—Classical task scheduling employs a very simplified model of the target parallel system. Experiments demonstrated that this leads to inaccurate and inefficient schedules. Contention aware scheduling heuristics take the contention for communication resources into account, which improves the schedules significantly. Yet, one aspect remains to be investigated: the involvement of the processors in communication. This paper proposes a new scheduling model, called involvement-contention model, that integrates the consideration for the processor involvement into task scheduling. A list scheduling based heuristic is proposed for the new model, which produces significantly more accurate and efficient schedules in experiments on real parallel systems.

Keywords: parallel programming, task scheduling, system models, processor involvement, contention

I. INTRODUCTION

One challenging aspect of parallel programming is the spacial and temporal assignment of the tasks of a program to the processors of the target system, referred to as task scheduling. In its general form task scheduling is an NP-hard problem [18], which has motivated the investigation of many heuristics for its near optimal solution.

Most of these heuristics [2], [5], [6], [10], [12], [19] are based on a very idealised system model that supposes a fully connected and contention free communication network and a dedicated subsystem that performs all interprocessor communications. Intuition and experiments demonstrated that this model results in inaccurate and inefficient schedules [11], [14].

Various approaches to include contention awareness into task scheduling have been undertaken [3], [8], [13]. In [15] a contention aware task scheduling approach was proposed that integrates the various aspects of contention, such as endpoint and network contention as well as network heterogeneity. Experimental results verified the improvements of the new contention model [15], even though the results do not completely satisfy.

One aspect not considered in the classic and neither in the contention model is the involvement of the processor in interprocessor communication. In many parallel systems, a communicating processor participates in the communication, for example by copying data to and from a shared buffer.

This paper investigates the involvement of the processors in communication and its integration into task scheduling. A new system model for scheduling that considers the involvement of the processor in communication is proposed based on the contention model. As a result, the new model is general and unifies the existing scheduling models. Since scheduling under the new model requires the adjustment of the existing techniques, it is shown how this is done with the proposal of a list scheduling heuristic. Experimental results performed on real parallel systems demonstrate the significantly improved accuracy and efficiency of the schedules produced under the new involvement-contention model.

The rest of this paper is organised as follows. Section II establishes the background and definitions of task scheduling under the classic and the contention model. Section III analyses in general the processor involvement in communication. Based on this analysis, Section IV investigates the integration of the awareness for the processor involvement in task scheduling. Section V discusses scheduling heuristics based on the new model. Experimental results are presented in Section VI and this paper concludes with Section VII.

II. TASK SCHEDULING

In task scheduling a program $\mathcal{P}$ to be scheduled is represented by a directed acyclic graph (DAG) $G = (V, E, w, c)$. The nodes in $V$ represent the tasks of $\mathcal{P}$ and the edges in $E$ the communications between the tasks. An edge $e_{ij} \in E$ represents the communication from node $n_i$ to node $n_j$. The positive weight $w(n)$ associated with node $n \in V$ represents its computation cost and the non-negative weight $c(e_{ij})$ associated with edge $e_{ij} \in E$ represents its communication cost.

The nodes are strict with respect to both their inputs and their outputs: that is, a node cannot begin execution until all its inputs have arrived, and no output is available until the computation has finished and at that time all outputs are available for communication simultaneously. The set $\{n_x \in V : e_{xi} \in E\}$ of all direct predecessors of $n_i$ is denoted by $\text{pred}(n_i)$.

A schedule of a DAG is the association of a start time and a processor with every node of the DAG. To describe a schedule $S$ of a DAG $G = (V, E, w, c)$ on a target system consisting of a set of dedicated processors $P$ the following terms are defined: $t_s(n, P)$ denotes the start time and $\omega(n, P)$ the execution time of node $n \in V$ on processor $P \in P$. Thus, the node’s finish time is given by $t_f(n, P) = t_s(n, P) + \omega(n, P)$. In a homogeneous system the execution time is equivalent to the computation cost of the node, thus $\omega(n, P) = w(n)$. In a heterogeneous system the computation cost $w(n)$ of
node \( n \) describes its average computation cost. The processor to which \( n \) is allocated is denoted by \( \text{proc}(n) \). Further, let \( t_f(P) = \max_{n \in V} \text{proc}(n) = P \{ t_f(n) \} \) be the processor finish time of \( P \) and let \( s(l(S)) = \max_{n \in V} \{ t_f(n) \} \) be the schedule length (or makespan) of \( S \), assuming \( \min_{n \in V} \{ t_s(n) \} = 0 \).

For a such defined schedule to be feasible, the following two conditions must be fulfilled for all nodes in \( G \).

**Condition 1 (Processor Constraint (dedicated processor)):**

For any two nodes \( n_i, n_j \in V \)

\[
\text{proc}(n_i) = \text{proc}(n_j) = P \Rightarrow \left\{ \begin{array}{l}
t_f(n_i, P) \leq t_s(n_j, P) \\
\text{or } t_f(n_j, P) \leq t_s(n_i, P)
\end{array} \right.
\]  

(1)

**Condition 2 (Precedence Constraint (node strictness)):**

For \( n_i, n_j \in V, e_{ij} \in E \),

\[
t_s(n_j, P) \geq t_f(e_{ij}),
\]

(2)

where \( t_f(e_{ij}) \) is the edge finish time of the communication associated with \( e_{ij} \).

The earliest time a node \( n_j \in V \) can start execution on processor \( P \in P \), which is constrained by \( n_j \)'s entering edges (2), is called the Data Ready Time (DRT)

\[
t_{dr}(n_j, P) = \max_{e_{ij} \in E, n_i \in \text{pred}(n_j)} \{ t_f(e_{ij}) \}
\]

(3)

and hence

\[
t_s(n, P) \geq t_{dr}(n, P)
\]

(4)

for all \( n \in V \). If \( \text{pred}(n) = \emptyset \), \( t_{dr}(n) = t_{dr}(n, P) = 0 \), for all \( P \in P \).

**A. Classic scheduling**

Most scheduling algorithms employ a strongly idealised model of the target parallel system [2], [5], [6], [10], [12], [19].

**Definition 1 (Classic System Model):** A parallel system \( M_{\text{classic}} = (P, \omega) \) consists of a finite set of dedicated processors \( P \) connected by a communication network. The processor heterogeneity, in terms of processing speed, is described by the execution time function \( \omega \). This dedicated system has the following properties: i) local communication has zero costs; ii) communication is completely performed by a communication subsystem; iii) communication can be performed concurrently; iv) the communication network is fully connected.

Based on this system model, the edge finish time only depends on the finish time of the origin node and the communication time.

**Definition 2 (Edge Finish Time):** The edge finish time of \( e_{ij} \in E \) is given by

\[
t_f(e_{ij}) = t_f(n_i) + \left\{ \begin{array}{l}
0 \quad \text{if } \text{proc}(n_i) = \text{proc}(n_j) \\
c(e_{ij}) \quad \text{otherwise}
\end{array} \right.
\]

(5)

Thus, communication can overlap with the computation of other nodes, an unlimited number of communications can be performed at the same time and communication has the same cost \( c(e_{ij}) \), indifferent of the origin and the destination processor, unless communication is local.

**B. Contention aware scheduling**

The classic scheduling model (Definition 1) does not consider any kind of contention for communication resources. To make task scheduling contention aware, and thereby more realistic, the communication network is modelled by a graph, where processors are represented by vertices and the edges reflect the communication links. The awareness for contention is achieved by edge scheduling [13], i.e. the scheduling of the edges of the DAG onto the links of the network graph, likewise the nodes are scheduled on the processors.

Due to space limitations, the details of contention aware scheduling cannot be discussed here, the interested reader should refer to [15]. Here, it suffices to define a simplified topology network graph \( TG = (P, L) \), where \( P \) is a set of vertices representing the processors and \( L \) is a set of edges representing the communication links. The system model is then defined as follows.

**Definition 3 (Target Parallel System – Contention Model):**

A target parallel system \( M_{TG} = (TG, \omega) \) consists of a set of possibly heterogenous processors \( P \) connected by the communication network \( TG = (P, L) \). This dedicated system has the following properties: i) local communication has zero costs; ii) communication is completely performed by a communication subsystem.

The loss of the properties of concurrent communication and a fully connected network in comparison with the classic model (Definition 1) is substituted by the scheduling of the edges \( E \) on the communication links \( L \). Corresponding to the scheduling of the nodes, \( t_s(e, L) \) and \( t_f(e, L) \) denote the start and finish time of edge \( e \in E \) on link \( L \in L \), respectively.

When a communication, represented by the edge \( e \), is performed between two distinct processors \( P_{src} \) and \( P_{dst} \), the routing algorithm of \( TG \) returns a route from \( P_{src} \) to \( P_{dst} \): \( R = \langle L_1, L_2, \ldots, L_l \rangle \). The edge \( e \) is scheduled on each link of the route.

The entire edge scheduling only affects the node scheduling with an altered definition of the edge finish time (Definition 2).

**Definition 4 (Edge Finish Time – Contention Model):** Let \( R = \langle L_1, L_2, \ldots, L_l \rangle \) be the route for the communication of \( e_{ij} \in E, n_i, n_j \in V \), if \( \text{proc}(n_i) \neq \text{proc}(n_j) \). The finish time of \( e_{ij} \) is

\[
t_f(e_{ij}) = \left\{ \begin{array}{l}
t_f(n_i) \quad \text{if } \text{proc}(n_i) = \text{proc}(n_j) \\
t_f(e_{ij}, L_1) \quad \text{otherwise}
\end{array} \right.
\]

(6)

Thus, the edge finish time \( t_f(e_{ij}) \) is now the finish time of \( e_{ij} \) on the last link of the route, \( L_l \), unless the communication is local.

**III. PROCESSOR INVOLVEMENT IN COMMUNICATION**

Experimental results demonstrated that the utilisation of the contention aware model in scheduling heuristics significantly improves the accuracy and efficiency of the produced schedules [14]. Yet, the experiments also showed that the contention model is still not realistic enough what regards communication [16].

The contention model (Definition 3) supposes, likewise the classic model (Definition 1), a dedicated communication subsystem to be present in the target system. With the assumed
subsystem, computation can overlap with communication, because the processor is not involved in communication. However, many parallel systems do not possess such a subsystem [4]. Therefore, in many systems the processors are involved, in one way or the other, in interprocessor communication, which serialises communication and prevents the assumed overlap of computation and communication.

A. Processor involvement types

Regarding the involvement of the processor, interprocessor communication can be divided into three basic types: two sided, one sided and third party, as illustrated in Figure 1.

In two sided communication both processors are involved in the communication (send and received). For one sided communication only one of the two processors participates and in third party communication a special subsystem exists that performs the communication. Examples for each of these interprocessor communication types are found among existing parallel machines, for example MPI-TCP/IP based PC clusters (two sided), Cray T3E (one sided), IBM SP-2 (third party).

It is important to note that the software layer employed in parallel programming significantly determines the type of communication. For example, in a shared memory system, communication can be one sided, but the software layer might use a common buffer (one processor writes, the other reads) which turns it into two sided communication (e.g. with MPI [7]).

Fortunately, the description of the processor involvement can be generalised using the notions of overhead and involvement. With these notions an explicit distinction between the different types becomes obsolete.

IV. INVOLVEMENT SCHEDULING

The notions of overhead and involvement are used to enhance task scheduling towards the awareness of processor involvement in communication. In the first step, a new target system model is defined.

Definition 5 (Involvement-Contention Model): A target parallel system $M = (TG, \omega, \alpha, i)$ consists of a set of possibly heterogenous processors $P$ connected by the communication network $TG = (P, L)$. This dedicated system has the following property: i) local communication has zero costs.

So, in comparison with the contention model (Definition 3), the involvement-contention model departs from the assumption of a dedicated communication subsystem. Instead, the role of the processors in communication is described by the new components $\alpha$ – for overhead – and $i$ – for (direct) involvement, see Figure 2.

Let $R = (L_1, L_2, \ldots, L_l)$ be the route for a communication from $P_{src}$ to $P_{dst}$.

Overhead $o_s(e, P_{src})$ is the computational overhead, i.e. the execution time, incurred by processor $P_{src}$ for preparing the transfer of the communication associated with edge $e$ and $o_v(e, P_{dst})$ is the overhead incurred by processor $P_{dst}$ after receiving $e$.

![Fig. 2. The decomposing of the processor's participation in communication into overhead and direct involvement](image)

Involvement $i_s(e, L_1)$ is the computational involvement, i.e. execution time, incurred by processor $P_{src}$ during the transfer of edge $e$ and $i_v(e, L_1)$ is the computational involvement incurred by $P_{dst}$ during the transfer of $e$.

The overhead depends largely on the employed communication environment (e.g. MPI, Active Messages, TCP/IP) and is thereby normally unaffected by the utilised communication resources. In contrast, the involvement depends to a large extent on the capabilities of the utilised communication resources. Hence, the processor involvement is characterised by the outgoing or incoming link utilised for a communication.

With the distinction between the sending ($o_s, i_s$) and the receiving side ($o_v, i_v$) of communication, all three types of communication – third party, one sided, two sided – can be precisely represented. The corresponding functions are simply defined accordingly, e.g. $i_s(e, L) = i_v(e, L) = 0$ for involvement-free third party communication.

A. Scheduling edges on the processors

Incorporating overhead and involvement into contention aware task scheduling is accomplished by extending edge scheduling so that edges are not only scheduled on the links but also on the processors.

So, the start time of an edge $e \in E$ on a processor $P \in P$ is denoted by $t_s(e, P)$. Let $R = (L_1, L_2, \ldots, L_l)$ be the route from $P_{src}$ to $P_{dst}$. The finish time of $e$ on $P_{src}$ is

$$t_f(e, P_{src}) = t_s(e, P_{src}) + o_s(e, P_{src}) + i_s(e, L_1) \quad (7)$$

and on $P_{dst}$ it is

$$t_f(e, P_{dst}) = t_s(e, P_{dst}) + o_v(e, P_{dst}) + i_v(e, L_1). \quad (8)$$

Figure 3 illustrates scheduling under the involvement-contention model.

For a meaningful and feasible schedule, the scheduling of the edges on the processors must obey the processor constraint (Condition 1) and the following condition.

Condition 3 (Causality in Involvement Scheduling): Let $R = (L_1, L_2, \ldots, L_l)$ be the route for the communication of edge $e_{ij} \in E$, $n_i, n_j \in V$, from $P_{src} \in P$ to $P_{dst} \in P$, $P_{src} \neq P_{dst}$.

To assure the node strictness of $n_i$

$$t_s(e_{ij}, P_{src}) \geq t_f(n_i, P_{src}). \quad (9)$$

Edge $e_{ij}$ can be transfered on the first link $L_1$ only after the overhead completed on the source processor $P_{src}$:

$$t_s(e_{ij}, L_1) \geq t_s(e_{ij}, P_{src}) + o_s(e_{ij}, P_{src}) \quad (10)$$
To assure the causality of the direct involvement on the destination processor \( P_{\text{dst}} \)

\[
t_s(e_{ij}, P_{\text{dst}}) \geq t_f(e_{ij}, L_1) - i_s(e_{ij}, L_1).
\]  
(11)

The three inequalities can be observed in effect in Figure 3.

**B. Scheduling**

Few alterations are imposed by the new model on the edge scheduling on the links and on the scheduling of the nodes.

**Edge scheduling on links** – The scheduling of the edges on the links is only constrained by (10) of the Causality Condition 3: \( t_s(e_{ij}, L_1) \geq t_s(e_{ij}, P_{\text{src}}) + o_s(e_{ij}, P_{\text{src}}) \). In comparison, under the contention model edge \( e_{ij} \) can start on the first link \( L_1 \) immediately after its origin node \( n_i \) has finished, \( t_s(e_{ij}, L_1) \geq t_f(n_i) \). Note, the rest of the edge scheduling procedure is completely unaffected by the scheduling of the edges on the processors and remains unmodified.

**Node scheduling** – To adapt the scheduling of the nodes to the new model, it is only necessary to redefine the finish time of the edge.

**Definition 6 (Edge Finish Time):** Let \( G = (V, E, w, c) \) be a DAG and \( M = ((P, L), \omega, o, i) \) a parallel system. The finish time of \( e_{ij} \in E \), \( n_i, n_j \in V \) is

\[
t_f(e_{ij}) = \begin{cases} 
  t_f(n_i) & \text{if } \text{proc}(n_i) = \text{proc}(n_j) \\
  t_f(e_{ij}, \text{proc}(n_j)) & \text{otherwise} 
\end{cases}
\]  
(12)

As it can be expected, scheduling under the involvement-contention model remains an NP-hard problem [15]. This is easy to see, as the involvement model is based on the contention model, which is NP-hard.

**V. SCHEDULING ALGORITHMS**

In contrast to scheduling on the links, the scheduling of the edges on the processors, which seems at first sight a simple extension, has a strong impact on the working mode of scheduling algorithms. Essentially, the problem is that at the time a node \( n \) is scheduled, it is generally unknown to where its successor nodes will be scheduled. It is not even known if the corresponding outgoing communications will be local or remote. Thus, no decision can be taken whether to schedule \( n \)'s leaving edges on its processor or not. Later, at the time a successor is scheduled, the period of time directly after node \( n \) might have been occupied with other nodes. Hence, there is no space left for the scheduling of the corresponding edge. Scheduling under the LogP model faces the same problem with the scheduling of \( o \) for each communication [8].

Two different approaches to handle the described issue in scheduling under the involvement-contention model, which have reasonable complexity, can be distinguished: i) direct scheduling; ii) scheduling based on a given processor allocation. This paper concentrates on direct scheduling; scheduling based on a given processor allocation is discussed elsewhere [15].

**A. Direct scheduling**

By direct scheduling it is meant that the processor allocation and the start/finish time attribution of a node are done in one single step. The application of the scheduling method from contention scheduling is inadequate under the new model, since the decision whether a communication is remote or local is made to late. Consequently, it is necessary to investigate how edges can be scheduled earlier.

The most viable solution is to reserve an appropriate time interval after a node for the later scheduling of the leaving edges. This must be done in a worst case fashion, which means the interval must be large enough to accommodate all leaving edges. A straightforward manner to do so, is to schedule all leaving edges on the source processor, directly after the origin node. The scheduling of the edges on the links and the destination processors can take place when the destination

![Fig. 1. The three types of interprocessor communication](image-url)

![Fig. 3. Scheduling under the involvement-contention model: edges are also scheduled on the processors; S - routing station (switch or other processor)](image-url)
node is scheduled. If the destination node is scheduled on the same processor as the origin node, the corresponding edge, which was provisionally scheduled with the origin node, is simply removed from that processor.

On heterogeneous systems, the described provisional scheduling of an edge on its source processor must consider that the involvement depends on the first link of the utilised route. Again, as the route is unknown at the time of the scheduling, the worst case must be assumed. So, the **provisional finish time** of edge \( e_{ij} \in E \) on its source processor \( P = \text{proc}(n_i), P \in P \), is

\[
t_f(e_{ij}, P) = t_s(e_{ij}, P) + o_s(e_{ij}, P) + i_{s, \text{max}}(e_{ij}, P),
\]

(13)

where \( i_{s, \text{max}}(e_{ij}, P) = \max_{L \in \text{L} \text{leaving} \ P \{i_s(e_{ij}, L)\}} \). When the destination node \( n_j \) is scheduled, the finish time must be reduced, if applicable, to the correct value.

With the reservation of a time interval for the outgoing edges on the processor, the rest of scheduling can be performed as under the contention model.

### B. List scheduling

In this section list scheduling [1], as a heuristic using the direct scheduling approach, is adapted for the involvement-contention model. In the simple, but common, variant of list scheduling (Algorithm 1), the nodes are ordered according to a priority in the first part of the algorithm.

**Algorithm 1** List scheduling

1. \( \triangleright \text{ I. Part:} \)
2. Sort nodes \( n \in V \) into list \( L \), according to priority scheme and precedence constraints.
3. \( \triangleright \text{ II. Part:} \)
4. for each \( n \in L \)
5. Find processor \( P \in P \) that allows earliest finish time of \( n \).
6. Schedule \( n \) on \( P \).

The schedule order of the nodes is important for the schedule length and many different priority schemes have been proposed, e.g. [1, 17, 19]. A common and usually good priority is the node’s **bottom level** \( bl \), which is the length of the longest path leaving the node. Recursively defined it is

\[ bl(n_i) = w(n_i) + \max_{n_j \in \text{succ}(n_i)} \{c(e_{ij}) + bl(n_j)\} \]

(14)

Under the involvement-contention model, and in accordance with the direct scheduling approach, the scheduling of a node (line 6 in Algorithm 1) is performed as described in Algorithm 2.

As under the contention model, finding the processor that allows the earliest finish time of a node involves the tentative scheduling on every processor (line 5 of Algorithm 1), including the incoming edges on the links and the destination processor. Only this way, the communication contention and the processor involvement is considered in the scheduling decisions.

**Algorithm 2** Scheduling of node \( n_j \) on processor \( P \) in involvement-contention model

1. for each \( n_i \in \text{pred}(n_j) \) do
2. if \( \text{proc}(n_i) = P \) then
3. remove \( e_{ij} \) from \( P \)
4. for each \( n_i \in \text{pred}(n_j) \) in a definite order do
5. if \( \text{proc}(n_i) \neq P \) then
6. determine route \( R = (L_1, L_2, \ldots, L_t) \) from \( \text{proc}(n_i) \) to \( P \)
7. correct \( t_f(e_{ij}, \text{proc}(n_i)) \)
8. schedule \( e_{ij} \) on \( R \)
9. schedule \( e_{ij} \) on \( P \)
10. schedule \( n_j \) on \( P \)
11. for each \( n_k \in \text{succ}(n_j) \) in a definite order do \( \triangleright \) reserve space for leaving edges
12. schedule \( e_{jk} \) on \( P \) with worst case finish time

To determine the start time (i.e. the “schedule” lines in Algorithm 2) of a node or edge (elements) on a processor or link (resources), the earliest idle interval \( [A, B] \) is searched on the resource that fulfills the discussed constraints, and which is large enough to accommodate the respective element. With the insertion technique [9] this interval is between two elements already scheduled on the resource, and with the end technique [1] it is the open interval after the finish time of the resource, i.e. \( B = \infty \). Under the involvement-contention model, insertion scheduling is more indicated, since the removing of provisionally scheduled edges leaves gaps, which should be filled by other nodes or edges.

In comparison to contention aware list scheduling, the time complexity under the involvement-contention model does not increase. The complexity of the second part of list scheduling is \( O(P(V + E_0(\text{routing}))) \) (end technique) or \( O(V^2 + PE^2 O(\text{routing})) \) (insertion technique) [15], where \( O(\text{routing}) \) is the complexity of routing, which is often \( O(P) \) or even \( O(1) \). For comparison, the complexity expressions for the classic model are obtained by setting \( O(\text{routing}) = O(1) \).

### VI. Experimental results

For the evaluation of the new involvement-contention model and the proposed list scheduling heuristic, the experimental methodology proposed in [14] is employed. A large set of graphs is generated and scheduled by algorithms under the different models to several target systems. From the produced schedules code is generated, using C and MPI, and executed on the real parallel systems. The execution times of these codes directly show which algorithms and models produce the best schedules.

The evaluation is divided into two parts: the evaluation of the accuracy and the evaluation of the execution time. In the following the most important results are presented; for more experiments and details refer to [15].

**A. Improved accuracy**

In [14], the accuracy of the classic and the contention model are examined using the mentioned methodology. To evaluate
the accuracy of the new involvement-contention model, the schedules produced in those experiments under the contention model with heuristic LS(dls) – list scheduling with DLS’s node order [13] – are rescheduled, but now under the involvement-contention model. This rescheduling allocates the nodes to the same processors in the same local order as in the original schedule. Consequently, the code generated for the schedule under the involvement-contention model would be identical to the one generated for the original schedule, under the contention model. Hence the execution time of that code would be identical to the execution time of the original code, which was already obtained experimentally. By comparing this execution time with the prediction under the involvement-contention model, the new model’s accuracy is determined.

Three target systems were employed in the experiments of [14]: a cluster (BOBCAT) of 16 PCs, modelled as a switched network; a shared memory multiprocessor system Sun E3500 with 8 processors, modelled as a bus network; a massively parallel system Cray T3E-900 with a total of 344 processors, modelled as a fully connected network.

Due to the lack of a profound insight into the target systems’ communication mechanisms and their MPI implementations, 100% involvement is assumed, i.e. the source and destination processors are involved during the entire communication time on the first and last link, respectively: \( i_s(e, L_1) = t_f(e, L_1) \) and \( i_s(e, L_1) = t_f(e, L_1) \). The overhead is intuitively set to the experimentally measured setup time: \( o_s(e, P) = setup\_time \). While it is clear that this definition of the overhead and the involvement is probably not an accurate description of the target systems’ communication behaviour, it is very simple. The idea is to demonstrate that accuracy and efficiency of scheduling can be improved even with a rough but simple estimate of the overhead and involvement functions.

1) Results: Figure 4 visualises the average accuracy deviations \( \Delta acc(S) \) with

\[
\Delta acc(S) = \left\{ \begin{array}{ll}
acc(S) - 1 & \text{if } acc(S) \geq 1 \\
1/acc(S) - 1 & \text{if } acc(S) < 1
\end{array} \right.,
\]

(15)

where \( acc(S) \) is the ratio of the execution time of the code produced for schedule \( S \) to its schedule length \( sl(S) \). The communication to computation ratio is defined as the sum of all communication costs divided by the sum of all computation costs, \( CCR = \frac{\sum_{e \in E} \omega(w)}{\sum_{w \in V} \omega(w)} \).

It is immediately apparent from Figure 4 that the accuracy profoundly improved under the new involvement-contention model. While this improvement is already considerable for low communication (\( CCR = 0.1 \)), it is much more significant for medium (\( CCR = 1 \)) and especially high communication (\( CCR = 10 \)). The length of a schedule is now in a region, where it can be seriously considered an estimation of the real execution time.

The scheduling accuracy under the involvement-contention model is still not perfect, especially for low communication (\( CCR = 0.1 \)). A possible explanation might be the blocking communication mechanisms used in MPI implementations [7], which is contrary to the assumption of non-blocking communication made by the involvement contention model. Also, the employed overhead and involvement functions are very rough estimates, a better approximation of these functions might improve the accuracy. In any case, there will always be a difference between the predicted and the real execution time. Under this perspective, the results obtained for the T3E are very satisfying, which is probably due to the fact that the T3E-900, being a massively parallel system specifically designed for parallel processing, is the most predictable among the target systems.

B. Improved execution time

To determine if the new model also produces schedules, which lead to shorter execution times, new experiments were performed using the mentioned methodology of [14].

The workload consists of seven different graph types:intree, out-tree, multiple fork-join, Gaussian elimination, stencil pipeline and random structure. The seven DAGs are generated three times to support the analysis of the three CCR values of approximately 0.1, 1, 10.

These graphs are scheduled by a list scheduling heuristic with the insertion technique, where the nodes are ordered according to their bottom levels. This algorithm is applied under the classic (“classic”), the contention (“cont”) and the involvement-contention model (“invo-cont”).

Code is generated for the obtained schedules and executed on two different target systems: Sun E3500 and BOBCAT. Both systems were modelled as in the previous experiments of Section VI-A.

1) Results: Figures 5 and 6 visualise the average normalised speedups of the three different models on different configurations of the two target system. The speedup is defined as the ratio of the sequential time \( seq(G) = \sum_{n \in V} \omega(n) \) (local communication has zero costs) to the execution time of the code produced for schedule \( S \).

The involvement-contention model clearly demonstrated its ability to produce schedules with significantly reduced execution times. While this ability is very distinct for medium communication (\( CCR = 1 \)), with speedup improvements of up to 82% (Gauss elimination), it is only noticeable with many processors for low communication (BOBCAT with 16 processors). The speedup improvements for high communication of the contention and the involvement-contention model less important, because the absolute speedup is below 1 for all models.

Note, the improvements are achieved for those types of DAGs for which it is most important. For DAGs with low communication, a reasonable speedup is already provided under the classic model. Further, for DAGs with high communication, the results suggest that an efficient parallelisation is not possible.

Despite the very good results, the efficiency improvement lags behind the accuracy improvement demonstrated in the previous subsection. A possible explanation lies in the employed heuristic. To exploit the full potential of this new model, the investigation of algorithms which are better adapted to the new model is needed.
VII. Conclusions

This paper investigated the processor involvement in communication and its integration into task scheduling. A new system model was proposed, which extends the scheduling of the edges on the links to their scheduling on the processors. This technique allows to reflect the different types of processor involvement and the distinction between overhead and direct involvement. The impact of the new involvement-contention model on the scheduling heuristics were discussed and list scheduling was adapted to the new model. Extensive experiments demonstrated that the involvement-contention model significantly improves the accuracy and the execution time of the produced schedules. To further improve the obtained results, more research regarding algorithms which are better adapted to the new model is indicated.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support by FCT (Fundação para a Ciência e a Tecnologia) and FSE (Fundo Social Europeu) through grant number SFRH/BPD/14990/2004 in the context of the III European Framework of Support. Also, the authors would like to acknowledge the support of the European Commission through ARI grant number HPRI-CT-1999-00026 and through TMR grant number ERB FMGE CT950051 (the TRACS Programme at EPCC).

REFERENCES


