Communication contention in task scheduling

Oliver Sinnen* L A. Sousa†

*University of Auckland, o.sinnen@auckland.ac.nz

†Technical University of Lisbon,

Communication Contention in Task Scheduling

Oliver Sinnen and Leonel A. Sousa, *Senior Member*, *IEEE*

Abstract—Task scheduling is an essential aspect of parallel programming. Most heuristics for this NP-hard problem are based on a simple system model that assumes fully connected processors and concurrent interprocessor communication. Hence, contention for communication resources is not considered in task scheduling, yet it has a strong influence on the execution time of a parallel program. This paper investigates the incorporation of contention awareness into task scheduling. A new system model for task scheduling is proposed, allowing us to capture both end-point and network contention. To achieve this, the communication network is reflected by a topology graph for the representation of arbitrary static and dynamic networks. The contention awareness is accomplished by scheduling the communications, represented by the edges in the task graph, onto the links of the topology graph. Edge scheduling is theoretically analyzed, including aspects like heterogeneity, routing, and causality. The proposed contention-aware scheduling preserves the theoretical basis of task scheduling. It is shown how classic list scheduling is easily extended to this more accurate system model. Experimental results show the significantly improved accuracy and efficiency of the produced schedules.

Index Terms—Parallel processing, concurrent programming, scheduling and task partitioning, communication contention, heterogeneous system model.

1 Introduction

An essential aspect of parallel programming is scheduling, which is the spacial and temporal assignment of the tasks of a program to the processors of the target system. In task scheduling, the program is represented as a directed acyclic graph (DAG), where a node represents a task and an edge is the communication between two tasks. As the general task scheduling problem is NP-hard [26], [33], heuristics try to find near optimal solutions (e.g., [2], [3], [13], [14], [18], [20], [25], [26], [34], [35], [36]).

Most of these heuristics have in common that they employ a very idealized model of the target parallel system. In this model, every processor possesses a dedicated communication subsystem, the processors are fully connected, and all communications can be performed concurrently.

Intuition suggests that these assumptions are not met on real parallel systems, which is confirmed by experiments [23], [28]. An important aspect missed by the model is the contention for communication resources. If a resource is occupied by one communication, any other communication requiring the same resource has to wait until it becomes available. In turn, the task depending on the delayed communication is also forced to wait. Thus, conflicts among communications generally result in a higher overall execution time.

This paper investigates the incorporation of contention awareness into task scheduling. In previous attempts, a few algorithms were proposed that consider network contention [11], [21], [27], [30] or end-point contention [4], [6], [15], [22]. The new system model for task scheduling proposed in this paper is capable of capturing both end-point and network contention. This is achieved with the proposal of a new graph model for the representation of the system's network topology. This new topology graph allows, in contrast to the undirected graph model of previous approaches, the reflection of static and dynamic networks of arbitrary, possibly heterogenous, structure. Further, it also permits the distinction between different communication link types, i.e., the distinction between half duplex and full duplex links and busses.

Based on the topology graph, contention awareness is achieved by scheduling the edges of the DAG onto the links of the topology graph. While the idea of this edge scheduling was already proposed in [27], no theoretical background has been established so far. This paper investigates the theoretical background of edge scheduling, including aspects like heterogeneity, routing, and causality. It is also analyzed how the division of a message into packets affects edge scheduling and the consideration of contention. The proposed contention-aware scheduling preserves the theoretical basis of task scheduling. While the topology graph is able to represent complex communication networks, this complexity is encapsulated within the topology graph model and not revealed to the edge scheduling. In particular, classic list scheduling is easily extended to this more accurate system model. Finally, experimental results will be presented that show the significantly improved accuracy and efficiency of the schedules produced under the new scheduling model.

This paper continues in Section 2 by establishing the background of classic task scheduling together with the introduction of the necessary notions and definitions. Section 3 discusses contention in communication and reviews previous approaches to achieve an awareness for it in task scheduling. The new model for the representation of the target system's communication network is proposed in Section 4. Section 5 investigates the key technique to

- *O. Sinnen is with the Department of Electrical and Computer Engineering, University of Auckland, Private Bag 92019, Auckland, New Zealand. E-mail: o.sinnen@auckland.ac.nz.*
- *L.A. Sousa is with INESC-ID, Instituto Superior Ténico, Technical University of Lisbon, Rua Alves Redol 9, P-1000-029 Lisboa, Portugal. E-mail: las@inesc-id.pt.*

Manuscript received 25 Jan 2004; revised 30 June 2004; accepted 18 Oct. 2004; published online 21 Apr. 2005.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0024-0104.

achieve contention awareness: edge scheduling. Based on the network model and edge scheduling, Section 6 discusses how task scheduling is made sensible for contention and how list scheduling is adapted. Section 7 presents an experimental evaluation of the new scheduling model on real parallel systems and the paper concludes with Section 8.

2 Task Scheduling

In task scheduling, the program to be scheduled is represented by a directed acyclic graph:

Definition 1 (Directed Acyclic Graph (DAG)). *A DAG is a directed acyclic graph $G = (\mathbf{V}, \mathbf{E}, w, c)$ representing a program $\mathcal{P}$. The nodes in $\mathbf{V}$ represent the tasks of $\mathcal{P}$ and the edges in $\mathbf{E}$ the communications between the tasks. An edge $e_{ij} \in \mathbf{E}$ represents the communication from node n_i to node n_j. The positive weight $w(n)$ associated with node $n \in \mathbf{V}$ represents its computation cost and the nonnegative weight $c(e_{ij})$ associated with edge $e_{ij} \in \mathbf{E}$ represents its communication cost.*

All instructions or operations of one task are executed in sequential order and there is no parallelism within a task. 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 $\{n_x \in \mathbf{V} : e_{xi} \in \mathbf{E}\}$ of all direct **predecessors** of n_i is denoted by $\mathbf{pred}(n_i)$ and the set $\{n_x \in \mathbf{V} : e_{ix} \in \mathbf{E}\}$ of all direct **successors** of n_i is denoted by $\mathbf{succ}(n_i)$. A node $n \in \mathbf{V}$ without predecessors, $\mathbf{pred}(n) = \emptyset$, is named source node and if it is without successors, $\mathbf{succ}(n) = \emptyset$, it is named sink node.

2.1 Classic Task Scheduling

Most scheduling algorithms employ a strongly idealized model of the target parallel system [2], [3], [13], [14], [18], [20], [25], [34], [35]. This model, which shall be referred to as the classic model, is defined in the following, including the generalization towards heterogenous processors:

Definition 2 (Classic System Model). *A parallel system $M_{classic} = (\mathbf{P}, \omega)$ consists of a finite set of processors $\mathbf{P}$ connected by a communication network. The processor heterogeneity, in terms of processing speed, is described by the execution time function ω. This system is dedicated to the execution of the scheduled program and has the following properties:*

1. ***Dedicated processor**—A processor $P \in \mathbf{P}$ can execute only one task at a time and the execution is not preemptive.*
2. ***Zero cost local communication**—The cost of communication between tasks executed on the same processor, local communication, is negligible and therefore considered zero.*
3. ***Communication subsystem**—Interprocessor communication is performed by a dedicated communication subsystem. The processors are not involved in communication.*
4. ***Concurrent communication**—Interprocessor communication in the system is performed concurrently and there is no contention for communication resources.*
5. ***Fully connected**—The communication network is fully connected. Every processor can communicate with every other processor via a dedicated identical communication link.*

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 $\mathcal{S}$ of a DAG $G = (\mathbf{V}, \mathbf{E}, w, c)$ on a target system $M_{classic} = (\mathbf{P}, \omega)$, the following terms are defined: $t_s(n, P)$ denotes the **start time** and $\omega(n, P)$ the **execution time** of node $n \in \mathbf{V}$ on processor $P \in \mathbf{P}$. 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 $proc(n)$. Further, let $t_f(P) = \max_{n \in \mathbf{V}: proc(n)=P}\{t_f(n)\}$ be the **processor finish time** of P and let $sl(\mathcal{S}) = \max_{n \in \mathbf{V}}\{t_f(n)\}$ be the **schedule length** (or makespan) of $\mathcal{S}$, assuming $\min_{n \in \mathbf{V}}\{t_s(n)\} = 0$. The **sequential time** is G's execution time on one processor (local communication has zero costs) $seq(G) = \sum_{n \in \mathbf{V}} w(n)$.

For such a defined schedule to be feasible, the following two conditions most be fulfilled for all nodes in G:

Condition 1 (Processor Constraint). *For any two nodes n_i, $n_j \in \mathbf{V}$,*

$$proc(n_i) = proc(n_j) = P \Rightarrow \begin{cases} & t_f(n_i, P) \le t_s(n_j, P) \\ \text{or} & t_f(n_j, P) \le t_s(n_i, P). \end{cases} \quad (1)$$

Condition 2 (Precedence Constraint). *For $n_i, n_j \in \mathbf{V}$, $e_{ij} \in \mathbf{E}$, $P \in \mathbf{P}$,*

$$t_s(n_j, P) \ge t_f(e_{ij}), \quad (2)$$

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

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

$$t_{dr}(n_j, P) = \max_{e_{ij} \in \mathbf{E}, n_i \in \mathbf{pred}(n_j)} \{t_f(e_{ij})\} \quad (3)$$

and, hence,

$$t_s(n, P) \ge t_{dr}(n, P) \quad (4)$$

for all $n \in \mathbf{V}$. If $\mathbf{pred}(n_j) = \emptyset$, i.e., n_j is a source node, $t_{dr}(n_j) = t_{dr}(n_j, P) = 0$, for all $P \in \mathbf{P}$. Owing to the system model, the edge finish time only depends on the finish time of the origin node and the communication time.

Definition 3 (Edge Finish Time). *The edge finish time of $e_{ij} \in \mathbf{E}$ is given by*

$$t_f(e_{ij}) = t_f(n_i) + \begin{cases} 0 & \text{if } proc(n_i) = proc(n_j) \\ c(e_{ij}) & \text{otherwise.} \end{cases} \quad (5)$$

Thus, communication can overlap with the computation of other nodes (Property 3 of Definition 2), an unlimited number of communications can be performed at the same time (Property 4), and communication has the same cost $c(e_{ij})$, indifferent of the origin and the destination processor (Property 5), unless communication is local (Property 2).

2.2 Scheduling Heuristics

The scheduling problem is to find a schedule with minimal length. As this problem is NP-hard [26], [33], many heuristics have been proposed for its solution. A heuristic must schedule a node on a processor so that it fulfils all resource (1) and precedence constraints (2). The following condition formulates that.

Condition 3 (Scheduling Condition). *Let $G = (\mathbf{V}, \mathbf{E}, w, c)$ be a DAG and $\mathbf{M}_{classic} = (\mathbf{P}, \omega)$ be a parallel system. Let $[A, B]$, $A, B \in [0, \infty]$, be an idle time interval on $P \in \mathbf{P}$, i.e., an interval in which no node is executed. A free node $n \in \mathbf{V}$ can be scheduled on P within $[A, B]$ if*

$$\max\{A, t_{dr}(n, P)\} + \omega(n, P) \leq B. \tag{6}$$

A free node is a node whose predecessors have already been scheduled, which is a requisite for the calculation of the DRT. So, Condition 3 allows node n to be scheduled between already scheduled nodes (**insertion technique**) [17], i.e., $[A, B] = [t_f(n_{P_l}, P), t_s(n_{P_{l+1}}, P)]$, or after the finish time of processor P (**end technique**) [1], i.e., $[A, B] = [t_f(P), \infty]$.

2.2.1 List Scheduling

The best known scheduling heuristic is list scheduling, e.g., [1], as given in Algorithm 1. In this simple, but common, variant of list scheduling, the nodes are ordered according to a priority in the first part of the algorithm. The schedule order of the nodes is important for the schedule length and many different priority schemes have been proposed [1], [16], [30], [34]. 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

$$bl(n_i) = w(n_i) + \max_{n_j \in \mathbf{succ}(n_i)} \{c(e_{ij}) + bl(n_j)\}. \tag{7}$$

Algorithm 1. List scheduling.
1: ▷ *1. Part:*
2: Sort nodes $n \in \mathbf{V}$ into list L, according to priority scheme and precedence constraints.
3: ▷ *2. Part:*
4: **for** each $n \in L$ **do**
5: Find processor $P \in \mathbf{P}$ that allows earliest finish time of n.
6: Schedule n on P.

To determine the start time of a node, the earliest interval $[A, B]$ is searched on each processor that complies with the scheduling Condition 3, either using the insertion or the end technique. For the found interval $[A, B]$, the start time of node n is determined as

$$t_s(n, P) = \max\{A, t_{dr}(n, P)\}. \tag{8}$$

Node n is scheduled on the processor that allows the earliest finish time $t_f(n, P) = t_s(n, P) + \omega(n, P)$, thereby integrating the awareness for processor heterogeneity.

For most of the priority schemes, the complexity of the first part is $O(\mathbf{V} \log \mathbf{V} + \mathbf{E})$ [29]. The complexity of the second part is $O(\mathbf{P}(\mathbf{V} + \mathbf{E}))$ (end technique) or $O(\mathbf{V}^2 + \mathbf{PE})$ (insertion technique) [29].

3 Contention Awareness

From Definition 2 of the system model, it is clear that there is absolutely no consideration of contention for communication resources in the classic scheduling approach. In the past, there were some attempts to consider contention in task scheduling, which shall be discussed in the next paragraphs. Roughly, contention can be divided into end-point and network contention.

3.1 End-Point Contention

End-point contention refers to the contention in the interfaces that connect the processors with the communication network. Only a limited number of communications can pass from the processor into the network and from the network into the processor at one instance in time.

End-point contention has been considered in task scheduling with the one-port model [4] and with the parameter g of the LogP model [9]. The one-port model extends the classic model by associating a communication port with each processor. In the few scheduling algorithms based on the LogP model, e.g., [6], [15], [22], the number of concurrent communications that can leave or enter a processor is limited, since the time interval between two consecutive communications must be at least g.

3.2 Network Contention

Network contention is caused by the limited number of resources within the network. To successfully handle this kind of contention in scheduling, an accurate model of the network topology is required. Static networks, i.e., networks with fixed connections between the units of the system, are commonly represented as undirected graphs. A vertex represents a processor and an undirected edge represents a bidirectional communication link between two processors.

A few scheduling algorithms employed this undirected topology graph to achieve the awareness of network contention, e.g., MH (Mapping Heuristic) [11], DLS (Dynamic Level Scheduling) [27], and BSA (Bubble Scheduling and Allocation) [21]. The more realistic view of the network traffic is gained by DLS and BSA through the scheduling of the edges of the DAG on the links of the network graph. This idea of edge scheduling was first proposed in [27], however, without further details. Edge scheduling has, thanks to its strong similarity with task scheduling, the potential to accurately reflect contention in communication. Therefore, the here proposed approach to contention awareness is based on the edge scheduling idea.

4 Network Model

In this section, a network model for contention-aware task scheduling is proposed. First, a topology graph more general

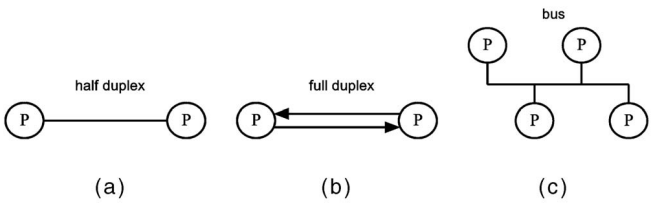


Fig. 1. Representing (a) a half duplex link, (b) a full duplex link, and (c) a bus with an undirected edge, two counter-directed edges, and a hyperedge, respectively.

than the undirected graph is proposed, which is capable of capturing both end-point and network contention.

4.1 Topology Graph

Definition 4 (Topology Graph). *The topology of a communication network is modeled as a graph* $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$, *where* $\mathbf{N}$ *is a finite set of vertices,* $\mathbf{P}$ *is a subset of* $\mathbf{N}$, $\mathbf{P} \subseteq \mathbf{N}$, $\mathbf{D}$ *is a finite set of directed edges, and* $\mathbf{H}$ *is a finite set of hyperedges. A vertex* $N \in \mathbf{N}$ *is referred to as a* ***network vertex****, of which two types are distinguished: a vertex* $P \in \mathbf{P}$ *represents a processor, while a vertex* $S \in \mathbf{N}, \notin \mathbf{P}$ *represents a switch. A directed edge* $D_{ij} \in \mathbf{D}$ *represents a* ***directed communication link*** *from network vertex* N_i *to network vertex* N_j, $N_i, N_j \in \mathbf{N}$. *A hyperedge* $H \in \mathbf{H}$ *is a subset of two or more vertices of* $\mathbf{N}$, $H \subseteq \mathbf{N}$, $|H| > 1$, *representing a* ***multidirectional communication link*** *between the network vertices of* H. *For convenience, the union of the two edge sets* $\mathbf{D}$ *and* $\mathbf{H}$ *is designated the link set* $\mathbf{L}$, *i.e.,* $\mathbf{L} = \mathbf{D} \cup \mathbf{H}$, *and an element of this set is denoted by* L, $L \in \mathbf{L}$. *The nonnegative weight* $b(L)$, *associated with a link* $L \in \mathbf{L}$, *represents its* ***relative data rate****.*

A hyperedge [5] is a generalization of an undirected edge as it can be incident on more than two vertices. It is assumed that a switch is ideal, i.e., there is no contention within the switch. Regarding the routing of a message in the network, switch and processor vertices are treated identically—a switch just cannot execute tasks.

The topology graph is an unusual graph in that it integrates two types of edges—directed and hyperedges. Thereby, it is able to address the shortcomings of the undirected graph network model.

Half and full duplex links—Through the introduction of directed edges, it can be distinguished between half and full duplex communication links. A hyperedge incident on two network vertices, i.e., an undirected edge, models a half duplex link (Fig. 1a), as in the undirected graph. A full duplex link is represented by two counter directed edges (Fig. 1b). Even a unidirectional communication link, as sometimes encountered in ring-based topologies (e.g., networks using the SCI (Scalable Coherent Interface) standard [10]) can be modeled with one directed link in the corresponding direction.

Buses—A hyperedge is a natural representation of a bus topology as illustrated in Fig. 1c. In the undirected graph model, buses are inaccurately reflected by fully connected networks, e.g., as shown in Fig. 2b. In both representations, every processor is adjacent to every other vertex. However, for contention scheduling, the fact that the bus is shared among all processors is crucial.

Switches—With the introduction of a switch vertex, it is now possible to represent dynamic networks containing communication switches. For example, in a binary tree network, only the leaves are processors and all other vertices are switches (Fig. 2a).

Another example is a cluster of workstations connected through the switch of a LAN (Local Area Network) (Fig. 2c). In the undirected topology graph, such a network must be represented as fully connected (Fig. 2b). Thus, every processor has its own link with every other processor, reducing the modeled impact of (end-point) contention.

Moreover, a switch vertex is utilized to model the bottleneck caused by the interface that connects a processor to the network, i.e., end-point contention. Imagine one processor in a two-dimensional processor mesh, i.e., it has direct links to four neighbors, whose network interface limits the number of communications to one at a time. Deploying a switch, as in Fig. 2d, reflects this situation, which, for example, is encountered in the Intel Paragon [10].

Heterogeneity—Heterogeneity, in terms of the links' transfer speeds, is captured by the topology graph through the association of relative data rates with the links. Of course, for homogenous systems, the relative data rate is set to 1 or simply ignored.

The topology graph is conservative in that it contains other simpler models. For example, in a network without busses, it reduces to a graph without hyperedges (only directed and undirected edges) or, in a static network, all network vertices are processors. In particular, the topology graph is a generalization of the undirected topology graph, which is given with $\mathbf{N} = \mathbf{P}$ (no switches), $|H| = 2 \forall H \in \mathbf{H}$ (only undirected edges, i.e., hyperedges with two vertices), and $\mathbf{D} = \emptyset$ (no directed edges).

4.2 Routing

To schedule edges on communication links, it must be known which links are involved in an interprocessor

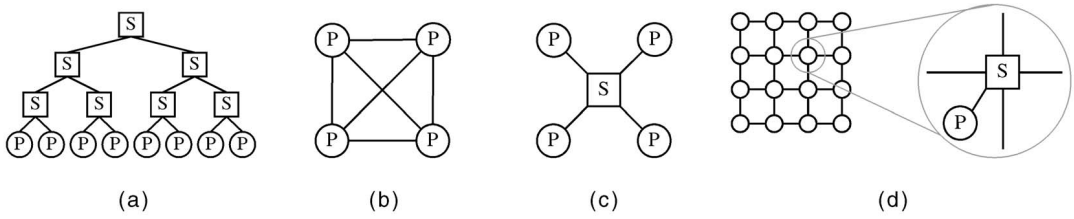


Fig. 2. Networks modeled with the topology graph (P—processor, S—switch). (a) Binary tree network. (b) Fully connected processors. (c) Lan with switch. (d) Switch used to model network interface bottleneck.

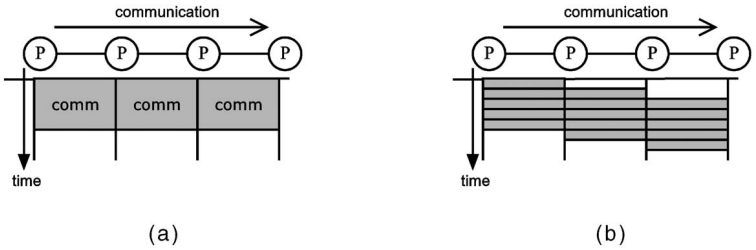


Fig. 3. (a) Package-based store-and-forward routing is similar to (b) cut-through routing.

communication and how. Essentially, this is described by the routing algorithm and policy of the target system's network [10], [19], which will be reviewed from the perspective of edge scheduling in the next paragraphs.

Nonadaptive versus adaptive routing—The contention-aware scheduling proposed in this paper assumes nonadaptive routing. Adaptive routing is seldom used in current parallel machines [10]. Moreover, it would require a modeling down to the data packet level in order to accurately reflect the traffic in the network at every instance in time.

Switching Strategy—In circuit switching, the path for the communication is established—by the routing algorithm—at the beginning of the transfer and all data takes the same circuit, i.e., route. With packet switching, the message is split into packets and routing decisions are made separately for every packet. Since edge scheduling does not consider the possible division of communications into packets, circuit switching is assumed.

Store-and-forward versus cut-through routing—In store-and-forward routing, the data of a communication is stored at every network station until the entire message, or packet, has arrived and is subsequently forwarded to the next station of the route. In cut-through routing, a station immediately forwards the data to the next station—the message "cuts through" the station. Store-and-forward routing has a higher latency than cut-through routing. It is used in wide area networks and was used in several early parallel computers, while modern parallel systems employ cut-through routing, e.g., Cray T3D/T3E, IBM SP-2 [10]. Moreover, systems that use store-and-forward behave approximately like systems with cut-through routing, regarding edge scheduling, when the communication is divided into several packets, as illustrated in Fig. 3. Thus, in edge scheduling, cut-through routing will be assumed.

Routing delay per hop—With every hop that a message or packet takes along its route through the network, a delay might be introduced. This delay is typically very small, e.g., Cray T3E has a hop delay of 1 network cycle [10]. For this reason, the hop delay is neglected for simplicity in edge scheduling, but it can be included in a straightforward manner, if necessary.

Routing algorithm—The routing algorithm of the network selects the links through which a communication is routed. In order to obtain the most accurate results possible, contention-aware scheduling ought to reflect the routing algorithm of the target system. Thus, it is proposed to employ the target system's own routing algorithm in contention-aware scheduling. The job of such a routing algorithm applied to the topology graph is to return an ordered list of links, the route, utilized by the corresponding interprocessor communication.

Fortunately, using the target system's own routing algorithm does not imply that, for every target system, a different routing algorithm must be implemented in scheduling. Most parallel computers employ **minimal routing**, which means they choose the shortest possible path, in terms of number of edges, through the network for every communication. Given the graph-based representation of the network, finding a shortest path can be accomplished with a Breadth First Search (BFS) algorithm [8]. Thus, the BFS can be used as a generic routing algorithm in the topology graph, which serves, at least, as a good approximation in many cases.

Although BFS is an algorithm for directed and undirected graphs, it can readily be applied to the topology graph. The only graph concept used in BFS is that of adjacency. For a hyperedge, it can be defined as follows:

Definition 5 (Adjacency with Hyperedges). *Let H be a hyperedge. A vertex $u \in \mathbf{H}$ is adjacent to all vertices $v \in H \setminus u$ and all vertices $v \in H \setminus u$ are adjacent to u.*

Now, in the topology graph, the total set of all vertices adjacent to a given vertex u is the union of the adjacent sets induced by the directed edges and the hyperedges. So, with this definition of vertex adjacency, the BFS can be applied to the topology graph without any modification and returns a shortest path in terms of number of edges.

Complexity—As routing depends on the algorithm of the target parallel system, there is no general time complexity expression which is valid for all networks. On that account, the routing complexity shall be denoted generically by $O(routing)$. In regular networks, it is usually linear in the number of network vertices or even of constant time. For example, in a fully connected network, it is obviously $O(1)$, as it is in a network with one central switch (Fig. 2c). In a topology graph for a mesh network of any dimension, it is at most linear in the number of processors $O(\mathbf{P})$. Whenever it is possible to calculate the routes for a given system once and then to store them, e.g., in a table, $O(routing)$ is just the complexity of the length of the route.

The BFS used to reflect minimal routing has linear complexity for directed and undirected graphs, $O(\mathbf{V} + \mathbf{E})$ [8]. With the above definition of adjacency with hyperedges, this result extends directly to the topology graph. Hence, in the topology graph, BFS's complexity is $O(\mathbf{N} + \mathbf{L})$. To obtain this complexity, it is important that every hyperedge is only considered once and not one time for every incident vertex.

4.3 Scheduling Network Model

Based on the previous analysis and the proposed topology graph, the network model is defined as follows:

Definition 6 (Network Model). *A communication network of a parallel system is represented by a topology graph $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$ (Definition 4). The routing algorithm carried out on the topology graph is that of the represented network (or its closest approximation). Further, routing has the following properties:*

1. ***Nonadaptive***,
2. ***Circuit switching***,
3. ***Cut-through***, *and*
4. ***No routing delay per hop***.

If the represented network employs adaptive routing, the default behavior is reproduced.

5 Edge Scheduling

In edge scheduling, communication resources are treated like processors in the sense that only one communication can be active on each resource at a time. Thus, edges are scheduled onto the communication links for the time they occupy them.

Let $G = (\mathbf{V}, \mathbf{E}, w, c)$ be a DAG and $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$ be a communication network according to Definition 6. Then, $t_s(e, L)$ denotes the **start time** of edge $e \in \mathbf{E}$ on link $L \in \mathbf{L}$ ($\mathbf{L} = \mathbf{D} \cup \mathbf{H}$). The **communication time** of e on L is $\varsigma(e, L) = \frac{c(e)}{b(L)}$ and the **finish time** of e on L is $t_f(e, L) = t_s(e, L) + \varsigma(e, L)$.

As links might have heterogenous data rates (Definition 4), the communication time is a function of the edge e and the link L. Accordingly, the edge cost $c(e)$ is now interpreted as the *average* time the communication represented by e occupies a link of $\mathbf{L}$ for its transfer. Like a processor (Condition 1), a link is exclusively occupied by one edge.

Condition 4 (Link Constraint). *For any two edges $e, f \in \mathbf{E}$ scheduled on link $L \in \mathbf{L}$,*

$$t_f(e, L) \leq t_s(f, L) \text{ or } t_f(f, L) \leq t_s(e, L). \tag{9}$$

Despite its similarity with task scheduling, edge scheduling differs in one important aspect. In general, a communication route between two processors consists of more than one link. An edge is scheduled on each link of the route, while a node is only scheduled on one processor (with the exception of the node duplication technique [18]).

5.1 Scheduling Edge on Route

Let $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$ be a communication network. For any two distinct processors P_{src} and P_{dst}, $P_{src}, P_{dst} \in \mathbf{P}$, the routing algorithm of TG returns a route $R \in TG$ from P_{src} to P_{dst} in the form of an ordered list of links $R = \langle L_1, L_2, \ldots, L_l \rangle$, $L_i \in \mathbf{L}$ for $i = 1, \ldots, l$.

Note that the route only depends on the source and the destination processor of a communication due to the network model's property of nonadaptive routing (Definition 6). As the network model also supposes circuit switching, the entire communication is transmitted on the established route. An edge using this route is scheduled on each of its links, whereby data traverses the links in the order of the route.

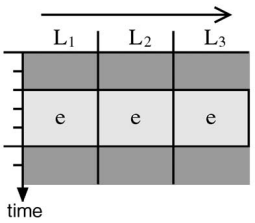


Fig. 4. Edge scheduling on a route without contention.

Condition 4 (Causality). *Let $G = (\mathbf{V}, \mathbf{E}, w, c)$ be a DAG. For the start and finish times of edge $e \in \mathbf{E}$ on the links of the route $R = \langle L_1, L_2, \ldots, L_l \rangle$, $R \in TG$,*

$$t_f(e, L_{k-1}) \leq t_f(e, L_k) \tag{10}$$

$$t_s(e, L_1) \leq t_s(e, L_k) \tag{11}$$

for $1 < k \leq l$.

Inequality (10) of the causality condition reflects the fact that communication does not finish earlier on a link than on the link's predecessor. For homogenous links, (11) is implicitly fulfilled by (10), but, for heterogenous links, this condition is important, as will be seen below.

The edge scheduling must further comply with the other routing properties of the network model in Definition 6, namely, the cut-through routing and the neglect of hop delays. Together, these two properties signify that all links of the route are utilized simultaneously. Hence, an edge must be scheduled at the same time on every used link (Fig. 4).

Edge scheduling has to be more sophisticated when contention comes into play, i.e., when a communication meets other communications along the route. Two different approaches must be examined for the determination of the scheduling times on a route with contention. Consider the same example as in Fig. 4, with the modification that link L_2 is occupied with the communication of another edge, say e_{xy}, at the time e is scheduled.

5.1.1 Nonaligned Approach

One solution to the scheduling of e is depicted in Fig. 5a. On link L_2, e is delayed until the link is available, thus, $t_s(e, L_2) = t_f(e_{xy}, L_2)$. To adhere to the causality condition, e is also delayed on link L_3; it cannot finish on the last edge until it has finished on the previous edge, i.e., $t_s(e, L_3) = t_s(e, L_2)$ and $t_f(e, L_3) = t_f(e, L_2)$. On L_1, e is scheduled at the same time as without contention (Fig. 4). This approach shall be referred to as nonaligned.

5.1.2 Aligned Approach

Alternatively, e can be scheduled later on all links, i.e., it starts on all links after edge e_{xy} finishes on link L_2, $t_s(e, L_1) = t_s(e, L_2) = t_s(e, L_3) = t_f(e_{xy}, L_2)$, even on the first L_1. At first, this aligned scheduling of the edges, illustrated in Fig. 5b, seems to better reflect cut-through routing, where communication takes place on all involved links at the same

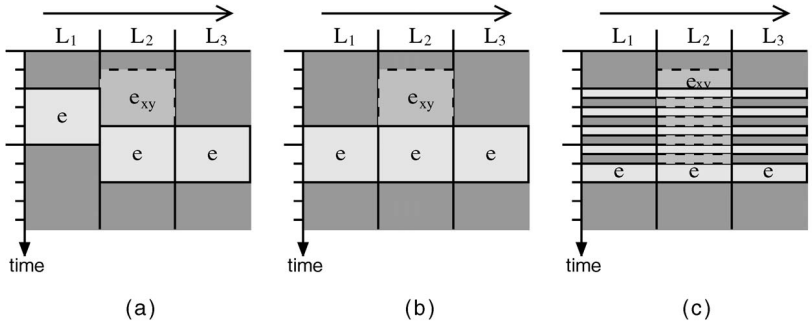


Fig. 5. Edge scheduling on a route with contention. (a) Nonaligned. (b) Aligned. (c) Packet view.

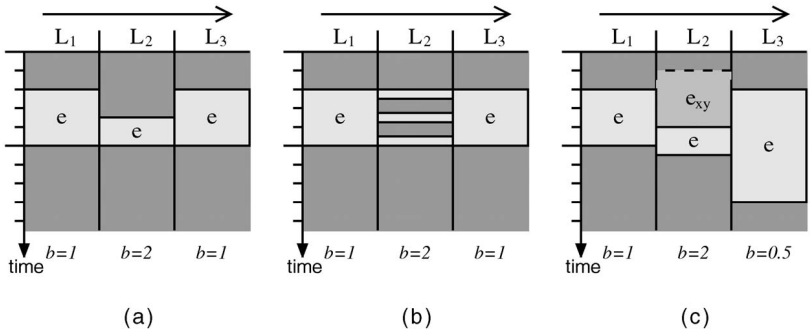


Fig. 6. Edge scheduling on a heterogenous route. (a) Without contention. (b) Packet view. (c) Contention.

time. Scheduling the edge at different times, as done in the first approach (Fig. 5a), seems to imply the storage of at least parts of the communication at the network vertices, i.e., a behavior similar to store-and-forward routing.

5.1.3 Packet View

However, communication in parallel systems is packet-based and the real process of the message transfer is better illustrated by the Gantt chart of Fig. 5c. Each of the symbolized packets performs cut-through routing and link L_2 is shared among both edges.

Even though edge scheduling does not reflect the division into packets, the packet view of the situation in Fig. 5c illustrates that the nonaligned scheduling (Fig. 5a) can be interpreted as a message-based view of the situation in Fig. 5c. Nonaligned scheduling holds an accurate view of the communication time spent in the network. The communication lasts from the start of edge e on the first link L_1 until it finishes on link L_3, exactly the same time interval as in the packet view of Fig. 5c. In contrast, aligned scheduling delays the start of the e on the first link L_1 and, therefore, does not reflect the same delay within the network as in the packet view. In both approaches, the total occupation time of the links is identical to the packet view, even though it is approximated as an indivisible communication. Moreover, scheduling the edges aligned on a route is more difficult to implement and likely to produce many idle time slots on the links.

In conclusion, the nonaligned approach appears to better reflect the behavior of communication networks and its utilization in edge scheduling is proposed. This is also true for heterogenous links, as shown in the following.

5.1.4 Heterogenous Links

Fig. 6a illustrates again the scheduling of an edge on three links, whereby link L_2 is now twice as fast as the other links ($b(L_2) = 2b(L_1) = 2b(L_3)$). Due to the causality condition, e cannot finish earlier on L_2 than on L_1 and, therefore, the start of e on L_2 is delayed accordingly. As above, this delay seems to suggest a storage between the first and the second link, but, in fact, it is the best approximation of the real, packet-based communication illustrated in Fig. 6b. This also explains why e starts and finishes on L_3 at the same times as on L_1, which is realistic since a faster link on the route obviously should not retard the communication. Thus, the causality condition ensures a realistic scheduling of the edge in this example.

The next example demonstrates the importance of (11) of the causality condition for heterogeneous links. Consider the same example as before, now with the last link slower than the two others ($b(L_3) = b(L_1)/2 = b(L_2)/4$), illustrated in Fig. 6c. Without (11), e could start earlier on L_3 than on L_1, it only had to finish not earlier than on L_2. Evidently, the principle of causality would be violated. Due to the lower data rate of L_3, the correct behavior, as shown in Fig. 6c, is not implicitly enforced by (10) as in all other examples. Notice that the communication is not delayed by the contention on L_2, because even without it, e could not start earlier on L_3. This is a realistic reflection of the behavior of a real network, where a fast link (L_2) can compensate contention in relation to slower links on the route.

5.2 The Edge Scheduling

The nonaligned approach allows the determination of the start and finish time successively for each link on the route. On the first link, the edge's scheduling is only constrained

by the finish time of its origin node on the source processor of the communication. On all subsequent links, the edge has to comply with the causality condition. As for the scheduling of the nodes (Condition 3), a scheduling condition can be formulated that integrates both the insertion and end technique (Section 2.2).

Condition 6 (Edge Scheduling Condition). *Let $G = (\mathbf{V}, \mathbf{E}, w, c)$ be a DAG, $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$ be a communication network, and $R = \langle L_1, L_2, \ldots, L_l \rangle$ be the route for the communication of edge $e_{ij} \in \mathbf{E}$, $n_i, n_j \in \mathbf{V}$. Let $[A, B]$, $A, B \in [0, \infty]$, be an idle time interval on L_k, $1 \leq k \leq l$, i.e., an interval in which no other edge is transferred on L_k. Edge e_{ij} can be scheduled on L_k within $[A, B]$ if*

$$B - A \geq \varsigma(e_{ij}, L_k) \tag{12}$$

$$B \geq \begin{cases} t_f(n_i) + \varsigma(e_{ij}, L_k) & \text{if } k = 1 \\ \max\{t_f(e_{ij}, L_{k-1}), t_s(e_{ij}, L_1) + \varsigma(e_{ij}, L_k)\} & \text{if } k > 1. \end{cases} \tag{13}$$

Condition 6 ensures that the time interval is large enough for e_{ij} ((12)). On the first link ($k = 1$), e_{ij}'s scheduling is only constrained by the finish time of its origin node n_i on the source processor of the communication. On all subsequent links ($k > 1$), the scheduling of e_{ij} has to comply with the causality condition (Condition 5).

A time interval, obeying the above condition, can be searched successively for each of the links on the route, using either the end or the insertion technique. For a given idle time interval $[A, B]$ on L_k, adhering to Condition 6, the start time of e_{ij} on L_k is determined as

$$t_s(e_{ij}, L_k) = \begin{cases} \max\{A, t_f(n_i)\} & \text{if } k = 1 \\ \max\{A, t_f(e_{ij}, L_{k-1}) - \varsigma(e_{ij}, L_k), t_s(e_{ij}, L_1)\} & \text{if } k > 1. \end{cases} \tag{14}$$

So, edge e_{ij} is scheduled as early within the idle interval as the causality condition or the finish time of the origin node admits. This corresponds exactly to the nonaligned approach discussed in Section 5.1.

6 Contention Aware Scheduling

First of all, the new, more realistic target system model for contention-aware task scheduling is defined:

Definition 7 (Target Parallel System—Contention Model). *A target parallel system $M_{TG} = (TG, \omega)$ consists of a set of possibly heterogenous processors $\mathbf{P}$ connected by the communication network $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$, according to Definition 6. The processor heterogeneity, in terms of processing speed, is described by the execution time function ω. This system has the Properties 1 to 3 of the classic model of Definition 2.*

Two of the classic model's properties are substituted by the detailed network model: concurrent communication (Property 4) and a fully connected network topology (Property 5). With the abandonment of the assumption of concurrent communication, task scheduling must consider the contention for communication resources. For this purpose, edge scheduling is employed. The integration of edge scheduling into task scheduling is relatively easy, due to the careful formulation of the scheduling problem in Section 2.1. It is only necessary to redefine the edge finish time (Definition 3).

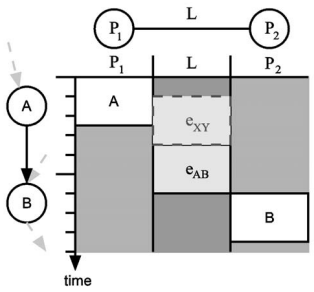


Fig. 7. Contention awareness achieved with edge scheduling.

Definition 8 (Edge Finish Time—Contention Model). *Let $G = (\mathbf{V}, \mathbf{E}, w, c)$ be a DAG and $M_{TG} = ((\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b), \omega)$ a parallel system. Let $R = \langle L_1, L_2, \ldots, L_l \rangle$ be the route for the communication of $e_{ij} \in \mathbf{E}$, $n_i, n_j \in \mathbf{V}$, if $proc(n_i) \neq proc(n_j)$. The finish time of e_{ij} is*

$$t_f(e_{ij}) = \begin{cases} t_f(n_i) & \text{if } proc(n_i) = proc(n_j) \\ t_f(e_{ij}, L_l) & \text{otherwise.} \end{cases} \tag{15}$$

So, the edge's (global) finish time is its finish time on the last link on the route, unless the communication occurs between nodes scheduled on the same processor (Property 2), where it remains the finish time of the origin node. The scheduling of the nodes is not concerned with the internals of edge scheduling. Everything that is relevant to node scheduling is encapsulated in the finish time of an edge.

Task scheduling becomes contention-aware, since the finish time of an edge depends on the contention for the communication links. This is illustrated in Fig. 7, where edge e_{AB} and, in turn, node B are delayed due to contention on the link L. A scheduling algorithm "sees" the contention through the later DRT's of the nodes.

Without contention, the new model behaves like the classic model, i.e., the time the edge spends in the network corresponds exactly to the communication delay in the classic model. Therefore, the edge weight in a DAG for the contention model can remain identical to the edge weight in a DAG for the classic model. Hence, the new system model has no impact on the creation of the DAG.

Scheduling under the new contention model remains an NP-hard problem.

Theorem 1 (NP-Completeness—Contention Model). *Let $G = (\mathbf{V}, \mathbf{E}, w, c)$ be a DAG and $M_{TG} = ((\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b), \omega)$ a parallel system. The decision problem C-SCHED(G, M_{TG}), associated with the scheduling problem is as follows: Is there a schedule $\mathcal{S}$ for G on M_{TG} with length $sl(\mathcal{S}) \leq T, T \in \mathbb{Q}^+$? C-SCHED($G$, M_{TG}) is NP-complete.*

Proof. This proof is the adapted proof of the NP-completeness of scheduling under the one-port model [4]. First, it is argued that C-SCHED belongs to NP, then, it is shown that C-SCHED is NP-hard by reducing the well-known

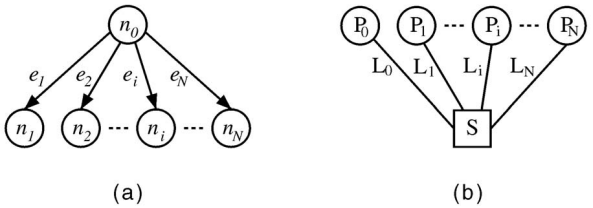


Fig. 8. NP-completeness proof: (a) the fork DAG and (b) the target system.

NP-complete problem 2-PARTITION [12] in polynomial time to C-SCHED. The 2-PARTITION problem is: Given l positive integer numbers $\{a_1, a_2, \ldots, a_l\}$, is there a subset $\mathbf{I}$ of indices such that $\sum_{i\in\mathbf{I}} a_i = \sum_{i\notin\mathbf{I}} a_i$?

Clearly, for any given solution $\mathcal{S}$ of C-SCHED, it can be verified in polynomial time that $\mathcal{S}$ is feasible and $sl(\mathcal{S}) \leq T$, hence, C-SCHED $\in$ NP.

From an arbitrary instance of 2-PARTITION $\{a_1, a_2, \ldots, a_l\}$, an instance of C-SCHED is constructed in the following way: Let $M = \max_{1\leq i\leq l} a_i$, $m = \min_{1\leq i\leq l} a_i$, and $2U = \sum_{i=1}^{l} a_i$ (if the sum is odd, there is no solution to the instance of 2-PARTITION).

DAG G—The constructed DAG G is a fork-graph as illustrated in Fig. 8a. It consists of one parent node n_0 and N child nodes $n_1, n_2, \ldots, n_N$, hence, $|\mathbf{V}| = N + 1$ nodes, where $N = l + 3$. There is an edge e_i directed from n_0 to every child node $n_i, 1 \leq i \leq N$. The weights assigned to the nodes are

- $w(n_0) = 1$ and $w(n_i) = 10(M + a_i + 1)$ for $1 \leq i \leq l$.
- $w(n_{l+1}) = w(n_{l+2}) = w(n_{l+3}) = w_{min}$, with $w_{min} = 10(M + m) + 1$. Hence, to the last three children, the same minimal weight is assigned. Note that $w_{min} < w(n_i) < 2w_{min}$ for $1 \leq i \leq l$, as can be verified straightforwardly.

The weight of each edge equals the weight of its destination node: $c(e_i) = w(n_i)$ for $1 \leq i \leq l + 3$.

Target system M_{TG}—The constructed target system $M_{TG} = ((\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b), \omega)$, illustrated in Fig. 8b, consists of $|\mathbf{P}| = N + 1$ identical processors (i.e., $\omega(n, P) = w(n)$), i.e., the number of processors is practically unlimited, as more processors than nodes can never be used. Each processor P_i is connected to a central switch S through a half duplex link L_i, which is represented by an undirected edge (a hyperedge incident on two vertices: $L_i = H_i = \{P_i, S\}$). Hence, $\mathbf{N} = S \cup \mathbf{P}$, $\mathbf{D} = \emptyset$, $\mathbf{H} = \{L_0, L_1, \ldots, L_N\}$, and, as all links are identical, $b(L_i) = 1$ for $0 \leq i \leq N$.

Time bound T—The time bound is set to

$$T = \frac{1}{2}\sum_{i=1}^{l} w(n_i) + 2w_{min} + w(n_0)$$
$$= 5n(M + 1) + 10U + 20(M + m) + 3.$$

Clearly, the construction of the instance of C-SCHED is polynomial in the size of the instance of 2-PARTITION.

It is now shown how a schedule is derived for C-SCHED from an instance $\{a_1, a_2, \ldots, a_l\}$ that admits a solution to 2-PARTITION: Let $\mathbf{I}$ be a subset of indices such that $U = \sum_{i\in\mathbf{I}} a_i = \sum_{i\notin\mathbf{I}} a_i$:

- Node n_0, all nodes n_i, $i \in \mathbf{I}$, and n_{l+1} and n_{l+2} are allocated to processor P_0. Obviously, P_0's finish time is exactly $t_f(P_0) = T$. Each other node is assigned to a distinct processor other than P_0.
- The outgoing edges e_i from P_0 are scheduled on L_0 in increasing index order; in particular, e_{l+3} is the last edge scheduled on L_0.
- The processor, say P_j, on which n_{l+3} is executed, receives e_{l+3} at time step

$$w(n_0) + \sum_{i\notin\mathbf{I}} c(e_i) + c(e_{l+3}) = w(n_0) + \sum_{i\notin\mathbf{I}} w(e_i) + w_{min}$$

  since e_{l+3} is the only edge on L_j, hence it finishes on L_j at the same time as on L_0. Processor P_j finishes execution at time step $t_f(P_j) = w(n_0) + \sum_{i\notin\mathbf{I}} w(e_i) + 2w_{min} = T$.
- All other processors terminate their execution earlier because they receive their communication not later than $w(n_0) + \sum_{i\notin\mathbf{I}} w(e_i)$ and their execution time $w(n_i)$ is smaller than $2w_{min}$.

Thus, a feasible schedule was derived whose schedule length matches the time bound T, hence it is a solution to the constructed C-SCHED instance.

Conversely, assume that the instance of C-SCHED admits a solution given by the feasible schedule $\mathcal{S}$ with $sl(\mathcal{S}) \leq T$. The processor to which n_0 is scheduled shall be referred to as P_0. Let $\mathbf{J} = \{i, 1 \leq i \leq l + 3 : proc(n_i) = P_0\}$ be the index set of all other nodes, besides n_0, scheduled on P_0. Thus, P_0 terminates execution not earlier than $A = w(n_0) + \sum_{i\in\mathbf{J}} w(n_i)$. All nodes $n_i \notin \mathbf{J}$ are scheduled on processors other than P_0. Let n_{last}, $proc(n_{last}) \neq P_0$, be the node receiving the last communication from P_0. The finish time of the processor executing n_{last} is not earlier than $B = w(n_0) + \sum_{1\leq i\leq l+3, i\notin\mathbf{J}} c(e_i) + w_{last}$. Since $sl(\mathcal{S}) \leq T$, $\max(A, B) \leq T$. As

$$A + B = 2w(n_0) + \sum_{1\leq i\leq l+3} w(n_i) + w_{last} = 2T + w_{last} - w_{min},$$

$w_{last} = w_{min}$ and, hence, $A = B = T$. Now, since $A = B$, also $A \equiv B \bmod 10$, thus—due to the choice of the node and edge weights—exactly two nodes with indices from $\{l + 1, l + 2, l + 3\}$ must be executed on P_0, i.e., these indices are in $\mathbf{J}$. To conclude, let $\mathbf{I}$ be equal to $\mathbf{J}$ minus these two indices to obtain a solution to 2-PARTITION. □

This proof showed weak NP-completeness [12] of C-SCHED. It seems possible to prove strong NP-completeness by reducing from 3-PARTITION [12], and we are working on this proof.

6.1 List Scheduling

In order to adapt list scheduling to the contention model, two of its aspects must be modified: the determination of the DRT and the scheduling of a node. Having a contention-aware list scheduling is crucial, due to list scheduling's significant role in task scheduling and allows to transform many of the existing heuristics into contention-aware algorithms. Moreover, the following discussion also serves as an example on how to adapt a scheduling algorithm to the new contention model. For example, in [28], the BSA

algorithm was easily adapted to the contention-aware scheduling proposed in this document.

Under the contention model, the scheduling of a node must also include the scheduling of its incoming edges on the communication links. Thus, line 6 of Algorithm 1 must be performed as outlined in Algorithm 2. For each entering edge $e_{ij} \in \mathbf{pred}(n_j)$ of n_j, the route is determined and e_{ij} is scheduled on it as described in Section 5.2. Only after that, when the finish times of all entering edges of n_j are known, node n_j's DRT can be determined and it can be scheduled on the processor.

Algorithm 2. Scheduling of a free node n_j on processor P in contention model.

for each $n_i \in \mathbf{pred}(n_j)$ in a definite order **do**
  if $proc(n_i) \neq P$ **then**
    determine route $R = \langle L_1, L_2, \ldots, L_l \rangle$ from $proc(n_i)$ to P
    schedule e_{ij} on R
schedule n_j on P

To find the processor that allows the earliest finish time of a node, this procedure must be performed tentatively for every processor (line 5 of Algorithm 1). So, after determining the finish time of n_j on a processor, the incoming edges are removed from the links before the procedure is repeated on the next processor. This is necessary to obtain the correct view of the communication times of the edges.

The scheduling order of the edges is relevant for the DRT's of their destination nodes. For this reason, Algorithm 2 requires a definite processing order of the edges (`for`-loop). For example, the edges might be processed in the order of their indices. Of course, an attempt can be undertaken to find an order that positively influences the DRTs of the destination nodes and, thereby, the schedule length. For example, the edges can be ordered according to the start time of their origin nodes or by the size of the communication.

The complexity of list scheduling under the contention model increases slightly compared to the classic model (Section 2.2.1), owing to the different calculation of the DRT. The calculation of the DRT involves now the routing and scheduling of an edge which is $O(routing)$ (Section 4.2). Thus, the complexity of the second part of contention-aware list scheduling is $O(\mathbf{P}(\mathbf{V} + \mathbf{E}O(routing)))$ with the end technique. Using the insertion technique has a complexity of $O(\mathbf{V}^2 + \mathbf{PE}^2O(routing))$ as the calculation of an edge's start time changes from $O(1)$ to $O(\mathbf{E})$, since there are at most $O(\mathbf{E})$ edges on each link.

Various node priority schemes for the first part of contention-aware list scheduling are analyzed and compared in [30], among which the **bottom level** provides the best results.

7 Experimental Results

This section presents experiments that analyze many aspects of the proposed contention-aware scheduling. The objective is to investigate the accuracy and the efficiency of the new model compared to the classic model. For this purpose, an experimental evaluation on real parallel systems is required. Such an evaluation was extensively performed in [28] and this section presents the major results.

7.1 Methodology

The experimental methodology [31] begins with the common procedure of scheduling algorithm comparisons: A large set of DAGs is scheduled by different heuristics on various target systems. In the next step, code is generated that corresponds to the DAGs and their schedules. This code is then executed on real parallel systems. Let $pt(\mathcal{S})$ be the execution time—the parallel time—of the code generated for schedule $\mathcal{S}$ and DAG $G = (\mathbf{V}, \mathbf{E}, w, c)$ on a target system. The **accuracy** of $\mathcal{S}$ is the ratio of the real execution time on the parallel system to the schedule length, i.e., the estimated execution time $acc(\mathcal{S}) = \frac{pt(\mathcal{S})}{sl(\mathcal{S})}$. The speedup of $\mathcal{S}$ is the ratio of sequential time to execution time $sup(\mathcal{S}) = \frac{seq(G)}{pt(\mathcal{S})}$. Finally, the efficiency of $\mathcal{S}$ is the ratio of speedup to processor number $eff(\mathcal{S}) = \frac{sup(\mathcal{S})}{|\mathbf{P}|} = \frac{seq(G)}{pt(\mathcal{S}) \times |\mathbf{P}|}$.

7.1.1 Workload

The workload for the experiments are randomly generated graphs, as are common for analyzing scheduling algorithms, e.g., [3], [13], [20]. Three parameters are varied to generate large sets of random graphs: the number of nodes (v), the average number of edges per node, i.e., the average in-degree or the average out-degree (they are identical) of a node ($\bar{\delta}$), and the communication to computation ratio ($CCR = \frac{\sum_{e \in \mathbf{E}} c(e)}{\sum_{n \in \mathbf{V}} w(n)}$).

7.1.2 Scheduling Algorithms

For the analysis of the classic and the contention model, one scheduling algorithm is considered under each model. For both models, list scheduling using the end technique is employed. Under the classic model, the nodes are ordered according to their bottom levels ((7)). For the contention model, the node priority scheme of the contention-aware DLS algorithm [27] (Section 3.2) was chosen. The priority of a node n_j is given by $bl_w(n_j) + \max_{e_{ij} \in \mathbf{E}, n_i \in \mathbf{pred}(n_j)} c(e_{ij})$, where bl_w is the bottom level *excluding* the edge weights, i.e., communication costs. Later experiments showed that the bottom-level produces better results also under the contention model [30].

For evaluating the scheduling accuracy, the actual performance of the algorithms, in terms of produced schedule lengths, is irrelevant. Scheduling is accurate if the schedule length is a good approximation of the execution time.

7.1.3 Code Generation

Code is generated using the C language and MPI (Message Passing Interface [24]). The code generated for each node is an active wait for the time corresponding to its weight. Each remotely transferred edge translates into a send command on the origin processor and a receive command on the destination processor. The amount of data to be transferred, which depends on the communication capabilities of the target system, corresponds to the weight $c(e)$ of the corresponding edge.

TABLE 1
Characteristics of the Employed Target Systems

	total processors	network	topology model	employed processors
BOBCAT	16	switched LAN	fully connected	8, 16
Sun E3500	8	bus	bus	4, 7
Cray T3E	344	3D cyclic torus	fully con./3D cyclic torus	32

7.1.4 Target Systems

The three employed target systems cover a wide spectrum of parallel architectures (Table 1): **BOBCAT**—a PC cluster [32], **Sun E3500** (Sun Enterprise 3500)—a shared memory SMP system [10], and **T3E** (Cray T3E)—a massively parallel system [7].

In the contention-aware algorithm, the topologies of the target systems are modeled as follows: BOBCAT is represented by a fully connected network because the utilized Ethernet switch can handle communications conflict free. As Sun E3500's communication network is essentially a bus, it is presented by one hyperedge. The T3E is modeled as fully connected and, alternatively, as a 3D cyclic torus. The decision to initially model the T3E as fully connected, even though its network is a 3D cyclic torus, is based on the circumstance that it is not known at compile how a parallel program is mapped onto T3E's processors. For the purpose of comparison, the T3E is represented as a 3D cyclic torus in additional experiments.

The routing algorithm of the target systems is reflected by a shortest-path algorithm in the topology graph (Section 4.2).

7.2 Results

To summarize, a large set of randomly generated DAGs ($v = 600, 800, 1,000$; $CCR = 0.1$, 1 and 10; $\bar{\delta} = 1.5$ and 3) is scheduled by an algorithm under the classic and under the contention model onto various configurations of the target systems: eight and 16 processors of BOBCAT, four and seven processors of Sun E3500, and 32 processors of T3E. For each of these schedules, code is generated and executed on the respective target system. The charts of Fig. 9 show the average accuracy and efficiency achieved under the two models over all DAGs on all configurations of the systems.

Accuracy—For low communication ($CCR = 0.1$), the accuracy difference between both models is negligible (Fig. 9a). This makes sense, as a small CCR means that communication is less important than computation. Thus, the effect of contention is almost negligible and the awareness of contention cannot improve the accuracy. With more communication, the accuracy becomes worse for both algorithms. However, the algorithm under the contention model now achieves much more accurate results for medium ($CCR = 1$) and, especially, for high communication ($CCR = 10$) than the algorithm under the classic model. In these cases, contention plays a significant role.

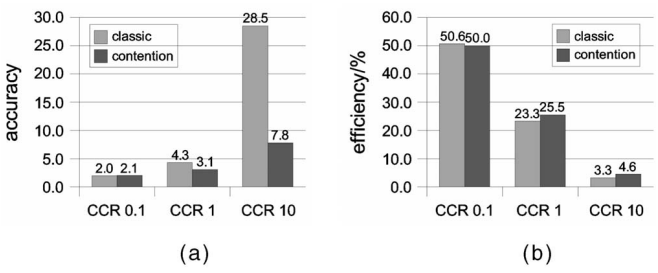


Fig. 9. (a) Average accuracy and (b) efficiency under the two scheduling models over all DAGs and various configurations of the three target systems.

Efficiency—It is intuitive that the accuracy of the schedules has a direct influence on their execution times, which is represented in a normalized form by the efficiency in Fig. 9b. Essentially, the efficiency confirms the results regarding the accuracy. Indeed, the contention-aware scheduling algorithm achieves better efficiency for medium ($CCR = 1$)—improvement of about 10 percent—and high communication ($CCR = 10$)—improvement of about 40 percent—than the one under the classic model. Unfortunately, the speedup of the schedules for $CCR = 10$ is often below one, but recall that the efficiency under the contention model suffers from the worse node priority (the contention-aware algorithm does not use the bottom-level scheme (Section 7.1.2)), thus, the results might improve further with a better node priority.

When the accuracy and efficiency results are separated according to the target system, it is revealed that the most accurate and efficient schedules are obtained for Sun E3500 (Figs. 10a and 10b). The accuracy is much better, compared to the average results for all target systems in Fig. 9, for medium ($CCR = 1$) and high communication ($CCR = 10$). Also, the efficiency improvement over the classic model is higher: on Sun E3500 it is about 11 percent for medium and about 53 percent for high communication.

The importance of the choice of the topology model is evidenced by the comparison of the two different topology graphs for the T3E (fully connected versus 3D cyclic torus) in contention scheduling (Fig. 10c). The more accurate model—the 3D cyclic torus—yields a much better prediction of the real execution times. The efficiency does not improve using the torus graph, which is comprehensible given that the processors might be mapped to completely different processors as assumed during scheduling. Hence, the contention-aware algorithm captures the importance of contention through the torus graph, but the produced schedules are obviously inappropriate when the mapping of the tasks onto the processors is different as decided by the scheduling algorithm.

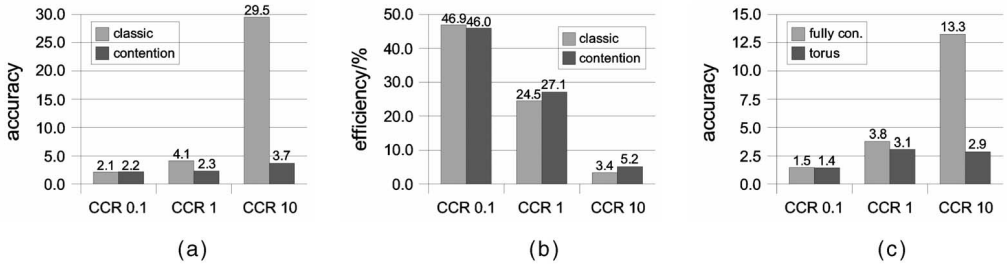


Fig. 10. (a) Average accuracy and (b) efficiency under the two scheduling models over all DAGs on the configurations of Sun E3500; (c) comparison of accuracy on T3E using fully connected and 3D cyclic torus model.

7.3 Discussion

Contention aware scheduling significantly improves both the accuracy and the efficiency of the produced schedules. Still, the results are improvable.

In part, this might be due to the topology graphs employed in some of the experiments. For example, modeling a system as fully connected allows a high degree of concurrent communication. However, an analysis of the target systems shows that, in all of them, each processor is connected to the network through an interface, allowing at most one incoming and one outgoing communication at a time [10]. In other words, end-point contention is not sufficiently considered when using the fully connected graph. This explanation is confirmed by the better results for Sun E3500. The employed bus model serializes all communications, thereby implicitly considering end-point contention. Probably, the results for BOBCAT and T3E can be further improved by using a switch vertex to model the interface bottleneck (Section 4.1).

Another possible explanation for the still improvable results is that communication contention is not the only aspect which is inaccurately reflected in the classic model. Even for the contention model, a relation between the increase of communication (CCR) and the degradation of accuracy can be observed in the presented experimental results. This indicates another deficiency of the scheduling models regarding communication. Like the classic model (Definition 2), the contention model (Definition 7) supposes a dedicated communication subsystem to exist in the target system (Property 3). With the assumed subsystem, computation can overlap with communication because the processor is not involved in communication. However, the analysis of the three parallel systems shows that none of them possesses such a dedicated communication subsystem. On all three systems, the processors are involved, in one way or the other, in interprocessor communication. Consequently, to further improve the accuracy and efficiency of scheduling, the involvement of the processor should be investigated [29].

Concluding, the experimental evaluation exposed that, despite the improvable results, the new contention model significantly improves both the accuracy and the efficiency of task scheduling,

8 Conclusions

This paper was dedicated to the incorporation of contention awareness into task scheduling. The ideal system model of classic task scheduling does not capture any kind of contention for communication resources. Therefore, a new system model was proposed that abandons the assumptions of a fully connected network and concurrent communication. Instead, the topology of the network is represented by a new graph model that allows us to capture both end-point and network contention. Contention awareness is achieved by scheduling the edges of the DAG onto the links of the topology graph. Edge scheduling was theoretically analyzed, including aspects like heterogeneity, routing, and causality. Based on the network model and the edge scheduling technique, task scheduling was enhanced with contention awareness. This approach preserves the theoretical basis of task scheduling and has a very small impact on the complexity of scheduling algorithms. It was shown how classic list scheduling is easily extended to be contention-aware.

Experimental results demonstrated the significantly improved accuracy and efficiency of the schedules. The results also suggest that the involvement of the processors in communication should be investigated in order to further improve task scheduling's accuracy and efficiency.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This work was partially supported by Portuguese Foundation for Science and Technology (FCT) through Program FEDER, by FCT and FSE (Fundo Social Europeu) in the context of III European Framework of Support, and by the European Commission through ARI grant HPRI-CT-1999-00026 and TMR grant ERB FMGE CT950051 (TRACS Programme at EPCC), which the authors gratefully acknowledge.

References

[1] T.L. Adam, K.M. Chandy, and J.R. Dickson, "A Comparison of List Schedules for Parallel Processing Systems," *Comm. ACM*, vol. 17, pp. 685-689, 1974.

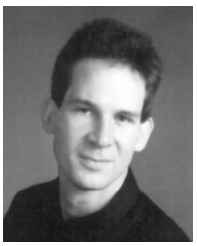
[2] I. Ahmad and Y.-K. Kwok, "On Exploiting Task Duplication in Parallel Program Scheduling," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 8, pp. 872-892, Sept. 1998.

[3] I. Ahmad, Y.-K. Kwok, and M.-Y. Wu, "Analysis, Evaluation, and Comparison of Algorithms for Scheduling Task Graphs on Parallel Processors," *Proc. Second Int'l Symp. Parallel Architectures, Algorithms, and Networks*, G.-J. Li, D.F. Hsu, S. Horiguchi, and B. Maggs, eds., pp. 207-213, June 1996.

[4] O. Beaumont, V. Boudet, and Y. Robert, "A Realistic Model and an Efficient Heuristic for Scheduling with Heterogeneous Processors," *Proc. 11th Heterogeneous Computing Workshop*, 2002.

[5] C. Berge, *Graphs and Hypergraphs*, second ed. North-Holland, 1976.

[6] C. Boeres and V.E.F. Rebello, "A Versatile Cost Modelling Approach for Multicomputer Task Scheduling," *Parallel Computing,* vol. 25, no. 1, pp. 63-86, Jan. 1999.
[7] S. Booth, J. Fisher, and M. Bowers, "Introduction to the Cray T3E at EPCC," *Edinburgh Parallel Computing Centre,* http://www.epcc.ed.ac.uk/t3d/documents/t3e-intro.html, June 1999.
[8] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms.* MIT Press, 1990.
[9] D.E. Culler, R.M. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation," *ACM SIG-PLAN Notices, Proc. Symp. Principles and Practice of Parallel Programming,* vol. 28, no. 7, pp. 1-12, July 1993.
[10] D.E. Culler and J.P. Singh, *Parallel Computer Architecture.* Morgan Kaufmann Publishers, 1999.
[11] H. El-Rewini and T.G. Lewis, "Scheduling Parallel Program Tasks onto Arbitray Target Machines," *J. Parallel and Distributed Computing,* vol. 9, no. 2, pp. 138-153, June 1990.
[12] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.
[13] A. Gerasoulis and T. Yang, "A Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessors," *J. Parallel and Distributed Computing,* vol. 16, no. 4, pp. 276-291, Dec. 1992.
[14] J.J. Hwang, Y.C. Chow, F.D. Anger, and C.Y. Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times," *SIAM J. Computing,* vol. 18, no. 2, pp. 244-257, Apr. 1989.
[15] T. Kalinowski, I. Kort, and D. Trystram, "List Scheduling of General Task Graphs under LogP," *Parallel Computing,* vol. 26, pp. 1109-1128, 2000.
[16] H. Kasahara and S. Narita, "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing," *IEEE Trans. Computers,* vol. 33, pp. 1023-1029, Nov. 1984.
[17] B. Kruatrachue, "Static Task Scheduling and Grain Packing in Parallel Processing Systems," PhD thesis, Oregon State Univ., 1987.
[18] B. Kruatrachue and T. Lewis, "Grain Size Determination for Parallel Processing," *IEEE Software,* vol. 5, no. 1, pp. 23-32, Jan. 1988.
[19] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing—Design and Analysis of Algorithms.* Benjamin/Cummings, 1994.
[20] Y.-K. Kwok and I. Ahmad, "Benchmarking the Task Graph Scheduling Algorithms," *Proc. Int'l Parallel Processing Symp./Symp. on Parallel and Distributed Processing,* pp. 531-537, Apr. 1998.
[21] Y.-K. Kwok and I. Ahmad, "Link Contention-Constrained Scheduling and Mapping of Tasks and Messages to a Network of Heterogeneous Processors," *Cluster Computing: J. Networks, Software Tools, and Applications,* vol. 3, no. 2, pp. 113-124, 2000.
[22] W. Löwe, W. Zimmermann, and J. Eisenberger, "On Linear Schedules of Task Graphs for Generalized LogP-Machines," *Proc. Euro-Par '97,* vol. 1300, pp. 895-904, 1997.
[23] B.S. Macey and A.Y. Zomaya, "A Performance Evaluation of CP List Scheduling Heuristics for Communication Intensive Task Graphs," *Proc. Int'l Parallel Processing Symp./Symp. Parallel and Distributed Processing,* pp. 538-541, 1998.
[24] "Message Passing Interface Forum," *MPI: A Message-Passing Interface Standard,* http://www.mpi-forum.org/docs/docs.html, June 1995.
[25] P. Rebreyend, F.E. Sandnes, and G.M. Megson, "Static Multiprocessor Task Graph Scheduling in the Genetic Paradigm: A Comparison of Genotype Representations," Research Report 98-25, Laboratoire de Informatique du Parallelisme, Ecole Normale Superieure de Lyon, 1998.
[26] V. Sarkar, *Partitionning and Scheduling Parallel Programs for Execution on Multiprocessors.* MIT Press, 1989.
[27] G.C. Sih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 2, pp. 175-186, Feb. 1993.
[28] O. Sinnen, "Experimental Evaluation of Task Scheduling Accuracy," *Tese de Mestrado,* master's thesis, Instituto Superior Técnico, Technical Univ. of Lisbon, Dec. 2001.
[29] O. Sinnen, "Accurate Task Scheduling for Parallel Systems," PhD thesis, Instituto Superior Técnico, Technical Univ. of Lisbon, Apr. 2003.
[30] O. Sinnen and L. Sousa, "List Scheduling: Extension for Contention Awareness and Evaluation of Node Priorities for Heterogeneous Cluster Architectures," *Parallel Computing,* vol. 30, no. 1, pp. 81-101, Jan. 2004.
[31] O. Sinnen and L. Sousa, "On Task Scheduling Accuracy: Evaluation Methodology and Results," *J. Supercomputing,* vol. 27, no. 2, pp. 177-194, Feb. 2004.
[32] S. Telford, "BOBCAT User Guide," *Edinburgh Parallel Computing Centre,* http://www.epcc.ed.ac.uk/sun/documents/introdoc.html, May 2000.
[33] J.D. Ullman, "NP-Complete Scheduling Problems," *J. Computing System Science,* vol. 10, pp. 384-393, 1975.
[34] M.Y. Wu and D.D. Gajski, "Hypertool: A Programming Aid for Message-Passing Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 1, no. 3, pp. 330-343, July 1990.
[35] T. Yang and A. Gerasoulis, "PYRROS: Static Scheduling and Code Generation for Message Passing Multiprocessors," *Proc. Sixth ACM Int'l Conf. Supercomputing,* pp. 428-437, Aug. 1992.
[36] T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 9, pp. 951-967, Sept. 1994.



Oliver Sinnen received three degrees in electrical and computer engineering: a diploma (equivalent to a master's) in 1997 from RWTH-Aachen, Germany, another master's degree, and the PhD degree in 2002 and 2003, respectively, both from Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal. Currently, he is working as a lecturer in the Department of Electrical and Computer Engineering at the University of Auckland, New Zealand. His research interests include parallel and distributed computing, reconfigurable computing, graph theory, and algorithm design.

Leonel A. Sousa received the PhD degree in electrical and computer engineering from the Instituto Superior Técnico (IST), Technical University of Lisbon, Portugal. He is currently working as an assistant professor in the Electrical and Computer Engineering Department at IST and as a senior researcher at INESC-ID, a research institute associated with IST. His research interests include computer architecture, parallel and distributed computing, and multimedia processors. He is a senior member of the IEEE and a member of the IEEE Computer Society.