Toward a Realistic Task Scheduling Model

Oliver Sinnen, Leonel Augusto Sousa, *Senior Member*, *IEEE*, and Frode Eika Sandnes

Abstract—Task scheduling is an important aspect of parallel programming. Most of the heuristics for this NP-hard problem are based on a very simple system model of the target parallel system. Experiments revealed the inappropriateness of this classic model to obtain accurate and efficient schedules for real systems. In order to overcome this shortcoming, a new scheduling model was proposed that considers the contention for communication resources. Even though the accuracy and efficiency improved with the consideration of contention, the new contention model is still not good enough. The crucial aspect is the involvement of the processor in communication. This paper investigates the involvement of the processor in communication and its impact on task scheduling. A new system model is proposed based on the contention model that is aware of the processor involvement. The challenges for the scheduling techniques are analyzed and two scheduling algorithms are proposed. Experiments on real parallel systems show the significantly improved accuracy and efficiency of the new model and algorithms.

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

1 Introduction

Scheduling is an important aspect of efficient parallel computer utilization. In task scheduling, the program is represented by a task graph, or Directed Acyclic Graph (DAG), where the nodes represent the tasks of the program and the edges the communications between the tasks. The scheduling problem is to find the spatial and temporal assignments of the tasks onto the processors of the target system which results in the shortest possible execution time of the program. In its general form, this problem is NP-hard [28], [35]. Many heuristics have been proposed for the near optimal solution of the problem [3], [4], [12], [14], [19], [22], [26], [37], [38].

Unfortunately, most of the algorithms are based on a very simple system model, which does not accurately reflect real parallel systems. The main problematic assumptions are: 1) a dedicated subsystem for the interprocessor communication, 2) completely concurrent communication, and 3) a fully connected communication network. The last two assumptions avoid the consideration of contention for communication resources in task scheduling. Experiments [24], [31] showed that the consideration of contention is essential for the generation of accurate and efficient schedules. A contention aware task scheduling strategy that captures end-point and network contention has been proposed in [34].

Even though the accuracy and efficiency of scheduling was significantly improved through the consideration of contention in task scheduling, the experiments also demonstrated that there is still one aspect regarding communication, which is not sufficiently addressed by the new contention model, namely, the involvement of the processors in communication, which is in opposition to the assumption of a dedicated subsystem for the interprocessor communication.

This paper investigates the involvement of the processor in communication, its impact on task scheduling and how it can be considered in 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. The concept of edge scheduling, used in contention aware scheduling, is extended to the scheduling of the edges on the processors in order to reflect the processors' involvement. Since scheduling under the new model requires the adjustment of the existing techniques, it is shown how this can be done for list scheduling and a new genetic algorithm based heuristic is proposed. Experiments are performed which demonstrate the greatly improved accuracy and efficiency of the schedules produced under the new involvement-contention model.

The remainder of this paper is organized as follows: Section 2 establishes the background and definitions of task scheduling under the classic and the contention model. Section 3 generically analyzes the processor involvement in communication. Based on this analysis, Section 4 investigates the integration of the awareness for the processor involvement in task scheduling. Section 5 discusses scheduling heuristics based on the new model. Experimental results are presented in Section 6 and this paper concludes with Section 7.

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 $e_{ij} \in \mathbf{E}$*

- *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 Tecnico, Technical University of Lisbon, Rua Alves Redol 9, P-1000-029 Lisboa, Portugal. E-mail: las@inesc-id.pt.*
- *F.E. Sandnes is with the Department of Computer Science, Faculty of Engineering, Oslo University College, PO Box 4, St. Olav Plass, N-0130 Oslo, Norway. E-mail: frode-eika.sandnes@iu.hio.no.*

Manuscript received 1 July 2004; revised 15 Feb. 2005; accepted 7 May 2005; published online 25 Jan. 2006.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0163-0704.

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, 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 Basis

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 consisting of a set $\mathbf{P}$ of **dedicated** processors, 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 $\min_{n \in \mathbf{V}}\{t_s(n)\} = 0$.

For such a 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,$n_j \in \mathbf{V}$,*

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

Condition 2 (Precedence Constraint (node strictness)). *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}, which is defined later, depending on the model.

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} with

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

and, hence,

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

for all $n \in \mathbf{V}$. If $\mathbf{pred}(n) = \emptyset$, i.e., n is a source node, $t_{dr}(n) = t_{dr}(n, P) = 0$, for all $P \in \mathbf{P}$.

2.2 Classic Scheduling

Most scheduling algorithms employ a strongly idealized model of the target parallel system [3], [4], [12], [14], [19], [22], [26], [37], [38]. This model, which shall be referred to as the classic model, is defined in the following, including a generalization toward heterogeneous processors.

Definition 2 (Classic System Model). *A parallel system $M_{classic} = (\mathbf{P}, \omega)$ consists of a finite set of dedicated 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 dedicated system has the following properties:*

1. *local communication has zero costs,*
2. *communication is performed by a communication subsystem,*
3. *communication can be performed concurrently, and*
4. *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 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, an unlimited number of communications can be performed at the same time, and communication has the same cost $c(e_{ij})$, regardless of the origin and the destination processor, unless the communication is local.

2.3 Contention Aware Scheduling

The classic scheduling model (Definition 2) does not consider any kind of contention for communication resources. To make task scheduling contention aware, and thereby more realistic, the communication network is modeled 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 [29], i.e., the scheduling of the edges of the DAG onto the links of the network graph, in a very similar manner to how the nodes are scheduled on the processors.

The details of contention aware scheduling are out of the scope of this paper, the interested reader should refer to [34]. The network model proposed in [34] captures network [29], [32] as well as end-point contention [6], [17] and can represent heterogeneous communication links. This is achieved by using different types of edges and by using switch vertices in addition to processor vertices. Here, it suffices to define the topology network graph to be $TG = (\mathbf{P}, \mathbf{L})$, where $\mathbf{P}$ is a set of vertices representing the processors and $\mathbf{L}$ is a set of edges representing the communication links. The system model is then defined as follows:

Definition 4 (Target Parallel System—Contention Model). *A target parallel system $M_{TG} = (TG, \omega)$ consists of a set of*

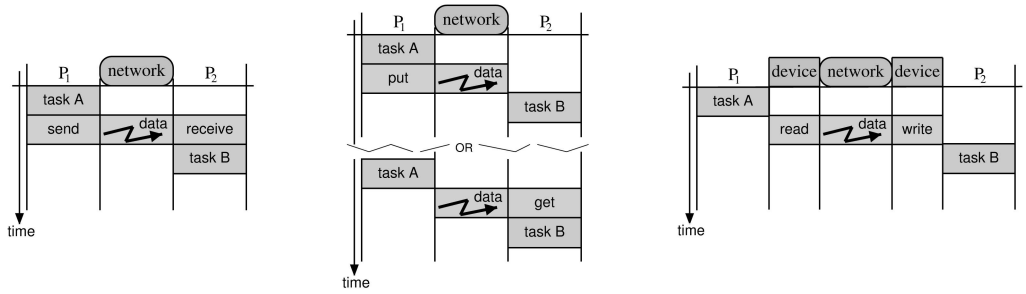

Fig. 1. The three types of interprocessor communication. (a) Two-sided. (b) One-sided. (c) Third party.

possibly heterogeneous processors $\mathbf{P}$ *connected by the communication network* $TG = (\mathbf{P}, \mathbf{L})$. *This dedicated system has the following properties:*

1. *local communications have zero costs and*
2. *communication is performed by a communication subsystem.*

The notions of concurrent communication and a fully connected network found in the classic model (Definition 2) are substituted by the notion of scheduling the edges $\mathbf{E}$ on the communication links $\mathbf{L}$. Corresponding to the scheduling of the nodes, $t_s(e, L)$ and $t_f(e, L)$ denote the **start** and **finish time** of edge $e \in \mathbf{E}$ on link $L \in \mathbf{L}$, 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}: $R = \langle L_1, L_2, \ldots, L_l \rangle$, $L_i \in \mathbf{L}$ for $i = 1, \ldots, l$. The edge e is scheduled on each link of the route.

This only affects the scheduling of the nodes with an altered definition of the edge finish time (Definition 3).

Definition 5 (Edge Finish Time—Contention Model). *Let* $G = (\mathbf{V}, \mathbf{E}, w, c)$ *be a DAG and* $M_{TG} = ((\mathbf{P}, \mathbf{L}), \omega)$ *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*

$$t_f(e_{ij}) = \begin{cases} t_f(n_i) & \text{if } proc(n_i) = proc(n_j) \\ t_f(e_{ij}, L_l) & \text{otherwise.} \end{cases} \quad (6)$$

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.

2.4 Scheduling Heuristics

The scheduling problem is to find a schedule with minimal length. As this problem is NP-hard, under the classic model [28], [35] as well as under the contention model [34], many heuristics have been proposed for its solution.

A heuristic must schedule a node on a processor so that it adheres to all resource (1) and precedence constraints (2). The following Condition 3 ensures this. The notion of a free node, used in the condition, is a node whose predecessors have already been scheduled, which is a requisite for the calculation of the DRT.

Condition 3 (Scheduling Condition). *Let* $G = (\mathbf{V}, \mathbf{E}, w, c)$ *be a DAG and* $\mathbf{M}_{classic} = (\mathbf{P}, \omega)$ *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*

$$\max\{A, t_{dr}(n, P)\} + \omega(n, P) \leq B. \quad (7)$$

So, Condition 3 allows node n to be scheduled between already scheduled nodes (**insertion technique**) [18], 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., $[A, B] = [t_f(P), \infty]$.

A similar condition is formulated for the scheduling of the edges on the links [34], additionally considering routing aspects and the causality of the communication along the route.

3 Processor Involvement in Communication

Experimental results demonstrated that the utilization of the contention aware model in scheduling heuristics significantly improves the accuracy and efficiency of the produced schedules [33]. Yet, the experiments also showed that the contention model is still not sufficiently realistic in terms of communication [31].

The contention model (Definition 4) supposes, as does the classic model (Definition 2), a dedicated communication subsystem to be present in the target system. With the assumed subsystem, computations can overlap with communications because the processor is not involved in communication. However, many parallel systems do not possess such a subsystem [10]. Therefore, in many systems the processors are involved, in one way or the other, in interprocessor communication. Furthermore, the involvement of the processor also serialises communication, even if the network interfaces were capable of performing multiple message transfers at the same time since a processor can only handle one communication at a time. For example, a processor can usually only perform one memory copy at a time. Thus, considering the processors' involvement in task scheduling is of utmost importance as it serializes the communication and, more importantly, prevents the overlap of computation and communication.

3.1 Involvement Types

In the context of the processor involvement, interprocessor communication can be divided into three basic types: **two-sided**, **one-sided**, and **third party**, as illustrated in Fig. 1.

Two-sided. In two-sided interprocessor communication both the source and the destination processor are involved in the communication (Fig. 1a). For example in a PC cluster,

the TCP/IP-based communication over the LAN involves both processors. The sending processor must divide a message into packages and wrap them into TCP/IP protocol envelopes before setting up the network card for the transfer. On the receiving side, the processor is involved in the unwrapping and assembling of the packages into the original message [10].

One-sided. Communication is said to be one-sided, if only one of the two participating processors is involved (Fig. 1b). Thus, this type of communication is limited to shared memory systems: either the source processor writes the data into its destination location (shared memory "put") or the destination processor reads the data from its source location (shared memory "get"). For example, on the Cray T3E a processor can read from or write into remote memory using special registers [10].

Third party. In third party interprocessor communication, the data transfer is performed by dedicated communication devices, as illustrated in Fig. 1c. The processor only informs the communication device of the memory area it wants transfered and the rest of the communication is performed by the device, usually employing some kind of Direct Memory Access (DMA). Examples for machines that possess such subsystems are Meiko CS-2 [5] or IBM SP-2 [15]. Task scheduling, both under the classic and under the contention model, assumes the third-party type of interprocessor communication (Definitions 2 and 4).

It is important to note that the software layer employed in parallel programming strongly affects the type of communication used. 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. This effect is not uncommon, as shown by the analysis of MPI implementations on common parallel systems in [16]. Task scheduling should of course reflect the effective type of involvement, taking into account the software layer employed in the code generation.

3.2 Involvement Characteristics

The classification of interprocessor communication into three different types can be refined with the notions of overhead and direct involvement.

Overhead. The first component of the processor's involvement is the communication setup or *overhead*. From the initiation of the communication process until the first data item is put onto the network, the processor is engaged in preparing the communication. An overhead is in general also imposed on the destination processor from after the data has arrived until it can be used by the receiving task. The overhead consists mainly of the path through the communication layers (e.g., MPI [25], Active Messages [36], TCP/IP) and, hence, is usually of constant time on both processors. In some environments, however, data might be copied into and from a buffer, which is an operation taking time proportional to the data size. Examples are some MPI implementations as described in [16]. Note, the overhead does not arise for communication between tasks executed on the same processor. Therefore, it cannot be made part of the computation reflected by the origin and the destination tasks of the communication.

Direct involvement. After the communication has been prepared by the processor during the overhead, any further participation in communication is the direct involvement of the processor. It is characterized by the circumstance that data is already in transit on the communication network. Fig. 2 features both the overhead and the direct involvement of the processor and thereby illustrates their differences. From now on, the term *involvement* means the direct involvement of the processor and the term *overhead* is used for the pre and postphases discussed above.

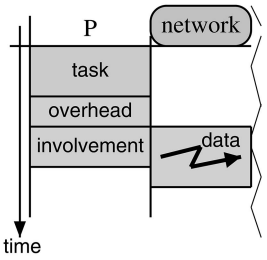

Fig. 2. The decomposing of the processor's participation in communication into overhead and direct involvement.

The distinction between communication types becomes obsolete if the communication is described in terms of overhead and involvement, even when the processors' participation is only partial. Therefore, it is assumed that overhead and involvement are imposed on both the source and the destination processor. Then, one communication type differs from another merely by the length of the two components. Logically, the involvement time is zero on some processors, namely, on both in third party communication and on one—either the sending or the receiving processor—in one-sided communication. In contrast, the overhead is a common part of interprocessor communication, independent of the type of communication. Thus, third party communication is affected just as much as two and one-sided communication, in terms of overhead.

The separation into overhead and involvement is also more general than the approach taken by the LogP model [9] and the computational overhead of communication considered in [28], which is discussed in detail in [30].

4 Involvement Scheduling

The notions of overhead and involvement discussed in the last section are the key concept to enhance task scheduling toward the awareness of the processor involvement in communication. In the first step, a new target system model is defined.

Definition 6 (Target Parallel System—Involvement-Contention Model). *A target parallel system $M = (TG, \omega, o, i)$ consists of a set of possibly heterogeneous processors $\mathbf{P}$ connected by the communication network $TG = (\mathbf{P}, \mathbf{L})$. This dedicated system has the following property:*

1. *local communication has zero costs.*

So, in comparison with the contention model (Definition 4), 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 o—for overhead—and i—for (direct) involvement.

Let $R = \langle L_1, L_2, \ldots, L_l \rangle$ be the route for the communication of edge $e \in \mathbf{E}$ from $P_{src} \in \mathbf{P}$ to $P_{dst} \in \mathbf{P}$, $P_{src} \neq P_{dst}$.

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 $o_r(e, P_{dst})$ is the overhead incurred by processor P_{dst} after receiving e.

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 $i_r(e, L_l)$ is the computational involvement incurred by P_{dst} during the transfer of e.

This is the general definition of overhead and involvement for heterogeneous arbitrary systems. Therefore, the overhead is made a function of the processor and the involvement a function of the utilized communication link. As discussed in the previous section, the overhead depends largely on the employed communication environment and is thereby normally unaffected by the utilized communication resources. In contrast, the involvement depends to a large extent on the capabilities of the utilized communication resources. Hence, the processor involvement is characterized by the outgoing or incoming link utilized for a communication.

With the distinction between the sending (o_s, i_s) and the receiving side (o_r, i_r) 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., $i_s(e, L) = i_r(e, L) = 0$ for involvement-free third party communication.

Note, for homogeneous systems or systems that have homogeneous parts, the definition of overhead and involvement can be significantly simplified. For example, in a systems with a homogenous network, the involvement functions can be defined globally, i.e., $i_{s,r}(e, L) = i_{s,r}(e)$.

4.1 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 $e \in \mathbf{E}$ on a processor $P \in \mathbf{P}$ is denoted by $t_s(e, P)$. Let $R = \langle L_1, L_2, \ldots, L_l \rangle$ be the route for the communication of edge $e \in \mathbf{E}$ from $P_{src} \in \mathbf{P}$ to $P_{dst} \in \mathbf{P}$, $P_{src} \neq P_{dst}$. The **finish time** of e on P_{src} is

$$t_f(e, P_{src}) = t_s(e, P_{src}) + o_s(e, P_{src}) + i_s(e, L_1) \tag{8}$$

and on P_{dst} it is

$$t_f(e, P_{dst}) = t_s(e, P_{dst}) + o_r(e, P_{dst}) + i_r(e, L_l). \tag{9}$$

Fig. 3 illustrates scheduling under the involvement-contention model. The execution time of an edge on a processor is the sum of the overhead and the involvement (see (8) and (9)).

As an edge scheduled on a processor represents computation, precisely the computation necessary for the communication of the edge, its scheduling must fulfil the processor constraint as formulated in Condition 1. For a meaningful and feasible schedule, the scheduling of the edges on the processors must obey this condition:

Condition 4 (Causality in Involvement Scheduling). *Let $R = \langle L_1, L_2, \ldots, L_l \rangle$ be the route for the communication of edge $e_{ij} \in \mathbf{E}$, $n_i, n_j \in \mathbf{V}$, from $P_{src} \in \mathbf{P}$ to $P_{dst} \in \mathbf{P}$, $P_{src} \neq P_{dst}$.*

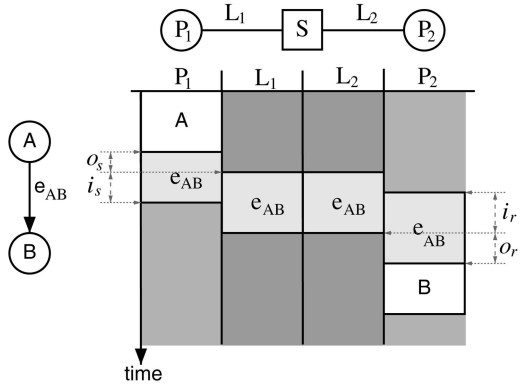

Fig. 3. Scheduling under the involvement-contention model: edges are also scheduled on the processors; (S—switch or other processor).

To assure the node strictness of n_i

$$t_s(e_{ij}, P_{src}) \geq t_f(n_i, P_{src}). \tag{10}$$

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

$$t_s(e_{ij}, L_1) \geq t_s(e_{ij}, P_{src}) + o_s(e_{ij}, P_{src}). \tag{11}$$

To assure the causality of the direct involvement on the destination processor P_{dst}

$$t_s(e_{ij}, P_{dst}) \geq t_f(e_{ij}, L_l) - i_r(e_{ij}, L_l). \tag{12}$$

The three inequalities can be observed in effect in Fig. 3. Edge e_{AB} starts on P_1 after the origin node A finishes ((10)). On the first link L_1, e_{AB} starts after the overhead finishes on P_1 ((11)), at which time the involvement of P_1 begins. And last, e_{AB} starts on P_2 so that the involvement finishes at the same time as e_{ij} on L_2 ((12)).

Condition 4 leaves scheduling algorithms some freedom for the node-edge order on the processor. An edge leaving a processor does not have to start immediately after the processor (10)—other edges or nodes can be executed before. The same principle holds on the destination processor (12).

4.1.1 Scheduling

Just as for the scheduling of the nodes on the processors (Section 2.4, Condition 3), a scheduling condition is formulated for the correct choice of an idle time interval into which an edge can be scheduled on a processor, with either the end or insertion scheduling technique.

Condition 5 (Edge Scheduling Condition on a Processor). *Let $G = (\mathbf{V}, \mathbf{E}, w, c)$ be a DAG, $M = ((\mathbf{P}, \mathbf{L}), \omega, o, i)$ a parallel system and $R = \langle L_1, L_2, \ldots, L_l \rangle$ the route for the communication of edge $e_{ij} \in \mathbf{E}$, $n_i, n_j \in \mathbf{V}$, from $P_{src} \in \mathbf{P}$ to $P_{dst} \in \mathbf{P}$, $P_{src} \neq P_{dst}$. Let $[A, B]$, $A, B \in [0, \infty]$, be an idle time interval on P, $P \in \{P_{src}, P_{dst}\}$. Edge e_{ij} can be scheduled on P within $[A, B]$ if*

$$B - A \geq \begin{cases} o_s(e_{ij}, P_{src}) + i_s(e_{ij}, L_1) & \text{if } P = P_{src} \\ o_r(e_{ij}, P_{dst}) + i_r(e_{ij}, L_l) & \text{if } P = P_{dst} \end{cases} \tag{13}$$

$$B \geq \begin{cases} t_f(n_i, P_{src}) + o_s(e_{ij}, P_{src}) + i_s(e_{ij}, L_1) & \text{if } P = P_{src} \\ t_f(e_{ij}, L_l) + o_r(e_{ij}, P_{dst}) & \text{if } P = P_{dst}. \end{cases} \tag{14}$$

This condition ensures that the time interval $[A, B]$ adheres to the inequalities (10) and (12) of the causality Condition 4. For an idle time interval $[A, B]$ adhering to Condition 5, the start time of e_{ij} on P_{src} and P_{dst} is

$$t_s(e_{ij}, P) = \begin{cases} \max\{A, t_f(n_i)\} & \text{if } P = P_{src} \\ \max\{A, t_f(e_{ij}, L_l) - i_r(e_{ij}, L_l)\} & \text{if } P = P_{dst}. \end{cases} \quad (15)$$

So, the edge is scheduled as early as possible within the limits of the interval. Of course, the choice of the interval should follow the same policy on the links and on the processors, i.e., either end or insertion scheduling should be used.

Note, the size of the involvement does not depend on the contention in the network. The assumption is that if the processor has to wait to send or receive a communication due to contention, this wait is passive or nonblocking, which means it can perform other operations in the meantime.

4.2 Node and Edge 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 Causality Condition 4 of Involvement Scheduling only imposes a constraint on the scheduling of an edge on the first link of its communication route. This is formulated in (11), which requires that edge e_{ij} must not start on the first link $L_1$of its route until after the overhead has finished on the source processor P_{src}, $t_s(e_{ij}, L_1) \geq t_s(e_{ij}, P_{src}) + o_s(e_{ij}, P_{src})$. 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 unchanged.

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

$$t_f(e_{ij}) = \begin{cases} t_f(n_i) & \text{if } proc(n_i) = proc(n_j) \\ t_f(e_{ij}, proc(n_j)) & \text{otherwise.} \end{cases} \quad (16)$$

4.3 NP-Completeness

Scheduling under the involvement-contention model remains an NP-hard problem. This is easy to see, as the involvement model is based on the contention model, which is NP-hard. It is proven with the straightforward reduction from the NP-complete decision problem C-SCHED (G, M_{TG}) associated with the scheduling under the contention model [30].

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

The formal proof is given in [30].

5 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 operating mode of scheduling algorithms. Essentially, the problem is that at the time a free 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 [17].

Two approaches to scheduling under the involvement-contention model are proposed, namely, 1) direct scheduling and 2) scheduling based on a given processor allocation. Both approaches are investigated in the next sections, followed by the proposal of two scheduling heuristics, one for each approach.

5.1 Direct Scheduling

Direct scheduling means 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 too 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 postponed 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 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.

Fig. 4 shows an example. First, A is scheduled on P_1, together with its three leaving edges (Fig. 4a), hence, the worst case that B, C, and D are going to be scheduled on P_2 is assumed. Indeed, node B is scheduled on P_2, which includes the preceding scheduling of e_{AB} on the link and on P_2. Next, C is scheduled on P_1, hence, e_{AC} is removed from P_2 (Fig. 4b). Finally, D is scheduled on P_2 with the respective scheduling of e_{AD} on the link and P_2 (Fig. 4c).

On heterogeneous systems, provisional scheduling of an edge on its source processor must take the fact that the involvement depends on the first link of the utilized route into consideration. Again, as the route is unknown during the scheduling, the worst case must be assumed. So, the ***provisional* finish time** of edge $e_{ij} \in \mathbf{E}$ on its source processor $P = proc(n_i)$, $P \in \mathbf{P}$, is

$$t_f(e_{ij}, P) = t_s(e_{ij}, P) + o_s(e_{ij}, P) + i_{s,max}(e_{ij}, P), \quad (17)$$

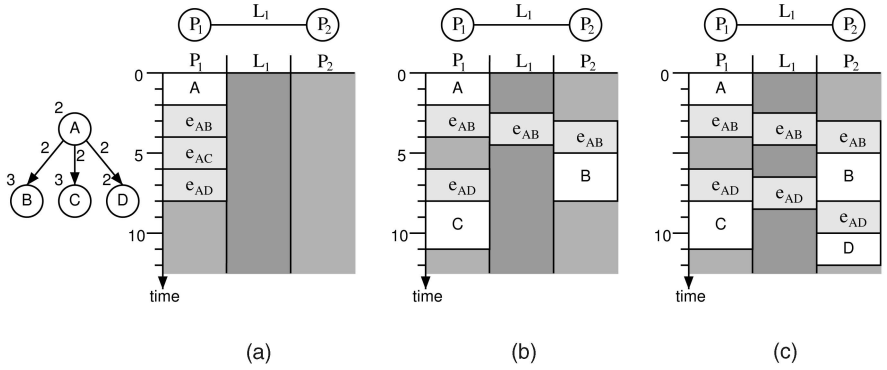

Fig. 4. Direct scheduling: edges are provisionally scheduled on source processor. (a) and (b) chart illustrate intermediate schedules and (c) the finial schedule of the depicted DAG; $o_{s,r}(e,P) = 0.5$, $i_{s,r}(e,L) = 0.75 \cdot (t_f(e,L) - t_s(e,L))$.

where $i_{s,max}(e_{ij},P) = \max_{L \in \mathbf{L}: L \, leaving \, P}\{i_s(e_{ij},L)\}$. 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 remaining scheduling can be performed as under the contention model. The disadvantage of this approach is the gaps left behind by removed edges, which can make a schedule less efficient. This shortcoming can be relieved using the insertion technique in scheduling. Inserting a node or an edge into a gap, is very likely to separate edges from their origin or destination node, but this is allowed by the causality Condition 4. Further, the gaps can be eliminated, i.e., the schedule is compacted, by repeating the scheduling. In this rescheduling, the nodes and their edges must be scheduled in the exact same order as in the first run, because as under the contention model the scheduling order of the edges matters. Then, with the processor allocation taken from the completed schedule, the provisional scheduling of edges becomes obsolete and the gaps are avoided.

5.2 Scheduling with Given Processor Allocation

The second approach to involvement scheduling assumes a given processor allocation, also referred to as a mapping, of the nodes to the processors as a prerequisite to the scheduling procedure. Hence, for every node it is known to where its successors will be scheduled. With a given mapping, the schedule is constructed with a list scheduling heuristic (Section 5.3), where the processor choice is simply a lookup from the given mapping.

The scheduling of an edge e_{ij} can be divided into three parts: scheduling on the source processor P_{src}, on the links of the route R, and on the destination processor P_{dst}. On the source processor, an edge must be scheduled together with its origin node n_i, as the foregoing considerations in the context of the direct scheduling showed. The scheduling on the links and on the destination processor can take place with either the origin node n_i or the destination node n_j. Hence, there are three alternatives for the scheduling of an edge e_{ij}.

1. The first alternative is identical to the approach of direct scheduling, where the edge e_{ij} is scheduled on the links of R and on the destination processor P_{dst}, when its destination node n_j is scheduled.
2. In this alternative, e_{ij} is not only scheduled on P_{src}, but also on the links, when n_i is scheduled. This way, the edges are scheduled on the links of R in the scheduling order of their origin nodes, while in the first alternative the edges are scheduled on the links in the order of their destination nodes. There is no clear advantage of the first over the second alternative or vice versa. Which one is better, i.e., more realistic, depends on the way the communication is realized in the target parallel system, whether it is initiated by the receiving (first alternative) or by the sending side (second alternative).
3. Edge e_{ij} is completely scheduled (i.e., on P_{src}, R, and P_{dst}), when its origin node n_i is scheduled. This alternative is likely to produce the best scheduling alignment of the edge on the source processor, the links and the destination processor, as the scheduling is done at once. Unfortunately, it has the disadvantage that the scheduling of the edges on their destination processors might prevent the efficient scheduling of the nodes. The early scheduling of the edges on their destination processors rapidly increases their finish times, leaving large idle gaps. Therefore, the conjoint scheduling of an edge on all resources is only sensible with the insertion scheduling technique.

The processor allocation can originate from any heuristic or can be extracted from a given schedule. For example, a schedule produced under the classic or contention model might serve as the input. In Section 5.4, a genetic algorithm is proposed for the determination of the processor allocation.

5.3 List Scheduling

In this section, list scheduling [1] is adapted for the involvement-contention model, using the direct scheduling approach (Section 5.1). 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: ▷ *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.

The schedule order of the nodes is important for the schedule length and many different priority schemes have

been proposed, e.g., [1], [32], [37]. 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, bl is defined as

$$bl(n_i) = w(n_i) + \max_{n_j \in \mathbf{succ}(n_i)} \{c(e_{ij}) + bl(n_j)\}. \quad (18)$$

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.

Algorithm 2 Scheduling of node n_j on processor P in involvement-contention model
1: **for** each $n_i \in \mathbf{pred}(n_j)$ **do**
2: **if** $proc(n_i) = P$ **then**
3: remove e_{ij} from P
4: **for** each $n_i \in \mathbf{pred}(n_j)$ in a definite order **do**
5: **if** $proc(n_i) \neq P$ **then**
6: determine route $R = \langle L_1, L_2, \ldots, L_l \rangle$ from $proc(n_i)$ to P
7: correct $t_f(e_{ij}, proc(n_i))$
8: schedule e_{ij} on R
9: schedule e_{ij} on P
10: schedule n_j on P
11: **for** each $n_k \in \mathbf{succ}(n_j)$ in a definite order **do**
▷ *reserve space for leaving edges*
12: schedule e_{jk} on P with worst case finish time

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. In this way, it is possible to consider the communication contention and the processor involvement in the scheduling decisions.

To determine the start time (i.e., the "`schedule`" lines in Algorithm 2) of a node or edge on a processor or link, both the end technique and insertion technique (Section 2.4) can be employed with list scheduling. Under the involvement-contention model, the insertion technique is more indicated, since the removing of provisionally scheduled edges leaves gaps (Section 5.1), which should be filled by other nodes or edges.

Compared 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 $O(\mathbf{P}(\mathbf{V} + \mathbf{E}O(routing)))$ (end technique) or $O(\mathbf{V}^2 + \mathbf{PE}^2O(routing))$ (insertion technique) [30]. $O(routing)$ is the complexity of determining the communication route and scheduling an edge on this route (lines 6 and 8 of Algorithm 2). In regular networks, determining the route is often $O(\mathbf{P})$ or even $O(1)$ (e.g., central switch or fully connected network). If the route can be determined in $O(1)$ time (calculated or looked up), then $O(routing)$ is just the complexity of the length of the route (the edge must be scheduled on each link of the route). For comparison, the complexity expressions under the classic model are $O(\mathbf{P}(\mathbf{V} + \mathbf{E}))$ (end technique) and $O(\mathbf{V}^2 + \mathbf{PE})$ (insertion technique).

5.4 A Genetic Scheduling Algorithm

In this section, a genetic algorithm (GA) based scheduling heuristic is proposed that follows the approach of using a given processor allocation (Section 5.2). Genetic algorithms have been successfully employed for the scheduling problem, e.g., [2], [7], [13], [21], [27]. A GA is a search algorithm which is inspired by the principles of evolution and natural genetics [11]. It begins with an initial population (a set of chromosomes) and then operates through a simple cycle of stages: evaluation of the chromosomes of the population, stochastic selection of the best chromosomes, and reproduction to create a new population, using crossover and mutation operators. This process is repeated and terminates after a specified number of generations or when a time limit is exceeded.

The idea of the proposed heuristic, referred to as Genetic Involvement-Contention Scheduling (GICS), is that the genetic algorithm searches for an efficient processor allocation, while the actual scheduling is performed with a list-scheduling-based heuristic, where the decision for a processor is taken from the calculated processor allocation. Below, the components of GICS are described.

Chromosome. The chromosome, encoded as an array of size $|\mathbf{V}|$, represents the processor allocation [7]. Hence, the value of array element i is the index of the processor assigned to node $n_i \in \mathbf{V}$.

Initial population. Randomly created chromosomes make the largest part of the initial pool of chromosomes. In order to avoid the creation of schedules that are slower than the sequential execution, sequential processor allocations, i.e. all nodes are allocated to one single processor, are also included. The pool is completed by one allocation extracted from a schedule produced with a list scheduling heuristic [21]. The chosen list scheduling uses finish time minimization, insertion scheduling and the nodes are in bottom-level order (Section 5.3).

Evaluation. At each iteration, a schedule is produced for every chromosome, i.e., processor allocation, with a heuristic based on the list scheduling structure (the decision for the processor is taken from the chromosome). The heuristic employs the third alternative for the scheduling of the edges (Section 5.2), where an edge is completely scheduled with its origin node. This kind of edge scheduling requires the insertion technique. To reduce the running time of the evaluation, the node order is determined only once at the beginning of the algorithm, namely, according to their bottom-levels. The **fitness value** of each chromosome is the length of its schedule.

Selection. Selection is performed by randomly picking two chromosomes out of the current pool from which the fitter one goes into the next generation, i.e., so called tournament selection. This process is repeated until the pool of the next generation has the specified population size. It is also guaranteed that the fittest chromosome, the one with the shortest schedule, survives.

Crossover. Two new chromosomes, nc_1, nc_2 are generated from two randomly selected chromosomes, c_1, c_2, by swapping the array values of a randomly determined element interval. Let the interval range from i to j. Outside this interval, i.e., $[1, i-1]$ and $[j+1, |\mathbf{V}|]$, the elements of nc_1 have the values of c_1 and inside those of c_2. For nc_2 it is converse.

Mutation. The mutation operator creates a new chromosome by copying a randomly picked chromosome and swapping the values of two randomly determined array elements. This operation helps to balance the load, as the number of nodes per processor is not changed. For DAGs with a high CCR, i.e., with a lot of communication, load balancing is not very promising. The CCR is the communication to computation ratio defined as the sum of all communication costs divided by the sum of all computation costs,

$$CCR = \frac{\sum_{e \in \mathbf{E}} c(e)}{\sum_{n \in \mathbf{V}} w(n)}.$$

Therefore, if the CCR is high (currently > 5), the swapping is substituted with an increase or decrease (randomly decided) by one of an array element.

The complexity of GICS is

$$O(population\_size \times number\_of\_generations \times (\mathbf{V} + \mathbf{E}O(routing))),$$

where the last term is the complexity of list scheduling without the processor choice (hence, without factor $\mathbf{P}$, see Section 5.3).

The edge scheduling of GICS is different to the edge scheduling of list scheduling as proposed in the previous section. Consequently, the comparison of the two heuristics in the following experiments will also provide some insights regarding the quality of the different edge scheduling approaches.

Finally, the control parameters which delivered good results and were used in the experiments are: population size = 100, number of generations = 40 (60 for DAGs with high CCR, see mutation operator), probability of a chromosome to participate in a crossover = 0.4, probability of mutation = 0.15.

6 Experimental Results

For the evaluation of the new involvement-contention model and the two proposed heuristics, the experimental methodology proposed in [33] is employed. A large set of graphs is generated and scheduled by algorithms under the different models to several target systems. Code is generated from the produced schedules, 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: accuracy and execution time. In the following, only the most important results are presented, but more experiments and details can be found in [30].

6.1 Improved Accuracy

In [31], 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 classic and the contention model are rescheduled, but now under the involvement-contention model. The two original heuristics are **Cl-LS(bl)**—list scheduling with bottom-level order—as a **classic** model heuristic, and **LS(dls)**—list scheduling with DLS's node order [29]—as a **contention** model heuristic. 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: a cluster (BOBCAT) of 16 PCs, modeled as a switched network; a shared memory multiprocessor system Sun E3500 with 8 processors, modeled as a bus network; a massively parallel system Cray T3E-900 with a total of 344 processors, modeled as a fully connected network.

Due to the lack of a profound insight into the target systems' communication mechanisms and their MPI implementations, 100 percent involvement is assumed, i.e., the source and destination processors are involved during the entire communication time on the first and last link, respectively: $i_s(e, L_1) = t_f(e, L_1) - t_s(e, L_1)$ and $i_r(e, L_l) = t_f(e, L_l) - t_s(e, L_l)$. The overhead is intuitively set to an experimentally measured setup time: $o_s(e, P) = o_r(e, P) = setup\_time$. While it is clear that this definition of the overhead and the involvement is probably not an accurate description of the target systems' communication behavior, 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.

6.1.1 Results

Fig. 5 visualizes the average accuracy deviations $\Delta acc(\mathcal{S})$ with

$$\Delta acc(\mathcal{S}) = \begin{cases} acc(\mathcal{S}) - 1 & \text{if } acc(\mathcal{S}) \geq 1 \\ 1/acc(\mathcal{S}) - 1 & \text{if } acc(\mathcal{S}) < 1, \end{cases} \quad (19)$$

where $acc(\mathcal{S})$ is the ratio of the execution time of the code produced for schedule $\mathcal{S}$ to its schedule length $sl(\mathcal{S})$.

It is immediately apparent from Fig. 5 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 more significant for medium ($CCR = 1$) and, especially so, for 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.

Generally, the difference between the reschedules from the classic model (Cl-LS(bl)) and the contention model (LS(dls)) on the same system is small. This is desirable, as the origin of the original schedule should not affect the accuracy under the involvement-contention model.

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 [16], which does not match the assumption of nonblocking communication made in the involvement contention model. Further, the employed overhead and involvement functions are very rough estimates, a better approximation of these functions might improve the accuracy. In any case, it is in the nature of any model that there is a

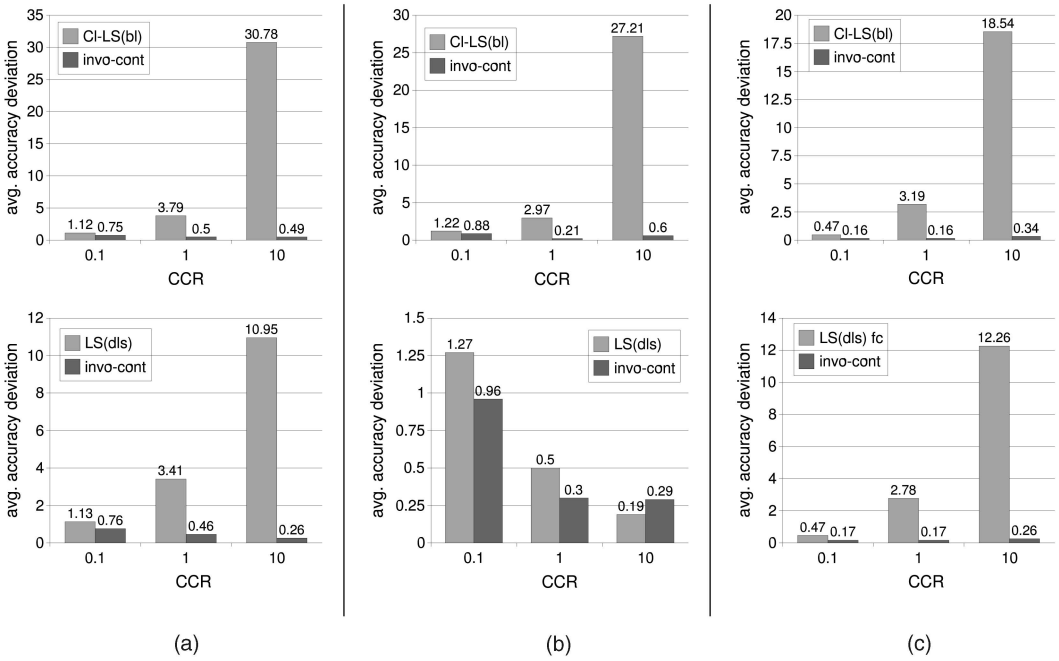

Fig. 5. Average accuracy deviation of Cl-LS(bl) (classic model) (top) and LS(dls) (contention model) (bottom) compared with rescheduling under the involvement-contention model (invo-cont). (a) BOBCAT. (b) SUN E3500. (c) T3E-900.

TABLE 1
Workload DAGs

structure	# nodes	avg. in/out-degree $\bar{\delta}$	weights
random in-tree	175, 406, 233 ($CCR = 0.1, 1, 10$)	≈1.0	random
random out-tree	176, 648, 319 ($CCR = 0.1, 1, 10$)	≈1.0	random
random multiple fork-join	500	1.94	random
Gaussian elimination (e.g. [8])	495 ($n = 31$)	1.88	regular
stencil (e.g. [23])	400 (20×20)	2.76	unit
pipeline (e.g. [20])	500 (25×20)	1.86	unit
random	600	≈2.0	random

difference between prediction and reality. 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.

6.2 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 [33]. As the intention here is to compare scheduling models and not algorithms, the same heuristic is employed for each model analyzed.

Seven different graph types constitute the DAG workload, listed in Table 1. The average in-degree or the average out-degree (they are identical) of the nodes ($\bar{\delta}$), i.e., the average number of incoming or outgoing edges, characterizes the density of a graph. Three instances of each of the seven DAGs are generated, one instance with a CCR of 0.1, one with a CCR of 1 and one with a CCR of 10. To achieve the CCR values in the DAGs with regular weights, the node and edge weights are scaled accordingly.

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, the contention and the involvement-contention model, denoted by "classic," "cont," and "invo-cont," respectively. Using the same algorithm for each model allows the impact of the model on the quality of the produced schedules to be analyzed and compared, without the influence of different scheduling techniques. Further, in order to evaluate scheduling with a given processor allocation under the involvement-contention model, the genetic algorithm "GICS" (Section 5.4) is utilized.

Code is generated for the obtained schedules and executed on two different target systems: Sun E3500 and BOBCAT (Cray T3E-900 was not used, as it simply was taken out of service at the Edinburgh Parallel Computing Centre before the realization of the experiments). Both systems were modeled as in the previous experiments of Section 6.1.

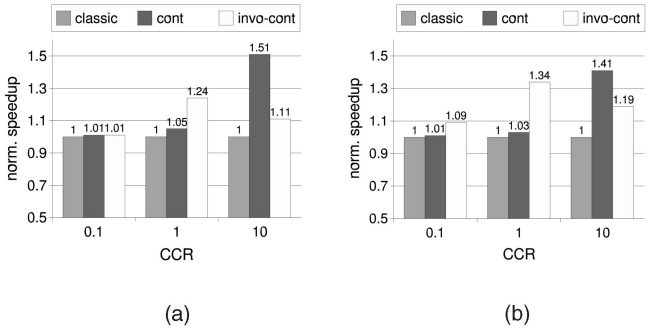

Fig. 6. BOBCAT: Average speedups under the three models (speedups are normalized to values under classic model). (a) On eight processors. (b) On 16 processors.

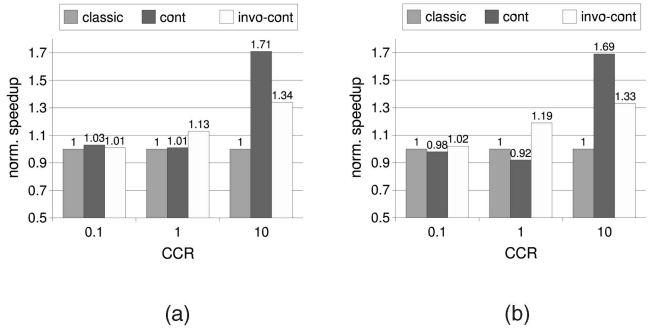

Fig. 7. Sun E3500: Average normalized speedups under the three models (speedups are normalized to values under classic model). (a) On four processors. (b) On seven processors.

6.2.1 Results

Figs. 6 and 7 visualize the average normalized speedups of the three different models on different configurations of the two target systems. Since GICS is based on a different algorithm (not list scheduling), GICS is treated separately below. The speedup is defined as the ratio of the sequential time $seq(G) = \sum_{n \in \mathbf{V}} w(n)$ (local communication has zero costs) to the execution time of the code produced for schedule $\mathcal{S}$.

Low communication ($CCR = 0.1$). For a CCR of 0.1, communication is of low significance compared to the computation costs. Nevertheless, the involvement-contention model noticeably reduces the execution times—compared to the classic model—on BOBCAT with 16 processors (average speedup is improved by 9 percent). With more processors, communication becomes more important, because it is probable that more communications are performed remotely.

Medium communication ($CCR = 1$). The situation changes for DAGs with medium communication. Schedules produced under the involvement-contention model have significantly shorter execution times, with speedup improvements of up to 82 percent (Gauss elimination). Again, the improvement increases with the number of the utilized processors.

High communication ($CCR = 10$). The highest reduction in the execution time is apparent in the results of the involvement-contention for DAGs with high communication. Unfortunately, this reduction is irrelevant: the absolute speedup of the schedules of all models is below 1. Hence, a parallelization is meaningless and the sequential execution is indicated.

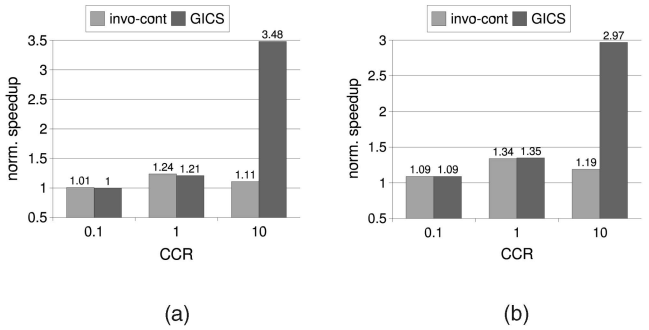

Fig. 8. BOBCAT: Average normalized speedups of cont-invo and GICS (speedups are normalized to values under classic model). (a) On eight processors. (b) On 16 processors.

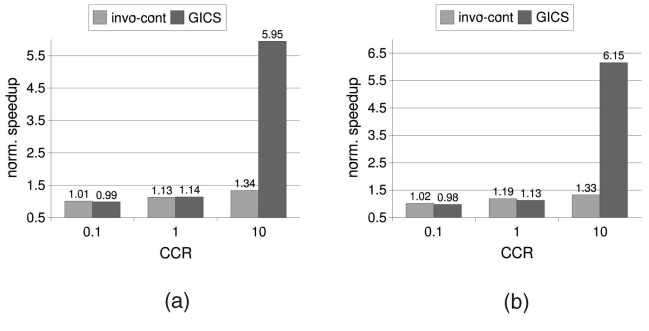

Fig. 9. Sun E3500: Average normalized speedups of cont-invo and GICS (speedups are normalized to values under classic model). (a) On four processors. (b) On seven processors.

GICS. Figs. 8 and 9 summarize the average normalized speedups of GICS compared to invo-cont. Not regarding high communication, the execution times of the schedules by GICS are almost identical to those by invo-cont. This is surprising, as at least a slight improvement was expected, since one of GICS's initial chromosomes is the processor allocation of the invo-cont schedule (Section 5.4). Yet, invo-cont and GICS differ in the way they schedule edges: invo-cont employs the direct scheduling approach (Section 5.1), which corresponds to the first alternative of the edge scheduling approaches (Section 5.2). In contrast, GICS uses the third alternative, which apparently leads to no improvement by GICS. Hence, the first alternative should be preferred over the third in scheduling under the involvement-contention model.

GICS's good results for high communication are due to the fact that sequential processor allocations are part of its initial population. Indeed, nearly all schedules by GICS for $CCR = 10$ are (almost) sequential, with an absolute speedup of 1. Thus, neither GICS is able to efficiently parallelize the DAGs with $CCR = 10$. However, the accuracy of the involvement-contention model allows GICS to know that.

6.2.2 Discussion

In the experiments conducted, the involvement-contention model clearly demonstrated its ability to produce schedules with significantly reduced execution times.

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. List scheduling is a greedy algorithm,

which tries to reduce the finish time of each node to be scheduled. Thereby it does not consider the leaving communications of a node, which may impede an early start of following nodes. The high importance of communication under the involvement-contention model seems to demand the research of more sophisticated algorithms in order to exploit the full potential of this new model.

7 Conclusions

This paper investigated the involvement of the processor in communication, its impact on task scheduling and how to make task scheduling aware of it. First, the different types of processor involvement and their characteristics were investigated. A new system model was proposed, which extends the scheduling of the edges on the links to their scheduling on the processors. This technique can reflect the three types of processor involvement and the distinction between overhead and direct involvement. Scheduling the edges on the processors has an impact on the operating techniques of scheduling heuristics. This challenge was investigated and two solutions were proposed: provisional scheduling and using a given processor allocation. Based on these solutions, two scheduling algorithms were proposed for the new involvement-contention model: an adapted list scheduling and a genetic algorithm (GICS). Extensive experiments demonstrated that the involvement-contention model significantly improves the accuracy and the execution time of the produced schedules. The improved accuracy now allows for a useful estimation of the execution time. In order to achieve further improvements, research into algorithms that are better at exploiting the new model is needed.

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 they 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 M.K. Dhodhi, "Multiprocessor Scheduling in a Genetic Paradigm," *Parallel Computing,* vol. 22, pp. 395-406, 1996.

[3] 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, Aug. 1998.

[4] 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,* pp. 207-213, June 1996.

[5] A. Alexandrov, M. Ionescu, K.E. Schauser, and C. Scheimann, "LogGP: Incorporating Long Messages into the LogP-Model—One Step Closer Towards a Realistic Model for Parallel Computation," *Proc. Seventh Ann. Symp. Parallel Algorithms and Architectures,* pp. 95-105, 1995.

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

[7] M.S.T. Benten and S.M. Sait, "Genetic Scheduling of Task Graphs," *Int'l J. Electronics,* vol. 77, no. 4, pp. 401-415, 1994.

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

[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 SIGPLAN 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, 1999.

[11] L. Davis, *Handbook of Genetic Algorithms.* New York: Van Nostrand-Reinhold, 1991.

[12] 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.

[13] E.S.H. Hou, N. Ansari, and H. Ren, "Genetic Algorithm for Multiprocessor Scheduling," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 2, pp. 113-120, Feb. 1994.

[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] IBM, *SP Switch2 Technology and Architecture,* Mar. 2001, http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/sp_switch2.pdf.

[16] J. White III and S. Bova, "Where's the Overlap? An Analysis of Popular MPI Implementations," *Proc. MPIDC,* 1999.

[17] T. Kalinowski, I. Kort, and D. Trystram, "List Scheduling of General Task Graphs under LogP," *Parallel Computing,* vol. 26, pp. 1109-1128, 2000.

[18] B. Kruatrachue, "Static Task Scheduling and Grain Packing in Parallel Processing Systems," PhD thesis, Oregon State Univ., 1987.

[19] B. Kruatrachue and T. Lewis, "Grain Size Determination for Parallel Processing," *IEEE Software,* vol. 5, no. 1, pp. 23-32, Jan. 1988.

[20] S.Y. Kung, *VLSI Array Processors,* Information and System Sciences Series. Prentice Hall, 1988.

[21] Y.-K. Kwok and I. Ahmad, "Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using a Parallel Genetic Algorithm," *J. Parallel and Distributed Computing,* vol. 47, no. 1, pp. 58-77, Nov. 1997.

[22] Y.-K. Kwok and I. Ahmad, "Benchmarking the Task Graph Scheduling Algorithms," *Proc. Int'l Parallel Processing Symp. /Symp. Parallel and Distributed Processing,* pp. 531-537, Apr. 1998.

[23] W. Löwe, W. Zimmermann, S. Dickert, and J. Eisenbiegler, "Source Code and Task Graphs in Program Optimization," *Proc. Conf. High Performance Computing and Networking,* pp. 273-282, 2001.

[24] B.S. Macey and A.Y. Zomaya, "A Performance Evaluation of CP List Scheduling Heuristics for Communication Intensive Task Graphs," *Proc. Parallel Processing Symp.,* pp. 538-541, 1998.

[25] "Message Passing Interface Forum," *MPI: A Message-Passing Interface Standard,* June 1995, http://www.mpi-forum.org/docs/docs.html.

[26] 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, Ecole Normale Superieure de Lyon, Laboratoire de Informatique du Parallelisme, Lyon, France, 1998.

[27] F.E. Sandnes and G.M. Megson, "An Evolutionary Approach to Static Taskgraph Scheduling with Task Duplication for Minimised Interprocessor Traffic," *Proc. Int'l Conf. Parallel and Distributed Computing, Applications, and Technologies,* pp. 101-108, July 2001.

[28] V. Sarkar, *Partitionning and Scheduling Parallel Programs for Execution on Multiprocessors.* MIT Press, 1989.

[29] 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.

[30] O. Sinnen, "Accurate Task Scheduling for Parallel Systems," PhD thesis, Instituto Superior Técnico, Technical Univ. of Lisbon, Portugal, Apr. 2003.

[31] O. Sinnen and L. Sousa, "Experimental Evaluation of Task Scheduling Accuracy: Implications for the Scheduling Model," *IEICE Trans. Information and Systems,* vol. 86, no. 9, pp. 1620-1627, Sept. 2003.
[32] 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.
[33] O. Sinnen and L. Sousa, "On Task Scheduling Accuracy: Evaluation Methodology and Results," *J. Supercomputing,* vol. 27, no. 2, pp. 177-194, Feb. 2004.
[34] O. Sinnen and L. Sousa, "Communication Contention in Task Scheduling," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 6, pp. 503-515, June 2005.
[35] J.D. Ullman, "NP-Complete Scheduling Problems," *J. Computing System Science,* vol. 10, pp. 384-393, 1975.
[36] T. von Eicken, D.E. Culler, S.C. Goldstein, and K.E. Schauser, "Active Messages: A Mechanism for Integrated Communication and Computation," *Proc. 19th Ann. Int'l Symp. Computer Architecture,* pp. 256-266, May 1992.
[37] 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.
[38] 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.

Oliver Sinnen received three degrees in electrical and computer engineering: the diploma degree (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 Tecnico (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 Augusto Sousa received the PhD degree in electrical and computer engineering from the Instituto Superior Tecnico(IST), Universidade Tecnica de Lisboa, Lisbon, Portugal, in 1996. He is currently a member of the Electrical and Computer Engineering Department at IST and a senior researcher at the Instituto de Engenharia de Sistemas e Computadores-R&D. His research interests include VLSI architectures, computer architectures, parallel and distribuded computing, and multimedia systems. He has contributed to more than 70 papers to journals and international conferences and he is currently a member of HiPEAC European Network of Excellence. He is a senior member of IEEE and the IEEE Computer Society and a member of ACM.

Frode Eika Sandnes received the BSc degree in computing science from the University of Newcastle-Upon-Tyne, England, in 1993, and the PhD degree in computer science from the University of Reading, England, in 1997. He has several years of experience from the space industry developing communications and onboard systems for low-earth orbit environmental satellites. He is currently an associate professor in the Department of Computer Science at Oslo University College, Norway. Dr. Sandnes' research interests include multiprocessor scheduling, error-correction, and mobile human computer interaction.