CHAPTER 7

Communication Contention in Scheduling

While task scheduling is an NP-hard problem, its theoretical foundations, as discussed in previous chapters, are easily understood through the high level of abstraction. Computation and communication are strictly separated, whereby the communication is free of contention. Yet, for task scheduling to be more than a mere theoretical exercise, the level of abstraction must be sufficiently close to the reality of parallel systems. During the course of this and the next chapter, it will become clear that this is not the case for the classic model of the target parallel system (Definition 4.3). Therefore, recently proposed models are analyzed that overcome the shortcomings of the classic model, while preserving the theoretical basis of task scheduling.

The critical analysis of the classic model performed in this and the next chapter identifies three of the model's properties, all regarding communication, to be often absent from real systems. Many parallel systems do not possess a dedicated communication subsystem (Property 4); consequently, the processors are involved in communication. The number of resources for communication, such as network interfaces and links, is limited, which contradicts Property 5 of concurrent communication. And finally, only a small number of parallel systems have networks with a fully connected topology, which is Property 6 of the classic model. As a consequence, the following general issues must be investigated:

- The topology of communication network
- The contention for communication resources
- The involvement of processors in communication

This chapter primarily investigates how to handle contention for communication resources in task scheduling. As the network topology has a major impact on contention, the consideration of network topologies, other than fully connected, is a requisite and addressed beforehand. The next chapter will deal with the involvement of the processors in communication. This is done based on the target system model presented in this chapter, so that the final model considers all of the referred issues.

As will be seen, incorporating contention awareness and network topologies into task scheduling leaves the principles of scheduling almost untouched. In contrast, considering the involvement of the processor in communication has a significant impact on how scheduling algorithms work.

The objective of making task scheduling contention aware is the generation of more accurate and efficient schedules. For this objective the following goals are pursued: (1) achieving a more accurate and general representation of real parallel systems; (2) keeping the system model as simple as possible; and (3) allowing the adoption of the scheduling concepts and techniques of the classic model.

Section 7.1 begins the discussion of contention aware scheduling by stating the problem more precisely and reviewing different approaches to its solution. Based on this review, the principles of the approach adopted in this text are described. In order to achieve high generality, the new model should permit one to represent heterogeneity in terms of the communication network as well as in terms of processors. The representation of the communication network, addressed in Section 7.2, is designed accordingly. The essential concept to achieve contention awareness—edge scheduling—is presented and analyzed in Section 7.3. Using edge scheduling, task scheduling becomes contention aware, which is elaborated in Section 7.4, including an analysis of the consequences for the task scheduling framework. Finally, Section 7.5 shows how scheduling algorithms, in particular, list scheduling, are adapted to the new scheduling model.

7.1 CONTENTION AWARENESS

Contention for communication resources arises in parallel systems, since the number of resources is limited. 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.

The classic model of the target parallel system (Definition 4.3) ignores this circumstance by assuming that all communications can be performed concurrently (Property 5) and that the processors are fully connected (Property 6). However, recent publications have demonstrated the importance of contention awareness for the accuracy and the execution time of schedules (Macey and Zomaya [132], Sinnen and Sousa [171, 176, 178]). At the end of this chapter in Section 7.5.4 these experimental results are reviewed in more detail. Also, in data parallel programming, contention aware scheduling of communications is common (Tam and Wang [183]). Another indicator for the importance of contention is the fact that the LogP programming model (Culler et al. [46, 47]), discussed in Section 2.1.3, explicitly reflects communication contention.

To achieve an awareness of contention, scheduling heuristics must recognize the actualities of the parallel system and consequently adapt the target system model. Only a few scheduling algorithms have been proposed, having a target system view different from the classic model and thereby considering contention. These approaches are discussed in the following. Roughly, contention can be divided into end-point and network contention.

7.1.1 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.

Beaumont et al. [19] extend the classic model by associating a communication port with each processor, called one-port model. At any instance in time, only one incoming and one outgoing communication can be carried out through this port. For example, multiple leaving communications of a node, which are ready as soon as the task finishes execution (node strictness, Definition 3.8), are serialized through this approach. A port can be regarded as the processor's network interface, allowing bidirectional full duplex transfers. Communication between disjoint pairs of processors can still be performed concurrently. Thus, the one-port model captures end-point contention but does not reflect network contention.

A similar, but more general, approach is accomplished with the parameter g of the LogP model (Section 2.1.3). A few scheduling algorithms have been proposed for the LogP model (e.g., Boeres and Rebello [25], Kalinowski et al. [98], Löwe et al. [131]). In these algorithms, 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 (see Section 2.1.3). This approach is more general, because the gap can be set to a value smaller than $o + L$ (overhead + latency), thus allowing several communications to overlap. All of the above referenced algorithms neglect the network capacity defined in LogP of at most $\lceil L/g \rceil$ simultaneous message transfers in the network.

End-point contention is only a partial aspect of contention in communication. The limited number of resources within the network (Section 2.2) also leads to conflicts. Furthermore, both of the above approaches suppose completely homogeneous systems and are tied to systems where each processor has one full duplex communication port.

7.1.2 Network Contention

To successfully handle communication contention in scheduling, an accurate model of the network topology is required. Static networks, that is, networks with fixed connections between the units of the system, are commonly represented as undirected graphs. This representation form was informally used during the description of static networks in Section 2.2.1. In the following it is defined formally.

Definition 7.1 (Undirected Topology Graph) *The topology of a static communication network is modeled as an undirected graph* $UTG = (\mathbf{P}, \mathbf{L})$. *The vertices in* $\mathbf{P}$ *represent the processors and the undirected edges in* $\mathbf{L}$ *the communication links* *between them. Link $L_{ij} \in \mathbf{L}$ represents the bidirectional communication link between the processors P_i and P_j and $P_i, P_j \in \mathbf{P}$.*

Examples of such undirected topology graphs are omitted here, but a large number of them can be found in Section 2.2.1. The undirected graph model is a simple and intuitive representation of communication networks. In task scheduling it is employed for various purposes. Wu and Gajski [207] apply it to map scheduled task graphs onto physical processors, trying to balance the network traffic. Other algorithms utilize it for a more accurate determination of communication costs, for example, by counting the "hops" of a communication (i.e., the transitions between links) (e.g., Coli and Palazzari [39], Sandnes and Megson [164]).

Contention Awareness with Undirected Topology Graph Primarily, the undirected topology graph is the model employed in the few existing network contention aware scheduling heuristics.

El-Rewini and Lewis [63] propose a contention aware scheduling algorithm called MH (mapping heuristic). The target system is represented as an undirected topology graph and a table is associated with each processor. This table maintains for each processor (1) the number of hops on the route, (2) the preferred outgoing link, and (3) the communication delay due to contention. With this information, the communication route between processors is adapted during the execution of the list scheduling based MH. However, the tables are only updated at each start and at each finish of a communication, which is a trade-off between complexity and a real view of the traffic. Furthermore, such a scheduling approach is only meaningful if the target system can be forced to use the routes determined by the algorithm, which is usually not the case (Section 7.2.2).

Concept of Edge Scheduling A more realistic view of the network traffic is gained when, for each communication, its start and finish times on every utilized communication link are recorded. This corresponds to scheduling the edges on the links, like the nodes are scheduled on the processors, which is illustrated in the Gantt chart of Figure 7.1.

The idea of edge scheduling emerged with the DLS algorithm proposed by Sih and Lee [169]. They proposed representing the target system as an undirected topology

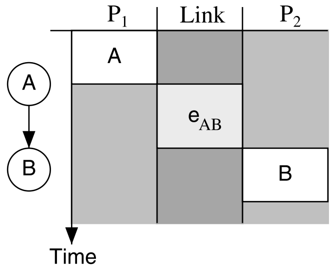


Figure 7.1. The concept of edge scheduling.

graph and employing architecture-dependent routing routines, as opposed to MH. Unfortunately, they do not elaborate on the edge scheduling technique and how it is integrated into classic scheduling.

BSA (bubble scheduling and allocation) is another contention aware algorithm that employs edge scheduling and the undirected topology graph (Kwok and Ahmad [114]). While DLS can be regarded as a dynamic list scheduling algorithm (Section 5.1.2), BSA's approach is quite different, in that it first allocates all nodes to one processor. Subsequently, nodes are migrated to adjacent processors if beneficial. Next, the nodes on the adjacent processors are considered for migration and so on until all processors have been regarded. The routing of the communication is done incrementally with each migration of a node.

BSA has two issues owing to its scheduling strategy, which are analyzed in Sinnen [171]. First, edge scheduling is performed with almost no restrictions. For example, an edge might be scheduled earlier on a link at the end of its route than on a link at the beginning of the route. Obviously, this contradicts causality; hence, it is not a realistic view of the network traffic. Effectively, this approach only captures contention at the last link of a route. Furthermore, the incremental routing can lead to situations where communications are routed in a circle or even use a link twice.

Another task scheduling algorithm that features the scheduling of messages is proposed in Dhodhi et al. [55]. In this genetic algorithm based heuristic, each interprocessor communication is scheduled on the connection between the two communicating processors. The communication network is not modeled, which consequently implies the assumption of a fully connected system.

Virtual Processors Contention for (any type of) resources can also be addressed with the virtual processor concept. This concept is typically found in the domain of real time system scheduling (Liu [127]). Each resource of such a system is modeled as a virtual processor. For interprocessor communication this means that each communication link is a (virtual) communication processor. Correspondingly, the task concept is also abstracted, for example, a communication is considered a job that can only be processed on the (virtual) communication processor.

While this approach is very general, it makes scheduling much more difficult. This is because scheduling has to deal with heterogeneous jobs and heterogeneous (virtual) processors, both in terms of functionality. Task scheduling, as discussed in this text, has only to deal with one type of task. Processor heterogeneity, as introduced in Section 6.3, deals with different processing speeds, not functionality. Due to this conceptual complexity, scheduling heuristics for virtual processors usually have two phases, one for the mapping of the jobs onto the processors and one for the actual ordering/scheduling of the jobs (see also Section 5.2).

This functional heterogeneity of the virtual processor concept is a crucial shortcoming, as many existing task scheduling algorithms cannot be employed. On the other hand, edge scheduling can be integrated into task scheduling with little impact on the general scheduling techniques. Algorithms designed for the classic model can be used almost without any modification as will be seen later. Also, a closer inspection reveals that edge scheduling is in fact similar to the concept of virtual processors: communications (jobs) are scheduled on the links (resources). Yet, edge scheduling is less general and can thereby consider the special circumstances of communication scheduling. For example, local communications should not be scheduled at all, because their costs are negligible.

7.1.3 Integrating End-Point and Network Contention

Edge scheduling, thanks to its strong similarity with task scheduling, has the potential to accurately reflect contention in communication. Therefore, the here adopted approach to contention awareness is based on the edge scheduling concept. The fundamentals of edge scheduling are investigated in Section 7.3. It is aimed at a theory that accurately reflects the real behavior of modern parallel systems, including aspects like routing and causality.

However, edge scheduling on top of the undirected topology graph is limited in several aspects:

- Only half duplex communication links are modeled by the graph.
- Only static networks can be represented.
- End-point contention (Section 7.1.1) is not reflected. A processor connected to multiple links, for example, in a mesh (Section 2.2.1), can simultaneously perform one communication on each link.

Thus, to make edge scheduling an efficient instrument for contention aware scheduling, these issues must be resolved with an improved network model. The next section presents such a model.

7.2 NETWORK MODEL

This section introduces a model for the representation of communication networks of parallel systems in contention aware scheduling (Sinnen and Sousa [172, 179]). The discussion begins with the topology graph that represents the resources of the network. Being a generalization of the undirected graph, the topology graph is capable of capturing both end-point and network contention. The discussion is followed by the analysis of routing, as it determines how the resources are employed.

7.2.1 Topology Graph

The undirected topology graph (Definition 7.1) is a simple and intuitive representation of communication networks. However, as stated in the previous section, it has several shortcomings regarding its utilization for edge scheduling. The discussion and analysis of these issues in the following lead to the proposition of a new topology model. Being a generalization, the new model includes the undirected topology graph but is not limited to static networks.

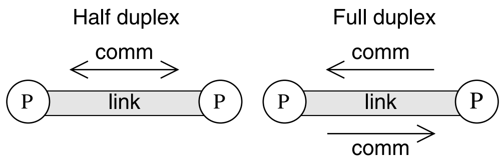


Figure 7.2. Half duplex (left, one communication at a time) and full duplex link (right, two communications—one in each direction).

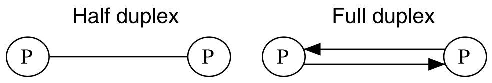


Figure 7.3. Representing half (left) and full duplex links (right) with undirected and directed edges, respectively.

Directed Links The undirected topology graph represents each communication link of the network as one undirected edge. Thus, it is incapable of distinguishing between full and half duplex links. A full duplex point-to-point communication link can transport two communications, or messages, at a time, one in each direction, whereas a half duplex link only transfers one message at a time, independent of its direction. This is illustrated in Figure 7.2.

A distinction between these two communication modes is achieved by representing a full duplex link as two counterdirected edges and a half duplex link as an undirected edge, as shown in Figure 7.3.

An alternative approach would be to use only undirected edges and a special identifier to distinguish between full and half duplex links. Then, however, edge scheduling would have to be aware of the type of link, that is, whether it can schedule two edges with overlapping start–finish intervals (full duplex) or not (half duplex). A better level of abstraction is attained with the utilization of directed edges. The graph encompasses all information about the topology and the communication characteristics of the links. Thus, edge scheduling can be insensible to the edge type.

For comparison, the contention aware scheduling algorithms based on the undirected topology graph (Section 7.1.2) suppose half duplex links, while end-point contention heuristics (Section 7.1.1) assume full duplex links.

Moreover, employing directed edges even admits the representation of unidirectional communication links. They are sometimes encountered in ring-based topologies, as illustrated in Figure 7.4, for example, networks using the SCI (scalable coherent interface) standard (Culler and Singh [48]).

Busses Even when extended with directed edges, the topology model of Definition 7.1 is still limited to static networks. To reflect the important bus topology of dynamic networks (Section 2.2.2) in this model, it must be approximated by a fully connected network (Section 2.2.1). However, for contention scheduling, the fact that the bus is shared among all processors is crucial. Graphs, as defined in Section 3.1, inherently can only represent point-to-point communication links,

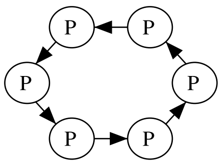


Figure 7.4. Unidirectional ring network, represented with directed edges.

regardless of whether edges are directed or undirected. Therefore, Sinnen and Sousa [176, 178] propose the utilization of hyperedges to represent bus-like topologies in contention aware scheduling. A hypergraph (i.e., a graph containing hyperedges) is a generalization of the undirected graph, where edges—hyperedges—can be incident on more than two vertices (Berge [21]).

Definition 7.2 (Hyperedge and Hypergraph) *Let* $\mathbf{V}$ *be a finite set of vertices. A hyperedge* H *is a subset consisting of two or more vertices of* $\mathbf{V}$, $H \subseteq \mathbf{V}$, $|H| > 1$. *A hypergraph* HG *is a pair* $(\mathbf{V}, \mathbf{H})$ *of a finite set of vertices* $\mathbf{V}$ *and a finite set of hyperedges* $\mathbf{H}$.

Note that an undirected edge is a hyperedge incident on exactly two vertices and an undirected graph can be considered a hypergraph containing only such edges. Figure 7.5 visualizes the difference between a 4-vertices undirected graph, fully connected, and a 4-vertices hypergraph, with one hyperedge. In both graphs, every vertex is adjacent to every other vertex.

Switches The above discussed shortcomings of the undirected topology graph and their solutions permit the improved representation of static networks and the incorporation of bus topologies. However, the modeling of dynamic networks in general has not been addressed yet. Since dynamic networks are widely employed in parallel systems (Section 2.2), the approximation of these networks as fully connected is not appropriate.

The essential component of a dynamic network is the switch, which significantly determines the network's characteristic. So while the above improvements target the edges of the topology model, switches are incorporated into the graph by means of a new vertex type—a switch vertex. In other words, a vertex in the topology graph

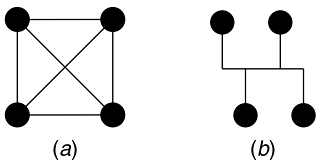


Figure 7.5. Undirected graph (*a*) with only point-to-point edges and hypergraph (*b*) with one hyperedge.

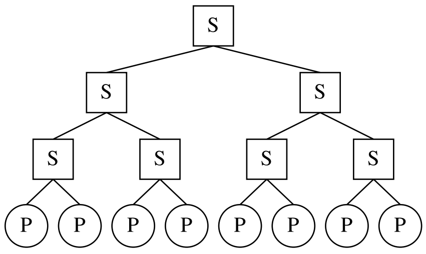


Figure 7.6. Simple binary tree network with processors (P) and switches (S).

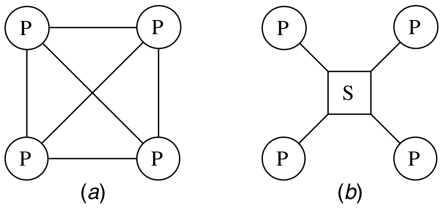


Figure 7.7. LAN modeled as fully connected processors (*a*) or with a switch (*b*).

represents either a processor or a switch. Note that there is still an *implicit* switch within each processor, as was established during the discussion of static networks in Section 2.2.1 (Figure 2.5). With this simple extension, many topologies can be modeled more accurately. For example, in a binary tree network only the leaves are processors; all other vertices are switches (see Figure 7.6).

Another example is a cluster of workstations connected through the switch of a LAN (local area network). In the undirected topology graph, such a network must be represented as fully connected (shown in Figure 7.7(*a*)). Thus, every processor has its own link with every other processor, reducing the modeled impact of contention. In the real system each processor is connected via one link to the switch, which is shared by all communications to and from this processor (Figure 7.7(*b*)).

Moreover, a switch vertex can even be utilized to model the bottleneck caused by the interface that connects a processor to the network, that is, end-point contention (see Section 7.1.1). Imagine one processor in a two-dimensional processor mesh (Section 2.2.1); that is, 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 Figure 7.8 reflects this situation, which, for example, is encountered in the Intel Paragon (Culler and Singh [48]).

Albeit there is already an implicit switch within each processor, as defined for the static network model in Section 2.2.1, Figure 2.5, this switch does not reflect the bottleneck of a network interface, as its form of interaction with the processor is not specified. For what it's worth, it must be assumed as having an unlimited bandwidth, thus being contention free.

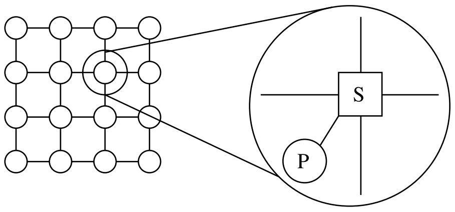


Figure 7.8. Switch used to model network interface bottleneck.

Many dynamic networks, for instance, the butterfly in Figure 2.10, can be described using the switch vertex. Some networks are in fact nonblocking (Cosnard and Trystram [45]), that is, contention free. In this case, the whole network is modeled by encapsulating it in one ideal switch to which each processor is connected with one link. This method was already used earlier for a LAN based cluster, illustrated in Figure 7.7(*b*). Another application of this method is the approximation of a network, for instance, when the impact of contention is negligible or the description of the network is too complex. Compared to a fully connected graph, this approach still offers the advantage of capturing end-point contention, that is, contention for the network interface. In fact, this approach is almost identical to the one-port model (Beaumont et al. [19]) but has the advantage that it is more flexible, because it can, besides full duplex ports, also represent half duplex ports, using directed or undirected edges, respectively.

Topology Graph Based on the foregoing considerations, the topology model is defined in the following.

Definition 7.3 (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 link set $\mathbf{L}$, that is, $\mathbf{L} = \mathbf{D} \cup \mathbf{H}$, and an element of this set is denoted by L, $L \in \mathbf{L}$. The weight $b(L)$ associated with a link $L \in \mathbf{L}$, $b : \mathbf{L} \rightarrow \mathbb{Q}^+$, represents its* relative data rate.

The topology graph is an unusual graph in that it integrates two types of edges—directed edges and hyperedges. Thereby it is able to address the previously discussed shortcomings of the undirected topology graph (Definition 7.1): it reflects directed links, busses, and switches (through the new vertex type). A hyperedge represents a bus or, if it is only incident on two network vertices, a half duplex link. A full duplex link is represented by two counterdirected edges and a unidirectional communication link is modeled by one directed link in the corresponding direction. For edge scheduling, the type of the edge is irrelevant (see Section 7.3); the only relevant consideration is which edges are involved in a communication. Thus, the different edge types are only relevant for the routing of a communication (see Section 7.2.2).

It is assumed that a switch is ideal—that is, there is no contention within the switch. All communications can take every possible route inside the switch—that is, a communication can arrive at any entering edge and leave at any outgoing edge, with the obvious exception of the edge from where it came. Essentially, a switch vertex behaves in the same way as a processor in the undirected topology graph. In fact, a switch can be treated as a processor on which tasks cannot be executed. Or, seen the other way around, a processor is the combination of a switch with a processing unit. This interpretation is used in the undirected topology graph (Figure 2.5, Section 2.2.1). A communication between two nonadjacent processors is routed through other processors, which therefore act as switches.

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).

Also, heterogeneity, in terms of the links' transfer speeds, is captured by the topology graph through the association of relative data rates with the links. In comparison with the introduced processor heterogeneity in Section 6.3, a distinction between consistent and inconsistent heterogeneity does not appear to be sensible. The transfer rate of a link does not depend on the type of data and is thus consistent. In contrast, the type of computation might very well have an impact on the execution speed, which can result in inconsistent heterogeneity. Of course, for homogeneous systems, the relative data rate is set to 1 or simply ignored.

Examples for networks modeled with the topology graph were already given in Figures 7.4 and 7.6–7.8. A topology graph containing directed edges and hyperedges is illustrated in Figure 7.9. It represents 8 dual processor systems, that is, 16 processors, in a dedicated LAN. In each dual system the processors and a network interface (here modeled as a switch) are grouped around one bus. Every dual system is connected to a central switch, representing the LAN, with a full duplex link (the counterdirected edges). Communication between the two processors of each system is performed via the bus, while communication between processors of different systems goes through the central switch. At any time only one of the two processors of a system can communicate, since all communications, local or remote, utilize the bus. Together with a heterogeneous setting of the links' data rates—a system bus is usually faster than a LAN—this topology graph reflects accurately the behavior of such dual processor clusters.

It should be noted that in comparison with a fully connected graph (the only sensible way to model such a network in the undirected topology graph), the number of vertices increases—from 16 to 25—but the number of edges is significantly reduced from 120 (see Eq. (3.2)) to 24.

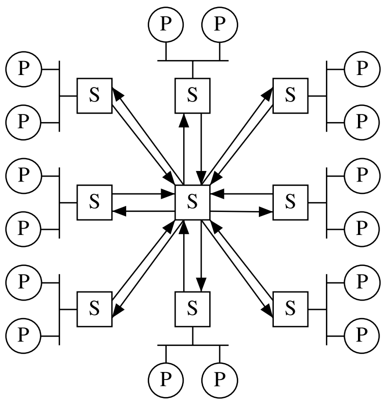


Figure 7.9. Example of topology model: 8 dual processors connected via a switched LAN and full duplex links.

7.2.2 Routing

In Section 7.2.1, an accurate model of the network topology was defined. However, the network's behavior is also affected by the routing of communications. To schedule edges on communication links, it must be known which links are involved in an interprocessor communication and how.

Essentially, this is described by the routing algorithm and the policy of the target system's network. The routing algorithm selects the route through the network, that is, the sequence of links that are employed for the communication, and the routing policy determines how these links are used.

In the next paragraphs the relevant aspects of routing will be reviewed from the perspective of edge scheduling. A general discussion of routing can be found in Culler and Singh [48] and Kumar et al. [108].

Static Versus Adaptive Routing A routing algorithm is *static*, or *nonadaptive*, if the route taken by a message is determined solely by its source and destination regardless of other traffic in the network. *Adaptive* routing algorithms allow the route for a message/packet to be influenced by traffic it meets along the way. Clearly, it is very complex to model adaptive routing in contention aware scheduling. To accurately simulate the effect of adaptive routing in the topology graph, the exact state of the network at every time instance must be known. Yet, the aim of contention aware scheduling is to avoid contention in the network through an appropriate schedule. Thus, with such a schedule, routes should seldom be adapted. Moreover, adaptive routing is not widely used in parallel machines (one of the few exceptions is the Cray T3E, but not the Cray T3D), due to its higher complexity and possible disadvantages (Culler and Singh [48]).

It is important to understand that static routing does not imply that every message from a given source to a given destination takes the same route. If multiple routes exist between two communication partners, mechanisms like randomization or round robin can choose one of the alternatives, yet not influence the traffic. As these routing techniques are independent of the traffic, they can be integrated into the routing algorithm for the topology graph.

Switching Strategy The next routing aspect to be considered is the switching strategy. Different strategies arise from the circumstance that data to be transfered in a network is split into packets. Two switching strategies can be distinguished: circuit switching and packet switching. 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. As a consequence, packets of the same message might be transfered over different routes. In scheduling, a communication associated with an edge is regarded in its entirety and its possible separation into packets is not reflected. Thus, edge scheduling must assume a circuit switched network or—as an alternative interpretation—a static packet switched network, where all packets of a message take the same route between two processors.

Store-and-Forward and Cut-Through Routing If a communication route involves more than one link, the routing of a message can be handled in two different ways: store-and-forward or cut-through routing. In *store-and-forward* routing the data of a communication is stored at every station (in the topology graph this is a processor or switch) on the path until the entire message, or packet, has arrived and is subsequently forwarded to the next station. In *cut-through* routing, a station immediately forwards the data, inflicting only a small routing delay (more on routing delays later), to the next station on the path—the message "cuts through" the station. By showing the transfer of one communication in a linear network of four processors in Figure 7.10, it is illustrated that store-and-forward routing has a much higher latency than cut-through routing, as the transfer time is a function of the number of "hops" (i.e., transitions between links). Store-and-forward routing is used in wide area networks and was used in several early parallel computers, while more modern parallel systems employ cut-through routing, for example, Cray T3D/T3E and IBM SP-2 (Culler and Singh [48]).

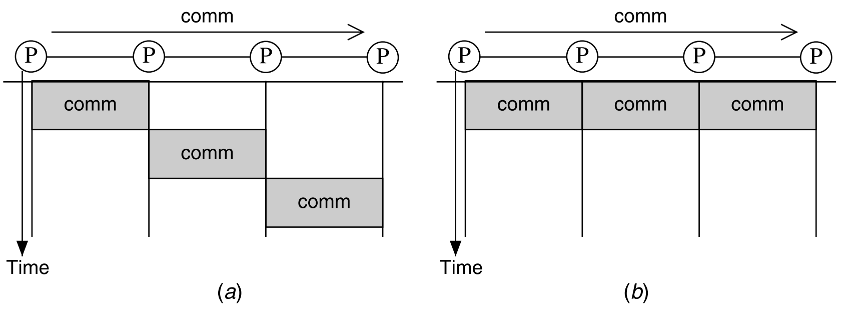


Figure 7.10. Store-and-forward routing (*a*) versus cut-through routing (*b*).

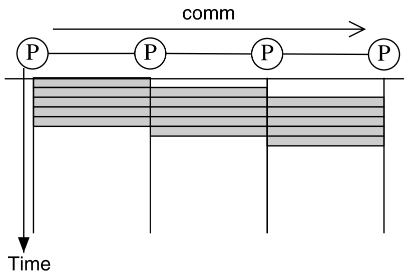


Figure 7.11. Packet based store-and-forward routing is similar to cut-through routing.

Still, for those parallel systems that use store-and-forward, assuming cut-through routing in edge scheduling is often a good approximation. Due to the limited capacity of communication buffers in the stations, store-and-forward routing works packet based. If a message consists of a sufficiently large number of packets, store-and-forward routing behaves approximately like cut-through, with only a packet size dependent delay per hop, as illustrated in Figure 7.11.

Routing Delay per Hop With every hop that a message or packet takes along its route through the network, a delay might be introduced. In cut-through routing this delay is typically very small, that is, on the order of some network cycles (e.g., Cray T3E has a hop delay of 1 cycle; Culler and Singh [48]). The store-and-forward hop delay is essentially the transfer time of one packet over one link. If the packet is small relative to the message size, it can be regarded as nonsubstantial. For this reason, the hop delay is neglected for simplicity in edge scheduling, but the next section also shows that 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. It is carried out by the involved processors, the switches of the network, or both. In order to obtain as accurate a result as possible, contention aware scheduling ought to reflect the routing algorithm of the target system. The approach that is obviously best suited for this aim is the projection of the target system's routing algorithm onto the model of its network. Since the network is represented by the topology graph (Definition 7.3), and a graph is the common representation form of a communication network, this is straightforward in most cases. Thus, it is proposed to employ the target system's own routing algorithm in contention aware scheduling. If the represented network should employ adaptive routing, the default behavior (i.e., the behavior without traffic conflicts) is reproduced. 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.

It is important to note that the routing approach of the algorithms BSA and MH (Section 7.1.2) is exactly the other way around: they determine the route according to their own routing algorithms. Hence, their produced schedules can only be accurate and efficient if the target parallel system can be directed to use the computed routes. However, that is rarely the case in a generic parallel system, especially at the application level (Culler and Singh [48]).

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. An example is dimension ordered routing in a multidimensional mesh (Culler and Singh [48], Kumar et al. [108]). A message is first routed along one dimension, then along the second dimension, and so on until it reaches its destination.

Given the graph based representation of the network, finding a shortest path can be accomplished with a BFS (breadth first search—Algorithm 1). 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.

Shortest Path in Topology Graph with BFS 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 (in the `for` loop, Algorithm 1). As this is already defined for directed and undirected edges (Section 3.1), it only remains to define adjacency for hyperedges.

Definition 7.4 (Adjacency—Hyperedge) *Let* $\mathbf{V}$ *be a finite set of vertices and* $\mathbf{H}$ *a finite set of hyperedges. A vertex* $u \in \mathbf{V}$, $u \in H$, $H \in \mathbf{H}$ *is adjacent to all vertices* $v \in H\text{-}u$ *and all vertices* $v \in H\text{-}u$ *are adjacent to* u*. The set* $\bigcup_{H \in \mathbf{H}: u \in H} H\text{-}u$ *of all vertices adjacent to* $u \in \mathbf{V}$ *is denoted by* $\mathbf{adj}(u)$.

Now in the topology graph, the total set of all vertices adjacent to a given vertex u is the union of the two 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 that is valid for all networks. On that account, the routing complexity shall be denoted generically by $O(\textit{routing})$.

The algorithm for routing in regular networks 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 (Figure 7.7(*b*)). In a topology graph for a mesh network of any dimension (Section 2.2.1), 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; for example, in a table, $O(\textit{routing})$ is just the complexity of the length of the route. For example, in a ring network the length of a route (i.e., the number of links) is $O(\mathbf{P})$; hence, $O(\textit{routing}) \geq O(\mathbf{P})$.

The BFS used to reflect minimal routing, has linear complexity for directed and undirected graphs, $O(\mathbf{V}+\mathbf{E})$ (Section 3.1.2). 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 considered only once and not one time for every incident vertex.

A final word regarding the number of network vertices. In a dynamic network, switches can outnumber the processors. Yet, in practice the number of switches is at most $O(\mathbf{P}^2)$—hence $O(\mathbf{N}) \leq O(\mathbf{P}^2)$—because a higher number of switches would make such a network more expensive than a fully connected network.

Other Aspects Other aspects of routing, such as flow control, deadlock handling, and buffer sizes (Culler and Singh [48]), are not relevant for edge scheduling. Also, packet dropping due to congestion, for example, in TCP/IP based networks, is not reflected, once more, because the separation into packets is unregarded. In any case, the effect of packet dropping should be significantly reduced when a program is executed according to a contention aware schedule. The contention is already taken care of by the scheduling algorithm and should not arise often during execution.

7.2.3 Scheduling Network Model

Sections 7.2.1 and 7.2.2 analyzed, from the perspective of scheduling, the topologies of communication networks and their routing policies. It remains merely to summarize this analysis to establish a network model based on the topology graph (Sinnen and Sousa [172, 179]).

Definition 7.5 (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 7.3). The routing algorithm carried out on the topology graph is that of the represented network (or its closest approximation). Furthermore, routing has the following properties:*

1. Static. *Routing in the network is static; the selected route does not depend on the state of the network. If the represented network employs adaptive routing, the default behavior (i.e., the behavior without traffic conflicts) is reproduced.*
2. Circuit Switching. *The communication route between two processors of the network is the same for the entire message.*
3. Cut-through. *A message cuts through the communication stations (processor or switch) on its route; that is, all links of the route are utilized at the same time.*
4. No Routing Delay per Hop. *No delay is inflicted by the transition between links on the communication route.*

Edge scheduling, which is investigated in the next section, employs this model for the representation of the network.

7.3 EDGE SCHEDULING

This section analyzes the theoretical background of the fundamental technique to achieve contention awareness in scheduling: edge scheduling (Sinnen and Sousa [172, 179]). The utilization of edge scheduling in contention aware heuristics and the interaction with task scheduling are treated in the next section.

Edge scheduling is the adoption of the task scheduling principle in the communication domain. In contrast to the classic model of the target system (Definition 4.3), 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.

To schedule an edge on a link, a start time and a finish time are assigned to the edge. The corresponding notations from node scheduling (Section 4.1) are adopted.

Definition 7.6 (Edge Start, Communication and Finish Time on Link) *Let $G = (\mathbf{V}, \mathbf{E}, w, c)$ be a task graph and $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$ a communication network according to Definition 7.5.*

The start time $t_s(e, L)$ of an edge $e \in \mathbf{E}$ on a link $L \in \mathbf{L}$ $(\mathbf{L} = \mathbf{D} \cup \mathbf{H})$ is the function $t_s : \mathbf{E} \times \mathbf{L} \to \mathbb{Q}_0^+$.

The communication time of e on L is

$$\varsigma(e, L) = \frac{c(e)}{b(L)}. \tag{7.1}$$

The finish time of e on L is

$$t_f(e, L) = t_s(e, L) + \varsigma(e, L). \tag{7.2}$$

So the time during which link L is occupied with the transfer of the data associated with the edge e is denoted by the communication time $\varsigma(e, L)$. As links might have heterogeneous data rates (Definition 7.3), the communication time is a function of the edge e and the link L. This also brings a change to the interpretation of the edge cost $c(e)$ (Definition 4.4).

Definition 7.7 (Communication Cost—Network Model) *Let $G = (\mathbf{V}, \mathbf{E}, w, c)$ be a task graph and $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$ a communication network.*

The communication cost $c(e)$ of edge e is the average *time the communication represented by e occupies a link of* $\mathbf{L}$ *for its transfer.*

Like a processor, a link is exclusively occupied by one edge.

Condition 7.1 (Link Constraint) *Let $G = (\mathbf{V}, \mathbf{E}, w, c)$ be a task graph and $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$ a communication network. For any two edges $e, f \in \mathbf{E}$ scheduled* *on link $L \in \mathbf{L}$:*

$$\begin{aligned} & t_s(e,L) \le t_f(e,L) \le t_s(f,L) \le t_f(f,L) \\ \text{or} \quad & t_s(f,L) \le t_f(f,L) \le t_s(e,L) \le t_f(e,L). \end{aligned} \tag{7.3}$$

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 node duplication, Section 6.2). The scheduling of an edge along the links of the route is significantly determined by the routing in the network.

7.3.1 Scheduling Edge on Route

In Section 7.2 the network model of the target system was carefully developed in a way that encapsulates and hides all the details about the network topology from edge scheduling. For any communication between two distinct processors, edge scheduling only sees the communication route in the form of an ordered list of involved links. This list is provided by the routing algorithm of the network model, when it is given the two communicating processors.

Definition 7.8 (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}, $P_{\text{src}}, P_{\text{dst}} \in \mathbf{P}$, 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 $i = 1, \ldots, l$.*

In Definition 7.8 the route depends only on the source and the destination processor of a communication and not on the traffic state of the network. This conforms to the network model's property of static routing (Definition 7.5). Since the network model also supposes circuit switching, the entire communication is transmitted on the returned route. Hence, the edge *e* of the communication is scheduled on each link of the route. The type of link (e.g., half or full duplex) is irrelevant to edge scheduling.

Data traverses the links in the order of the route, which must be considered in the scheduling of the edges.

Condition 7.2 (Causality) *Let $G = (\mathbf{V}, \mathbf{E}, w, c)$ be a task graph and $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$ a communication network. For the start and finish times of edge $e \in \mathbf{E}$ on the links of the route $R = \langle L_1, L_2, \ldots, L_l \rangle$, $R \in TG$,*

$$t_f(e, L_{k-1}) \le t_f(e, L_k), \tag{7.4}$$

$$t_s(e, L_1) \le t_s(e, L_k), \tag{7.5}$$

for $1 < k \le l$.

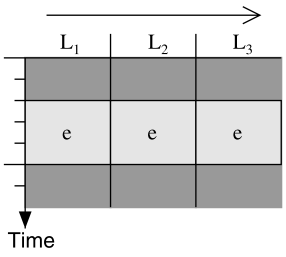


Figure 7.12. Edge scheduling on a route without contention.

Inequality (7.4) of the causality condition reflects the fact that communication does not finish earlier on a link than on the link's predecessor. For homogeneous links, inequality (7.5) is implicitly fulfilled by Eq. (7.4), but for heterogeneous links this condition is important as will be seen later.

The edge scheduling must further comply with the other routing properties of the network model in Definition 7.5, 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 link. Figure 7.12 illustrates this for a route consisting of three homogeneous links on which edge e is scheduled.

This example shows the trivial case of edge scheduling, where all links are homogeneous and idle. When contention comes into play (i.e., when a communication meets other communications along the route), edge scheduling must be more sophisticated. The scheduling times of an edge on the different links are strongly connected through the causality condition and the routing properties of the network model. If an edge is delayed on one link, the scheduling times of the edge on the subsequent links are also affected. Two different approaches must be examined for the determination of the scheduling times on a route with contention.

Consider the same example as in Figure 7.12, with the modification that link L_2 is occupied with the communication of another edge, say, e_{xy}, at the time e is scheduled.

Nonaligned Approach One solution to the scheduling of e is depicted in Figure 7.13(a). 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 link until it has finished on the previous link, that is, $t_s(e, L_3) = t_s(e, L_2)$ and $t_f(e, L_3) = t_f(e, L_2)$. On L_1, e is scheduled at the same time as without contention (Figure 7.12). This approach is referred to as nonaligned.

Aligned Approach Alternatively, e can be scheduled later on all links; that is, 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 Figure 7.13(b), seems to better reflect cut-through routing, where communication takes place on all involved links at the same time. Scheduling the edge at different times, as done in the first approach (Figure 7.13(a)), seems to imply the storage of

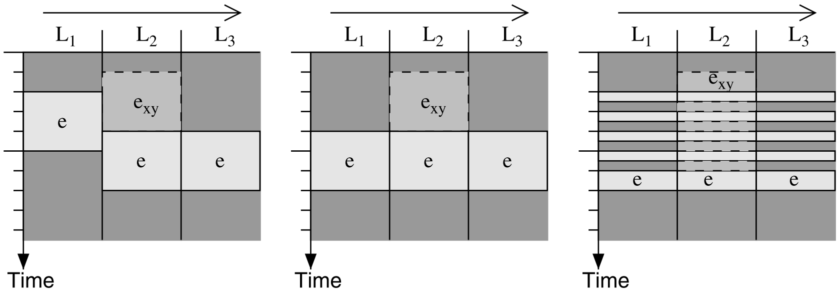


Figure 7.13. Edge scheduling on a route with contention.

at least parts of the communication at the network vertices—a behavior similar to store-and-forward routing.

Packet View However, communication in parallel systems is packet based (Section 7.2.2) and the real process of the message transfer is better illustrated by the Gantt chart of Figure 7.13(*c*) (assuming the usual fair sharing of the resources). Each of the symbolized packets performs cut-through routing and link L_2 is shared between both edges.

For several reasons, it is not the intention to go down to the packet level in edge scheduling: (1) the scheduling model would become more complicated; (2) new system parameters, like the packet size and arbitration policies, would be introduced; and (3) the scheduling of an edge would affect previously scheduled edges (e.g., in Figure 7.13(*c*), e_{xy} is affected by the scheduling of e).

But the packet view of the situation in Figure 7.13(*c*) illustrates that the nonaligned scheduling (Figure 7.13(*a*)) can be interpreted as a message based view of the situation in Figure 7.13(*c*). 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 Figure 7.13(*c*). 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 approximated as an indivisible communication.

Moreover, scheduling the edges aligned on a route means that idle intervals have to be found that are aligned across the entire route. In consequence, this approach is likely to produce many idle time slots on the links, since many solutions that are feasible under the causality condition (e.g., the solution of Figure 7.13(*a*)), cannot be employed. Also, employment of the insertion technique (Section 6.1) in edge scheduling could be virtually useless, as it can be expected that only a few of such idle channels exist. Nonaligned scheduling allows the successive treatment of each link, since the edge's start and finish times are only constrained by the scheduling on the foregoing links.

In conclusion, the nonaligned approach appears to better reflect the behavior of communication networks and its utilization in edge scheduling is adopted. Before edge scheduling is formally defined, the nonaligned approach must yet be analyzed for heterogeneous links.

Heterogeneous Links The causality condition must also be obeyed on heterogeneous links, that is, links with different data rates. Figure 7.14(*a*) 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 Eq. (7.4) of the causality Condition 7.2, e cannot finish earlier on L_2 than on L_1 and therefore the start of e on L_2 is delayed accordingly. As noted before, this delay seems to suggest a storage between the first and second link, but in fact it is the best approximation of the real, packet based communication illustrated in Figure 7.14(*b*). This explains also why e starts and finishes on L_3 at the same times as on L_1, which is in concordance with Eq. (7.5).

Another possibility, which seems more intuitive at first, is to start e on L_3 later, namely, at the same time it starts on L_2. This would correspond to the seemingly more intuitive $t_s(e, L_{k-1}) \leq t_s(e, L_k)$ as the causality equation (7.5). Then, however, e would finish on L_3 later than on L_2 and L_1. Consequently, this implies that having a faster link between two other links *retards* the communication. This is certainly not realistic and also does not correspond to the packet view in Figure 7.14(*b*). Thus, the causality Condition 7.2 as defined ensures a realistic scheduling of the edge in this example.

The next example further emphasizes the importance of inequality (7.5) of the causality Condition 7.2 for heterogeneous links. Consider the same example as before, now with the last link slower than the two others ($b(L_3) = b(L_1)/2 = b(L_2)/4$), which is illustrated in Figure 7.14(*c*). Without Eq. (7.5), 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, because the communication would be active on L_3 before it even entered into the network. Due to the lower data rate of L_3, the correct behavior, as shown in Figure 7.14(*c*), is not implicitly enforced by inequality (7.4) as in all other examples. Note 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

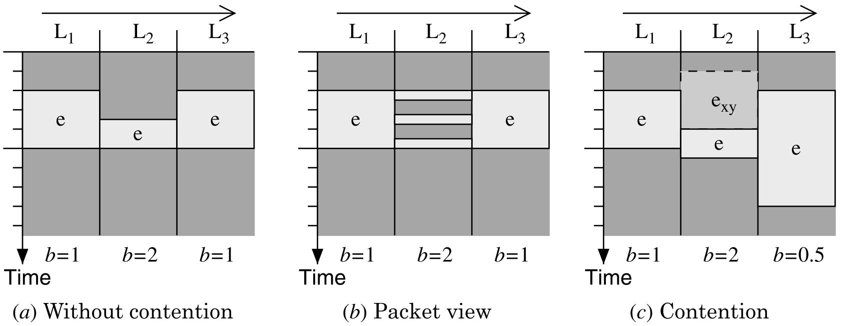


Figure 7.14. Edge scheduling on a heterogeneous route.

of a real network, where a fast link (L_2) can compensate contention in relation to slower links on the route.

7.3.2 The Edge Scheduling

In the following, the equations are formulated for edge scheduling. As stated earlier, the nonaligned approach allows determination of the start and finish times successively for each link on the route. It is not necessary to consider all links at the same time to find an idle time channel.

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 6.1), a scheduling condition can be formulated that integrates both the end technique (Definition 5.2) and the insertion technique (Definition 6.1).

Condition 7.3 (Edge Scheduling Condition) *Let $G = (\mathbf{V}, \mathbf{E}, w, c)$ be a task graph, $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$ a communication network, and $R = \langle L_1, L_2, \ldots, L_l \rangle$ the route for the communication of edge $e_{ij} \in \mathbf{E}$, $n_i, n_j \in \mathbf{V}$. Let $[A, B]$, $A, B \in [0, \infty]$, be an idle time interval on L_k, $1 \le k \le l$, that is, 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*

$$B - A \ge \varsigma(e_{ij}, L_k), \tag{7.6}$$

$$B \ge \begin{cases} t_f(n_i) + \varsigma(e_{ij}, L_1) & \text{if } k = 1 \\ \max\{t_f(e_{ij}, L_{k-1}), t_s(e_{ij}, L_1) + \varsigma(e_{ij}, L_k)\} & \text{if } k > 1 \end{cases}. \tag{7.7}$$

Inequality (7.6) ensures that the time interval is large enough for e_{ij}. Furthermore, inequality (7.7) guarantees that the constraint of the origin node's finish time or the causality Condition 7.2 can be fulfilled for e_{ij} within this interval. On the first link ($k = 1$) of the route R, 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 ($k > 1$) of R, the scheduling of e_{ij} has to comply with the causality Condition 7.2.

A time interval, obeying Condition 7.3, can be searched successively for each of the links on the route. The time interval can be between edges already scheduled on the link (insertion technique) or after the last edge (end technique); that is, A is the finish time of the last edge and $B = \infty$. Obviously, the scheduling technique (end or insertion) should be identical across the entire route.

For a given time interval, the start time of e_{ij} on L_k is determined as follows.

Definition 7.9 (Edge Scheduling) *Let $[A, B]$ be an idle time interval on L_k adhering to Condition 7.3. The start time of e_{ij} on L_k is*

$$t_s(e_{ij}, L_k) = \begin{cases} \max\{A, t_f(n_i)\} & \text{if } k = 1 \\ \max\{A, t_f(e_{ij}, L_{k-1}) - \varsigma(e_{ij}, L_k), t_s(e_{ij}, L_1)\} & \text{if } k > 1 \end{cases}. \tag{7.8}$$

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 7.3.1.

A last remark regarding hop delays: it should be easy to see that the introduction of such a hop delay is straightforward. A simple addition of a delay value to the calculation of the start times on the links L_k, $k > 1$, is sufficient. Of course, the conditions on the idle time interval must be adapted, too (Condition 7.3). Moreover, if the hop delay equals the time spent on the previous link, even store-and-forward routing can be simulated.

7.4 CONTENTION AWARE SCHEDULING

The network model and edge scheduling have been investigated in the previous sections. Based on this, contention aware task scheduling is analyzed in this section. The basics are presented next, followed by the proposal of contention aware scheduling heuristics.

7.4.1 Basics

The new, more realistic, target system model is defined as follows.

Definition 7.10 (Target Parallel System—Contention Model) *A target parallel system $M_{TG} = (TG, \omega)$ consists of a set of possibly heterogeneous processors* $\mathbf{P}$ *connected by the communication network $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$, according to Definition 7.5. The processor heterogeneity, in terms of processing speed, is described by the execution time function ω. This system has Properties 1 to 4 of the classic model of Definition 4.3:*

1. *Dedicated system*
2. *Dedicated processor*
3. *Cost-free local communication*
4. *Communication subsystem*

Two of the classic model's properties are substituted by the detailed network model: concurrent communication (Property 5) and a fully connected network topology (Property 6).

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 actual scheduling of the edges on the links was discussed in Section 7.3. It remains to establish a connection with task scheduling.

The integration of edge scheduling into task scheduling is simple, due to the careful formulation of the scheduling problem in Section 4.2. It is only necessary to redefine the total edge finish time (Definition 4.6).

Definition 7.11 (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. The finish time of $e_{ij} \in \mathbf{E}$, $n_i, n_j \in \mathbf{V}$, communicated from processor P_{src} to P_{dst}, $P_{\text{src}}, P_{\text{dst}} \in \mathbf{P}$, is*

$$t_f(e_{ij}, P_{\text{src}}, P_{\text{dst}}) = \begin{cases} t_f(n_i, P_{\text{src}}) & \text{if } P_{\text{src}} = P_{\text{dst}} \\ t_f(e_{ij}, L_l) & \text{otherwise} \end{cases} \tag{7.9}$$

where L_l is the last link of the route $R = \langle L_1, L_2, \ldots, L_l \rangle$ from P_{src} to P_{dst}, if $P_{\text{src}} \neq P_{\text{dst}}$.

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 3), 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. As before, a node must not start earlier than the incoming edge finishes communication. Figure 7.15 illustrates the precedence constraint with edge scheduling. It displays an integrated Gantt chart of the node and edge schedule of a two-node task graph on two processors connected by one link. The edge e_{AB} starts on the link after node A has finished (inequality (7.8), $k = 1$), and the execution of B is started immediately after the communication has terminated on the link (Eqs. (4.5) and (7.9)).

The time interval spanned by the edge's start time on the first link $t_s(e, L_1)$ and its finish time on the last link $t_f(e, L_l)$ of the route R corresponds to the communication delay (Definition 4.4) in the classic model. The interval is identical with the delay in the case of no contention. Furthermore, in this case an edge is scheduled on every link at the same time (cut-through routing) and the edge weight is defined to be the average time a link is occupied with the transfer (Definition 7.7). Hence, the edge weight in a task graph for the classic model is identical to the edge weight in a task graph for the contention model. Thus, the new system model has no impact on the

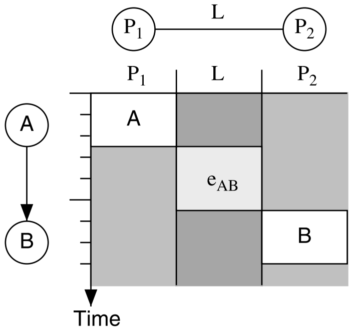


Figure 7.15. Precedence constraint with edge scheduling.

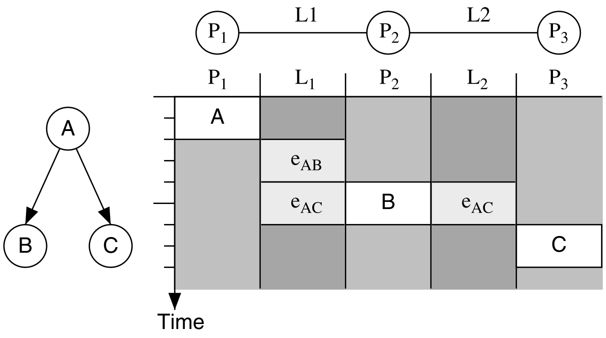


Figure 7.16. Contention aware scheduling: scheduling order of edges is relevant.

creation of the task graph. It only differs for heterogeneous target systems; but this case is not addressed in the classic model, anyway.

Contention Awareness When an edge is scheduled along its route, it might conflict with other, already scheduled edges on one or more links. As a result, the start of the edge is delayed on these links, which eventually results in a delayed total finish time of the edge. Thus, the contention is exposed in the extended transfer time of the edge. This delays the execution start of the destination node and can in the end result in a longer schedule. For instance, in Figure 7.16, showing the integrated schedule of the depicted task graph and homogeneous target system, edge e_{AC} is delayed on link L_1 due to contention, because e_{AB} already occupies L_1. This also delays the start of node C on P_3, as the communication arrives later. In the classic model, both communications are transferred at the same time so that node C starts at the same time as B. A scheduling heuristic "sees" the contention effect through the node's later DRT on the processors to which communication is affected by contention.

An interesting consequence of edge scheduling is that the order in which edges are scheduled is relevant. For example, the order of the outgoing edges of a node has an influence on the DRTs of the node's successors. This is desired, as it reflects the behavior of real systems. If edge e_{AC} in Figure 7.16 was scheduled before e_{AB}, the start times between node B and C would be swapped—that is, node C would start before node B.

7.4.2 NP-Completeness

Intuitively, scheduling is more complicated when considering contention, and in fact, the problem remains NP-complete (Dutot et al. [57]).

Theorem 7.1 (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 in the strong sense.*

Proof. 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 NP-complete problem 3-PARTITION (Garey and Johnson [73]) in polynomial time to C-SCHED. 3-PARTITION is NP-complete in the strong sense.

The 3-PARTITION problem is as follows. Given is a set **A** of $3m$ positive integer numbers a_i and a positive integer bound B such that $\sum_{i=1}^{3m} a_i = mB$ with $B/4 < a_i < B/2$ for $i = 1, \dots, 3m$. Can **A** be partitioned into m disjoint sets $\mathbf{A}_1, \dots, \mathbf{A}_m$ (triplets) such that each $\mathbf{A}_i$, $i = 1, \dots, m$, contains exactly 3 elements of **A**, whose sum is B?

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 3-PARTITION $\mathbf{A} = \{a_1, a_2, \dots, a_{3m}\}$, an instance of C-SCHED is constructed in the following way.

Task Graph G. The constructed task graph G is a fork graph as illustrated in Figure 7.17(a). It consists of one parent node n_x and $3m + 1$ child nodes $n_0, n_1, \dots, n_{3m}$. There is an edge e_i directed from n_x to every child node n_i, $0 \leq i \leq 3m$. The weights assigned to the nodes are $w(n_x) = 1$, $w(n_0) = B + 1$, and $w(n_i) = 1$ for $1 \leq i \leq 3m$. The edge weights are $c(e_0) = 1$ and $c(e_i) = a_i$ for $1 \leq i \leq 3m$.

Target System M_{TG}. The constructed target system $M_{TG} = ((\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b), \omega)$, illustrated in Figure 7.17(b), consists of $|\mathbf{P}| = m + 1$ identical, fully connected processors. Each processor P_i is connected to each other processor P_j through a half duplex link L_{ij}, which is represented by an undirected edge (a hyperedge incident on two vertices: $L_{ij} = H_{ij} = \{P_i, P_j\}$). Hence formally, $\mathbf{N} = \mathbf{P}$, $\mathbf{D} = \emptyset$, $\mathbf{H} = \bigcup_{i=1}^{m} \bigcup_{j=i+1}^{m+1} L_{ij}$, and, as all processors and links are identical, $\omega(n, P) = w(n)\ \forall P \in \mathbf{P}$ and $b(L_{ij}) = 1\ \forall L_{ij} \in \mathbf{H}$.

Time Bound T. The time bound is set to $T = B + 2$.

Clearly, the construction of the instance of C-SCHED is polynomial in the size of the instance of 3-PARTITION.

It is now shown how a schedule $\mathcal{S}$ is derived for C-SCHED from an arbitrary instance of 3-PARTITION $\mathbf{A} = \{a_1, a_2, \dots, a_{3m}\}$, which admits a solution

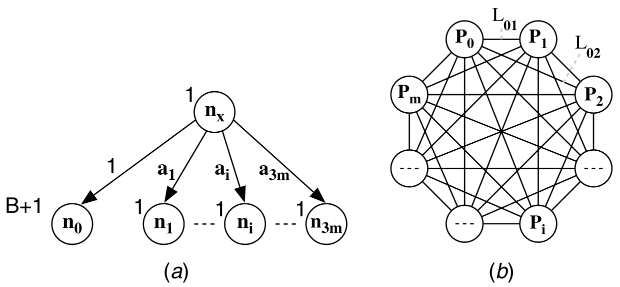


Figure 7.17. The constructed fork DAG (a) and target system (b).

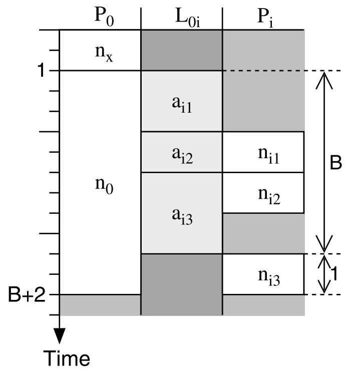


Figure 7.18. Constructed schedule of $n_x, n_0, n_{i1}, n_{i2}, n_{i3}$ and e_{i1}, e_{i2}, e_{i3} on P_0, P_i and L_{0i}.

to 3-PARTITION: let $\mathbf{A}_1, \ldots, \mathbf{A}_m$ (triplets) be m disjoint sets such that each $\mathbf{A}_i$, $i = 1, \ldots, m$, contains exactly 3 elements of $\mathbf{A}$, whose sum is B.

Nodes n_x and n_0 are allocated to the same processor, which shall be called P_0. The remaining nodes $n_1, \ldots, n_{3m}$ are allocated in triplets to the processors $P_1, \ldots, P_m$. Let a_{i1}, a_{i2}, a_{i3} be the elements of $\mathbf{A}_i$. The nodes n_{i1}, n_{i2}, n_{i3}, corresponding to the elements of triplet $\mathbf{A}_i$, are allocated to processor P_i. Their entering edges e_{i1}, e_{i2}, e_{i3}, respectively, are scheduled on L_{0i} as early as possible in any order. The resulting schedule is illustrated for P_0,P_i and L_{0i} in Figure 7.18.

What is the length of this schedule? The time to execute n_x and n_0 on P_0 is $w(n_x) + w(n_o) = B + 2 = T$. On each link L_{0i} the three edges corresponding to the triplet $\mathbf{A}_i$ take B time units for their communication (Figure 7.18). The first communication starts as early as possible, that is, after 1 time unit when n_x finishes. After the last communication has finished, only one node remains to be executed on processor P_i; the other two nodes are already executed during the communication (Figure 7.18). This is guaranteed by the fact that a_i is a positive integer and $w(n_i) = 1$ for $1 \leq i \leq 3m$. The execution of this last node takes 1 time unit and, hence, each processor P_i finishes at $1 + B + 1 = T$.

Thus, a feasible schedule $\mathcal{S}$ 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 $sl(\mathcal{S}) \leq T$. It will now be shown that $\mathcal{S}$ is necessarily of the same kind as the schedule constructed previously.

Nodes n_x and n_0 must be scheduled on the same processor, say, P_0, since otherwise the communication e_0 is remote and takes one time unit. As a consequence, the earliest finish time of n_0 would be $w(n_x) + c(e_0) + w(n_0) = B + 3$, which is larger than the bound T.

Since processor P_0 is fully occupied with these two nodes until $B + 2$, all remaining nodes have to be executed on the other processors. That means that all corresponding communications are remote.

Between the computation of n_x and the computation of the last node on each processor (both take one time unit) exactly B time units are available for the communication on the links, in order to stay below the bound $T = B + 2$. The total communication time of all edges is $\sum_{i=1}^{3m} c(e_i) = mB$. Since there are m processors connected to P_0 by m dedicated links, the available communication time B per link must be completely used. Hence, each link must be fully occupied with the communication between time instances 1 (when n_x finishes) and $B + 1$ (when the last node must start execution). The condition $B/4 < a_i < B/2$ (i.e., $B/4 < c(e_i) < B/2$) enforces that these B time units are occupied by exactly three edges; any other number of edges cannot have a total communication time of B. This distribution of the edges on the links corresponds to a solution of 3-PARTITION. □

$\alpha|\beta|\gamma$ Notation To characterize the problem of scheduling under the contention model with the $\alpha|\beta|\gamma$ classification (Section 6.4.1), the following extensions are proposed for the α and β fields.

The distinguishing aspect from the problems discussed in Section 6.4 is the contention awareness achieved through edge scheduling. This is symbolized by the extension -*sched* in the β field, immediately after the specification of the communication costs (e.g., c_{ij}-*sched*).

Furthermore, the complexity of a problem also depends on the topology graph of the parallel system. Hence, the α field is extended with a third subfield that specifies the communication network, similar to the description of the precedence relations of the task graph:

- $\circ$. The processors of the parallel system are fully connected.
- *net.* The parallel system has a communication network, modeled by a topology graph $TG = (\mathbf{N}, \mathbf{P}, \mathbf{D}, \mathbf{H}, b)$.
- *snet.* The parallel system has a static communication network, modeled as an undirected graph $UTG = (\mathbf{P}, \mathbf{L})$.
- *star.* The parallel system consists of a star network, where each processor is connected to a central switch via two counterdirected links (full duplex). This network corresponds to the one-port model (Beaumont et al. [19]), as discussed in Section 7.1.1.
- *star-h.* The parallel system consists of a star network, where each processor is connected to a central switch via an undirected edge (half duplex), as illustrated in Figure 7.7(*b*).

With these extensions, the general problem of scheduling under the contention model is specified by $P, net|c_{ij}\text{-}sched|C_{\max}$ for homogeneous processors.

Just before Theorem 7.1 and its proof, it was mentioned that scheduling with contention is intuitively more complicated than scheduling under the classic model. The proof provides evidence. It showed the NP-completeness of the special case $P|fork, c_{ij}\text{-}sched|C_{\max}$. While the corresponding problem under the classic model, $P|fork, c_{ij}|C_{\max}$, is also NP-complete (Section 6.4.3), the scheduling of the particular problem utilized in the forgoing proof is not. One can construct an optimal schedule in polynomial time for the fork graph of Figure 7.17 on the equivalent classic target system $M_{\text{classic}} = (\mathbf{P}, \omega)$. $M_{\text{classic}} = (\mathbf{P}, \omega)$ corresponds to M_{TG} by consisting of $|\mathbf{P}| = m + 1$ identical processors, that is, $\omega(n, P) = w(n) \forall P \in \mathbf{P}$. The construction of the schedule is described in the following.

With the same argument as in the previous proof, nodes n_x and n_0 must be scheduled on the same processor, say, P_0. Now all other nodes are scheduled on the remaining processors, by putting any three nodes on each processor. Let the nodes scheduled on P_i be n_j, n_k, n_l, $1 \le j, k, l \le 3m$. Since there is no contention for the communication resources, all data sent from n_x to n_j, n_k, n_l is transmitted at the same time immediately after n_x finishes. In other words, the edges e_j, e_k, e_l are communicated from P_0 to P_i at the same time. With $c(e_i) = a_i < B/2$, $1 \le i \le 3m$, the total communication time of these edges is less than $B/2$. It follows for the finish time of processor P_i that $f_t(P_i) < w(n_x) + B/2 + w(n_j) + w(n_k) + w(n_l) = 4 + B/2$ (Figure 7.19).

It holds that $4 + B/2 \le B + 2$ for $B \ge 4$. Furthemore, in order that B and a_i, $i = 1, \dots, 3m$, comply with the constraints of the 3-PARTITION problem, B must be an integer $B \ge 3$ (otherwise there can be no solution to the 3-PARTITION instance). If $B = 3$ then $a_i = 1, i = 1, \dots, 3m$, in order to comply with $B/4 < a_i < B/2$ and the fact that a_i is a positive integer; but then the finish time of processor P_i is $t_f(P_i) = 5 = B + 2 = T$. Hence, the constructed schedule under the classic model (without contention) has the schedule length T and is therefore optimal. Furthermore, the construction clearly takes only polynomial time.

A similar complexity result is obtained by Beaumont et al. [19], where the NP-completeness of scheduling under the one-port model is proved by demonstrating the NP-completeness of $P\infty, star|fork, c_{ij}\text{-}sched|C_{\max}$. Under the classic model, the corresponding problem $P\infty|fork, c_{ij}|C_{\max}$ is tractable (Section 6.4.3). Finally, Sinnen and Sousa [172, 179] show that $P\infty, star\text{-}h|fork, c_{ij}\text{-}sched|C_{\max}$ is also NP-complete.

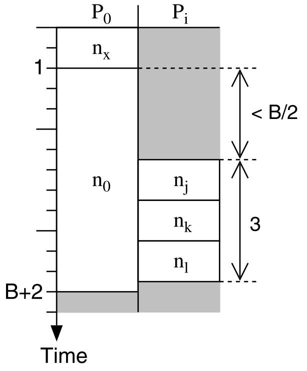


Figure 7.19. Schedule of n_x, n_0, n_j, n_k, n_l under classic model, that is, without contention.

7.5 HEURISTICS

Scheduling under the contention model essentially requires only one change compared to the classic model: an edge is scheduled on the network links in order to determine the delay of the corresponding interprocessor communication. In this section it is investigated how task scheduling heuristics can be adapted to the contention model. This is done exemplarily with list scheduling in its general form—static or dynamic (Section 5.1). Having contention aware list scheduling is crucial, due to list scheduling's significant role in task scheduling, and allows one to transform many of the existing heuristics into contention aware algorithms.

Other algorithms that are not list scheduling based can easily be enhanced with contention awareness based on the same procedures as described in the following. For example, in Sinnen [172] BSA (Section 7.1.2) was adapted to contention awareness as discussed in this chapter. Moreover, Section 7.5.3 briefly outlines how clustering can be adapted to the contention model.

The only obvious exception is node duplication. To integrate this technique, the duplication of the respective communications of duplicated nodes must first be investigated.

7.5.1 List Scheduling

List scheduling—static (Algorithm 10) or dynamic (Algorithm 11)—is only affected by the contention model and the associated edge scheduling in two aspects: the determination of the DRT and the scheduling of a node.

Scheduling a Node At each step of list scheduling, a partial feasible schedule is extended with one free node. In the contention model, a feasible schedule also comprises the schedule for all edges whose communication is transferred between processors. Thus, the scheduling of a free node implies the preceding scheduling of its entering edges. Algorithm 21 describes how a node n_j is scheduled in the contention model. For each of n_j's predecessors that is not executed on the same processor as n_j, the communication route is determined and e_{ij} is scheduled on this route. Only when all remote communications have been allocated to the links, is the node scheduled.

Algorithm 21 Scheduling of a Node n_j on Processor P in Contention Model

```
Require: $n_j$ is a free node
  for each $n_i \in \mathbf{pred}(n_j)$ in a deterministic 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 $R$ (Definition 7.9)
    end if
  end for
  schedule $n_j$ on $P$
```

Note that each leaving edge e_{ij} of a node n_i is only scheduled during the same algorithm in which its destination node n_j is scheduled. There is no alternative, since for edge scheduling $proc(n_j)$ must be known, which is only determined when n_j is scheduled. Thus, the leaving edges are automatically scheduled in the scheduling order of their destination nodes, that is, the nodes' priority order.

Determining DRT Most scheduling heuristics determine the DRT of a node not only for one but for several processors, before the decision is taken about which processor the node is to be scheduled on. List scheduling with start time minimization (Section 5.1) is a prominent example. In the contention model, the DRT is calculated by temporarily scheduling the entering edges of a node. This is necessary to obtain the correct view of the communication times of the edges. Algorithm 22 expresses the procedure in the contention model. Basically, the edges are scheduled as if the node was to be scheduled on the respective processor. While doing so, the DRT is calculated and before the procedure terminates, the scheduled edges are removed from the links.

Algorithm 22 Determine $t_{\mathbf{dr}}(n_j, P)$ in Contention Model

```
Require: $n_j$ is a free node
  $t_{dr} \leftarrow 0$
  for each $n_i \in \mathbf{pred}(n_j)$ in a deterministic order do
    $t_f(e_{ij}) \leftarrow t_f(n_i)$
    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 $R$ (Definition 7.9)
      $t_f(e_{ij}) \leftarrow t_f(e_{ij}, L_l)$
    end if
    $t_{dr} \leftarrow \max\{t_{dr}, t_f(e_{ij})\}$
  end for
  remove edges $\{e_{ij} \in \mathbf{E} : n_i \in \mathbf{pred}(n_j) \wedge proc(n_i) \neq P\}$ from links
  return $t_{dr}$
```

Scheduling Order of Edges As discussed in Section 7.4, and illustrated in Figure 7.16, the scheduling order of edges has an impact on the DRTs of the destination nodes. For a scheduling algorithm it is crucial that the determination of the DRT is deterministic. A DRT that varies between repeated calculations for the same node and processor might even lead to an infeasible schedule. For this reason, Algorithms 21 and 22 require a deterministic 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. Yet, sorting the edges increases the complexity of the list scheduling heuristic, albeit not significantly. Node priorities and the complexity of list scheduling are discussed later.

Heuristics With the discussed two algorithms for the DRT calculation (Algorithm 22) and the scheduling of a node (Algorithm 21), the basic structure of list scheduling need not be altered for the contention model. The actual scheduling of a node or an edge—in Algorithms 21 and 22 this is performed by the statements "schedule …"—can be done either with the end technique or the insertion technique. The scheduling condition (Condition 6.1) and the edge scheduling condition (Condition 7.3) regulate the choice of an appropriate time interval for node and edge scheduling, respectively.

An Example For the illustration of contention aware list scheduling, the sample task graph (e.g., in Figure 6.25) is scheduled on a homogeneous linear network of three identical processors, as depicted in Figure 7.20. The chosen node order is identical with the order of the list scheduling example in Figure 5.1, namely, *a*, *b*, *c*, *d*, *e*, *f*, *g*, *i*, *h*, *j*, *k*. Nodes *a*, *b*, and *c* are scheduled as in the classic model, but the scheduling of *d* is affected by contention on L_1. Edge e_{ad} would arrive at time unit 7 at P_3, since L_1 is occupied with e_{ac} until time unit 4. Thus, *d* is scheduled earlier on P_1 and in

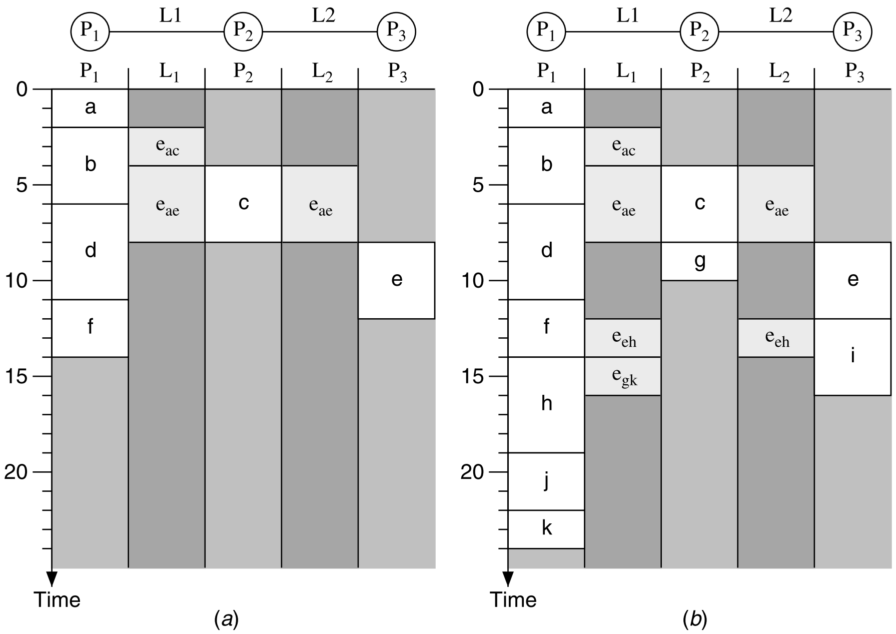


Figure 7.20. Example for contention aware list scheduling with start time minimization: intermediate (*a*) and final schedule (*b*). Node order of the sample task graph (e.g., in Figure 6.25) is *a*, *b*, *c*, *d*, *e*, *f*, *g*, *i*, *h*, *j*, *k*.

compensation e is scheduled on P_3 (vice versa in the classic model (Figure 5.1)). Also, f is scheduled on P_1 to avoid contention on L_1. Figure 7.20(a) shows the intermediate schedule at this point. Then, g and i are scheduled on the processors of their parent nodes. The following node h is decisive for the rest of the schedule: its earliest start time is on P_1, but its entering edge e_{eh} now blocks L_1 for the entering communication of j, which is quite large. Thus, j and k must be scheduled on P_1. The final schedule length (24) (Figure 7.20(b)) is not longer than in the classic model (Figure 5.1), but here more nodes are executed on P_1 and less interprocessor communications are performed. Hence, list scheduling achieves in this example the desired contention awareness in its scheduling decisions. In general, the schedule lengths under the contention model and the classic model are not directly comparable. Contention aware scheduling produces normally longer schedules, but shorter execution times on real machines. The experiments in Sinnen and Sousa [172, 176] verify this.

Complexity The complexity of contention aware list scheduling is similar to list scheduling under the classic model (Section 5.1). Essentially, only the calculation of the DRT is more complex than in the classic model. For example, the complexity of the second part of simple list scheduling (Algorithm 9) with start time minimization and the end technique is as follows. To calculate the start time of a node n, the data ready time $t_{\mathrm{dr}}(n)$ is computed, which involves the scheduling of n's entering edges. The scheduling of an edge includes the determination of the route and its scheduling on each link of the route. This is described by $O(\mathit{routing})$ (see Section 7.2.2) for each edge. If the route can be determined in $O(1)$ time (calculated or looked up), then $O(\mathit{routing})$ is just the complexity of the length of the route, as the edge must be scheduled on each link of the route. So for all nodes on all processors, calculating the DRT amortizes to $O(\mathbf{PE}O(\mathit{routing}))$. The start time of every node is computed on every processor, that is, $O(\mathbf{PV})$ times; hence, the total complexity is $O(\mathbf{P}(\mathbf{V} + \mathbf{E}O(\mathit{routing})))$. In comparison, the complexity under the classic model is $O(\mathbf{P}(\mathbf{V} + \mathbf{E}))$ and with the insertion technique $O(\mathbf{V}^2 + \mathbf{PE})$ (Section 6.1). Using the insertion technique in contention aware list scheduling has a complexity of $O(\mathbf{V}^2 + \mathbf{PE}^2O(\mathit{routing}))$, 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. As discussed in Section 7.2.2, for many systems $O(\mathit{routing})$ is at most linear in the number of processors $|\mathbf{P}|$ and links $|\mathbf{L}|$.

Sorting the entering edges of each node according to some heuristic amortizes to $O(\mathbf{E}\log\mathbf{E})$ for all nodes. Thus, the complexity of the second part of list scheduling is then $O(\mathbf{P}(\mathbf{V} + \mathbf{E}O(\mathit{routing})) + \mathbf{E}\log\mathbf{E})$ using the end technique and remains $O(\mathbf{V}^2 + \mathbf{PE}^2O(\mathit{routing}))$ using the insertion technique.

7.5.2 Priority Schemes—Task Graph Properties

So far, only the second part of list scheduling has been considered. But the question arises if the priority schemes used for the classic model are appropriate for the contention model. This question mainly refers to priority metrics based on the task graph properties (e.g., node levels) because dynamic metrics (e.g., the earliest start time of a node) are fully adapted to the contention model with the modified calculation of the DRT.

As analyzed in Section 7.4.1, the creation of the task graph remains unmodified for the contention model. Hence, all metrics like node levels or critical path can still be employed. However, in the contention model, the edge weights reflect the best case that no contention occurs. Thus, metrics involving edge weights (e.g., the bottom level) are inaccurate in two situations: (1) when a communication is performed locally, because it then has zero cost (as in the classic model); and (2) when a communication is delayed due to contention. As a result, metrics based on communication costs are less accurate. In particular, bounds on the schedule length, as established in Section 4.4, lose their validity. On the other hand, communication becomes even more important due to contention and must be considered in the ordering of the nodes.

Various node priority schemes for the first part of contention aware list scheduling are analyzed and compared in Sinnen and Sousa [177], among which the *bottom level* provides the best results. Experiments show that this combination—list scheduling with bottom-level node order—outperforms all other compared contention aware algorithms (Sinnen and Sousa [176, 178]).

7.5.3 Clustering

The major hurdle for the utilization of clustering (Section 5.3) under the contention model is the fact that clustering relies on an unlimited number of clusters (processors). As is well known by now, the classic model assumes a fully connected network (Property 6 of Definition 4.3), which is completely symmetric with an identical distance (of one) between any pair of processors. Consequently, the labeling of the clusters and their position within this fully connected network has no influence on the communication behavior and is therefore irrelevant for the clustering.

Under the contention model, however, the target system consists of a limited number of processors and is modeled by a topology graph. In general, the topology graph is not completely symmetric nor is the distance between all processor pairs identical. Hence, it does matter for the communication behavior where a processor is positioned within the network. How can such a topology graph be expanded in a meaningful way to an unlimited number of processors (i.e., at least $|\mathbf{V}|$ processors) for the clustering phase? The positioning of the clusters in this network should not influence the implicit schedule lengths of each clustering step.

For this reason, a solution to this problem can only be an approximation of the actual target system. During the clustering phase, a symmetric topology graph with an identical distance between every processor pair must be employed. Both the *fully connected network* and the *star network* (see Figure 7.7(*b*) and Section 7.4.2) fulfill this requirement. Employing the corresponding topology graphs with $|\mathbf{V}|$ processors makes clustering aware of contention, especially with the star network, which fully captures end-point contention. With such contention awareness, merging of clusters might be beneficial, when it would not be under the classic model.

Of course, in the third step of a clustering based heuristic (Algorithm 14)—after the mapping of the clusters to the real processors in the second step—the topology graph representing the real system should be employed. This third step is list scheduling based (Section 5.4), which has just been discussed in Section 7.5.1.

Clustering Approaches What follows is a brief discussion of how the three different approaches to clustering, as analyzed in Section 5.3, are employable under the contention model.

- *Linear clustering* (Section 5.3.2) does not require any modification whatsoever. There are no decisions taken in the linear clustering Algorithm 16 based on the schedule length. The algorithm only analyzes properties of the task graph; hence, the underlying scheduling model is irrelevant.
- *Single edge clustering* (Section 5.3.3) can be applied under the contention model with an approximated topology graph (see earlier discussion). The only part that requires some attention is the line "Schedule C_i using heuristic" in Algorithm 17, that is, the construction of the implicit schedule. Essentially, this is scheduling with a given processor allocation—a list scheduling that was already covered in Section 7.5.1.
- *List scheduling as clustering* (Section 5.3.4), as the name implies, is based on list scheduling, whose utilization under the contention model is analyzed in Section 7.5.1. It can be applied with an approximated topology graph as discussed earlier.

7.5.4 Experimental Results

It is a well-known fact that contention in the communication networks of parallel systems has a significant influence on the overall system performance. However, there are only a few experimental studies on the impact of contention in scheduling.

In Macey and Zomaya [132], schedules produced under the classic model are analyzed by simulating their execution in a system with link contention. The authors conclude that the assumption of concurrent communication in the classic model (Property 5 of Definition 4.3) is completely unrealistic; hence, it is imperative to consider contention in task scheduling.

Contention aware scheduling aims at producing schedules that are more efficient than those produced under the classic model. To attain this goal, the contention model attempts to reflect parallel systems more accurately than the classic model. But how accurate are the classic model and the contention model? And how efficient are the schedules produced under the two models?

Answers to these questions can only be found with an experimental evaluation on real parallel systems. Sinnen and Sousa [171, 176, 178] proposed a methodology with which they subsequently performed such an experimental evaluation. The methodology begins with the common procedure of scheduling algorithm comparisons: a large set of task graphs is scheduled by different heuristics—under the classic model and the contention model—on various target systems.

The employed target systems cover a wide spectrum of parallel architectures: (1) a cluster of 16 PCs (Telford [184]); (2) an 8-processor shared-memory SMP system, Sun Enterprise 3500 (Culler and Singh [48]); and (3) a 344-processor massively parallel system, Cray T3E-900 (Booth et al. [26]). These systems possess different network topologies and are modeled with corresponding topology graphs.

In the next step, code is generated that corresponds to the task graphs and their schedules, using the C language and MPI (message passing interface [138]), the standard for message passing on parallel systems.

This code is then executed on the real parallel systems and the accuracy, the speedup, and the efficiency of a schedule and its model are evaluated. Let $pt(\mathcal{S})$ be the execution time—the parallel time—of the code generated for schedule $\mathcal{S}$ and task graph $G = (\mathbf{V}, \mathbf{E}, w, c)$ 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) $acc(\mathcal{S}) = pt(\mathcal{S})/sl(\mathcal{S})$. The *speedup* of $\mathcal{S}$ is the ratio of sequential time to execution time $sup(\mathcal{S}) = seq(G)/pt(\mathcal{S})$. Finally, the *efficiency* of $\mathcal{S}$ is the ratio of speedup to processor number $eff(\mathcal{S}) = sup(\mathcal{S})/|\mathbf{P}| = seq(G)/(pt(\mathcal{S}) \times |\mathbf{P}|)$.

The experimental evaluation exposed that scheduling under the classic model results in very inaccurate and inefficient schedules for real parallel systems. Contention aware scheduling significantly improves both the accuracy and the efficiency of the produced schedules. Still, the results are improvable.

It is observed that results get worse—under both models—as the amount of communication in the task graph increases. In terms of the CCR (Definition 4.23), the classic model is extremely inaccurate for medium ($CCR = 1$) and high ($CCR = 10$) communications.

Note that the efficiency improvements from the employment of the contention model are superior to the typical difference between various scheduling algorithms under the classic model (Ahmad et al. [6], Khan et al. [103], Kwok and Ahmad [111]).

Nonetheless, contention aware scheduling, even though significantly more accurate and efficient than scheduling under the classic model, does not produce results that really satisfy. As analyzed by Sinnen and Sousa [171, 176, 178], this is in part due to inappropriate topology graphs employed in the experiments, which do not exploit the full potential of the topology graph of Definition 7.3. It is argued that especially the switch vertex and the hyperedge are important instruments of the topology graph which should be used for an accurate reflection of communication networks.

Another possible explanation for the still improvable results is that communication contention is not the only aspect that 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 4.3), the contention model (Definition 7.10) supposes a dedicated communication subsystem to exist in the target system (Property 4). 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. This is addressed in the next chapter.

7.6 CONCLUDING REMARKS

This chapter was dedicated to communication contention in task scheduling. Driven by the insight that the classic model is inappropriate for many parallel systems, a more realistic system model was studied. This model abandons the idealizing assumptions of completely concurrent communications and fully connected networks. Task scheduling under this model attains its awareness of contention by scheduling the edges onto the communication links. Owing to its similarity with the scheduling of the nodes, edge scheduling is intuitive and powerful.

The analyzed contention aware scheduling unifies the consideration of both types of contention—end-point and network contention—in one model. This is accomplished through an extended network model and its representation of the topology. At the same time, the details of the network are hidden from edge scheduling, which only sees the route for each communication.

It was shown that only a few modifications are necessary to adapt the techniques of classic scheduling for contention aware scheduling. In particular, the discussion covered how list scheduling, in all its forms, is employed under the new model.

In conclusion, there is almost no reason against favoring the contention model over the classic model in practice. The few changes that are necessary to make task scheduling algorithms contention aware usually result in only a small increase of the runtime complexity. The complexity difference between different scheduling heuristics for the classic model is often much larger. Yet, various experiments demonstrated that contention aware scheduling rewards with significantly increased accuracy and shorter execution times.

The only drawback is that the implementation of a contention aware algorithm is more involved. Its utilization also requires the modeling of the target system as a topology graph. In order to facilitate this task, one can start with a simple topology graph. Very appropriate in this context is the star topology, which corresponds to the one-port model and efficiently captures end-point contention. Later, the topology model can be enhanced, (i.e., made more accurate) in order to further improve the scheduling results.

Unfortunately, contention aware scheduling is not as accurate and efficient as one would like it to be. As discussed, this suggests that the scheduling model must be investigated further, especially with respect to the involvement of the processor in communication. This issue will be addressed in the next chapter.

7.7 EXERCISES

7.1 Draw the network of the following parallel systems as a topology graph (Definition 7.3):

(a) A cluster of PCs composed of three subnetworks. In each subnetwork 8 processors are connected to one central switch. The networks are linked through a direct connection from each switch to every other switch.

(b) A system of 9 processing nodes, which are connected through a 3 × 3 cyclic grid (Section 2.2.1). Each processing node consists of one interface to the grid and two processors, all connected via a single bus.

7.2 Describe how to model a butterfly network of 8 processors (depicted in Figure 2.11) as a topology graph. Consider:

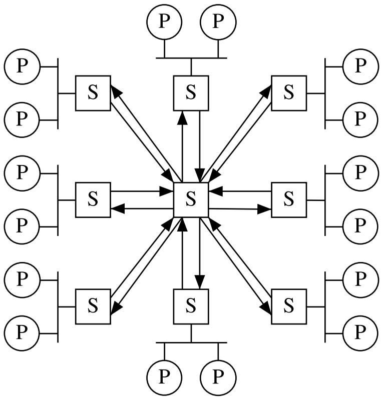
(a) The edge types to be employed, that is, directed, undirected, or hyperedge.

(b) Can the communication routes in such a topology graph be determined with a shortest-path algorithm or is a specific routing algorithm necessary? Give examples for routes between processors.

(c) Draw the topology graph.

7.3 Visit the online *Overview of Recent Supercomputers* by van der Steen and Dongarra [193] on the site of the TOP500 Supercomputer Sites [186] and choose one modern system. Model this system with a topology graph (Definition 7.3).

7.4 Figure 7.9 displays the topology graph for 8 dual processors connected via a switch:

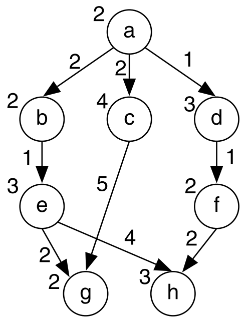


(a) Find examples for four processor pairs that can communicate concurrently, and indicate which links are occupied by the communication.

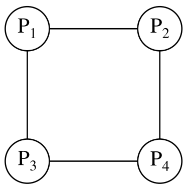
(b) Find examples for processor pairs that cannot communicate concurrently and indicate the links where the communication conflicts.

(c) Repeat this exercise for the topology graphs constructed in the previous exercises.

7.5 Schedule the following task graph



under the contention model on four processors connected as a ring, employing half duplex links:

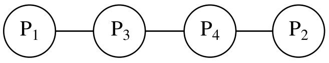


The nodes are allocated to the processors as specified in the following table:

Node	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*
Processor	1	3	2	4	2	1	3	4

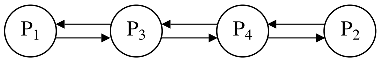
Schedule the nodes in alphabetic order as early as possible.

7.6 Repeat the scheduling done in Exercise 7.5, now on a heterogeneous linear network

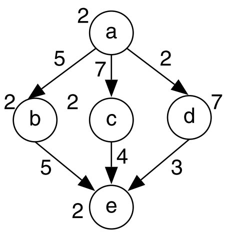


where link L_{34} is twice as fast as the other two links; that is, $b(L_{34}) = 2b(L_{13}) = 2b(L_{42})$.

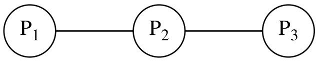
How does the resulting schedule change for full duplex links (i.e., each undirected edge of the linear network is substituted by two counterdirected edges)?



7.7 Schedule the following fork-join task graph



under the contention model on a three-processor linear network



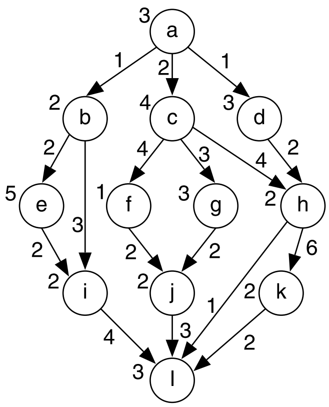
Nodes a and b shall be scheduled on P_1, c on P_2, and d and e on P_3. Their start times should be as early as possible.

(a) Does it make a difference in which order nodes c and d are scheduled? Compare to scheduling under the classic model.

(b) Node e, as the exit node, must be the last node to be scheduled. Does the order in which its incoming communications are scheduled make a difference for the schedule length?

(c) Describe in general terms the situation when the scheduling order of incoming communications makes a difference. Suggest an ordering strategy.

7.8 Perform list scheduling with start time minimization under the contention model with the following task graph on a four-processor ring (see Exercise 7.5). Order the nodes in decreasing bottom-level order.



7.9 Section 7.5 studies how scheduling algorithms can be employed under the contention model. For the cases discussed this is relatively simple. One exception seems to be the utilization of node duplication (Section 6.2) under the contention model.

Describe the challenges that node duplication imposes under the contention model. Try to find examples for these challenges.