Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Parallel Computing 30 (2004) 81–101

PARALLEL COMPUTING

www.elsevier.com/locate/parco

List scheduling: extension for contention awareness and evaluation of node priorities for heterogeneous cluster architectures ☆

Oliver Sinnen, Leonel Sousa *

Universidade Técnica de Lisboa, IST/INESC-ID, Rua Alves Redol 9, 1000-029 Lisboa, Portugal

Received 22 September 2001; received in revised form 20 August 2003; accepted 10 September 2003

Abstract

In the area of static scheduling, list scheduling is one of the most common heuristics for the temporal and spatial assignment of a directed acyclic graph (DAG) to a target system. As most scheduling heuristics, list scheduling assumes fully connected homogeneous processors and ignores contention on the communication links.

This article extends the list scheduling heuristic for contention aware scheduling on heterogeneous arbitrary architectures. The extension is based on the idea of scheduling edges to links, likewise the scheduling of nodes to processors. Based on this extension, we compare eight priority schemes for the node order determination in the first phase of list scheduling. Random graphs are generated and scheduled with the different schemes to homogenous and heterogeneous parallel systems from the area of cluster computing. Apart from identifying the best priority scheme, the results give new insights in contention aware DAG scheduling. Moreover, we demonstrate the appropriateness of our extended list scheduling for homogeneous and heterogeneous cluster architectures.

Keywords: List scheduling; Directed acyclic graph (DAG); Communication contention; Heterogeneous cluster architectures

☆ Part of this work was supported by FCT (Fundação para a Ciência e a Tecnologia) and FSE (Fundo Social Europeu) in the context of the III European Framework of Support, which we gratefully acknowledge.

* Corresponding author.
E-mail addresses: oliver.sinnen@inesc-id.pt (O. Sinnen), las@inesc-id.pt (L. Sousa).

0167-8191/$ - see front matter
doi:10.1016/j.parco.2003.09.002

1. Introduction

Programming a parallel system requires the scheduling of sub-tasks to the resources of the parallel system, while respecting the precedence constraints among the sub-tasks. For static scheduling, that is scheduling at compile time, the program to be parallelised is usually modelled as a directed graph. Nodes represent the tasks, i.e. the computation, and edges the inter task communication. The directed acyclic graph (DAG) is the most common graph model used for static scheduling [26].

The scheduling of a DAG (or task graph) is, in its general form, an *NP*-hard problem [5], i.e. an optimal solution cannot be calculated in polynomial time (unless $NP = P$). Scheduling algorithms are therefore based on heuristics that try to produce near optimal solutions. Early scheduling algorithms do not take communication into account [1,3,10,11,13], but due to the increasing gap between computation and communication performance of parallel systems, the consideration of the communication became soon more important and is included in many scheduling algorithms [9,12,14,16,18,20,21,23–25,28,31,32]. Most of these algorithms assume the target system to be a homogenous system with fully connected processors and a dedicated communication sub-system. Moreover, contention for communication resources is neglected. List scheduling [7], in one form or the other, is the most employed scheduling heuristic by these algorithms, given a bounded number of processors. Very few algorithms model the target system as an arbitrary processor network and incorporate contention in the scheduling heuristic [8,17,27,30]. However, Macey and Zomaya [22] showed that the consideration of link contention is significant to produce accurate and efficient schedules.

In this article we extend the classic list scheduling for contention aware scheduling on heterogeneous arbitrary systems. The extended list scheduling is appropriate for heterogeneous target systems, both in terms of processors as well as in terms of communication links. Based on this extended list scheduling we compare eight priority schemes for the node order determination in the first phase of list scheduling. The evaluation of the schemes is based on the results they produce for random graphs scheduled to several parallel systems of the area of cluster computing. Our intention is not only to find the best scheme, but also to show the appropriateness of our extended list scheduling for cluster computing. Further, by analysing the behaviour of the different schemes, we aim to obtain a deeper insight into DAG scheduling.

The rest of the article is organised as follows. Section 2 introduces the necessary models and gives the background and Section 3 proposes the extended list scheduling. In Section 4, the various priority schemes for the node order determination are discussed and in Section 5 the experimental results are presented and analysed. The article concludes with Section 6.

2. Definitions and background

The program to be scheduled to a target system is represented as a directed acyclic graph (DAG). The DAG is a directed acyclic graph $G = (V, E)$, where V is a finite set

of $|V|$ nodes (vertices) and E is a finite set of $|E|$ directed edges. The nodes of G represent the computation and the edges the communication of the modelled program. The weight $w(n_i)$ assigned to node n_i represents its computation cost and the weight $w(e_{ij})$ assigned to edge e_{ij} represents its communication cost. The indices of an edge denote its direction, that is edge e_{ij} is directed from node n_i to n_j. Fig. 1 shows an example DAG with the assigned node and edge weights.

The topology of the target system is modelled as an undirected graph $G_T = (P, L)$, where P is a finite set of $|P|$ vertices and L is a finite set of $|L|$ undirected edges. A vertex P_i represents the processor i and an undirected edge L_{ij} represents a bi-directional communication link between the incident processors P_i and P_j. This definition of a topology graph, which is employed in [8,17,27,30], is able to represent most static communication networks. Yet, it fails to accurately reflect the important bus topology. Inherent in the undirected graph, an edge is only incident on two vertices, whereas a bus is connected to multiple processors. We therefore propose the utilisation of a hypergraph [2] with one hyperedge for the representation of a bus topology. Let P be a finite set of $|P|$ vertices and L be a hyperedge incident on all vertices of the vertex set P. A vertex P_i represents the processor i and the hyperedge L represents the bus connecting all processors.

A weight $w(P_i)$ assigned to a processor P_i represents its relative execution speed and a weight $w(L_{ij})$ assigned to a link L_{ij} represents its relative communication speed. Fig. 2 shows examples of undirected graphs and a hypergraph representing three different target system topologies. The processing speed of a system resource is defined relatively to the average speed of all resources. In order to calculate the average speed, the speed value of 1 is attributed to one randomly chosen resource and the speed of the other resources is defined relative to this resource. The average speed is then calculated and the speeds of the resources are normalised based on the average speed. The costs of the elements of the DAG are in turn calculated based on the average speed of the target system's resources. For example a processor P_i with its

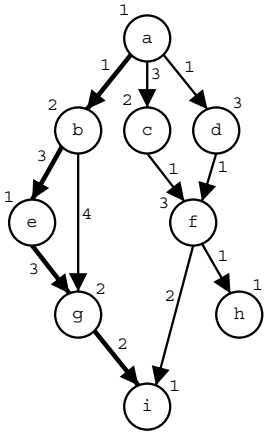

Fig. 1. A DAG.

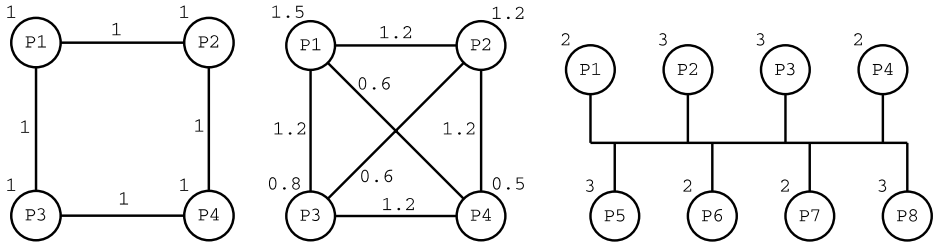

Fig. 2. Topology graphs representing target systems: (a), (b) undirected graphs with four processors; (c) hypergraph with eight processors. (a) Homogeneous ring, (b) heterogeneous homogeneous fully connected and (c) heterogeneous bus.

relative speed $w(P_i)$ executes the node n_j in $w(n_j)/w(P_i)$ time. If $w(P_i) = 1$, the processor P_i runs with the average speed and executes the node n_j in $w(n_j)$ time.

2.1. DAG scheduling

Without considering communication contention and assuming a homogeneous fully connected architecture, scheduling a DAG to a target system is to generate an execution schedule of the graph's nodes on the system's processors. Every node is assigned to a processor and attributed a start and a finish time. We say, node n_i is scheduled to processor P_k, and denote its start time on this processor as $\mathrm{ST}(n_i, P_k)$ and its finish time as $\mathrm{FT}(n_i, P_k)$. The finish time of processor P_k is defined as the finish time of the last node executed on this processor, $\mathrm{FT}(P_k) = \max_i\{\mathrm{FT}(n_i, P_k)\}$. After all nodes of the DAG have been scheduled to the target system, the schedule length is defined as $\max_k\{\mathrm{FT}(P_k)\}$, that is the finish time of the last node. The aim of all scheduling algorithms is to minimise the schedule length without violating the precedence-constraints among the tasks (nodes).

The start time of a node n_i is constrained by its dependences on other nodes, which are the incoming communications represented by the edges of the graph. A node can start execution when all data has arrived. The data ready time $\mathrm{DRT}(n_i, P_k)$ of node n_i on processor P_k is defined as the time when the last communication from its parent nodes arrives:

$$\mathrm{DRT}(n_i, P_k) = \max_j\{\mathrm{FT}(n_j) + w(e_{ji})\} \tag{1}$$

with $n_j \in V$ and $e_{ji} \in E$, $j = 1, \ldots, |V|$. The communication costs between two nodes scheduled to the same processor is assumed to be negligible and is set to zero. Note, the data ready time for node n_i can only be determined if all parent nodes have already been scheduled. $\mathrm{FT}(n_j)$ therefore denotes the finish time of the node n_j in the partial schedule. Omitting the processor in FT is non-ambiguous since every node is only contained once in the partial schedule. For a valid schedule, $\mathrm{ST}(n_i, P_k) \geqslant \mathrm{DRT}(n_i, P_k)$ must be true for all nodes.

DAG scheduling in general expects a dedicated target system. In other words, the scheduled program does not share the resources of the target system with any other program during its execution.

2.2. List scheduling

The most common heuristic for DAG scheduling is the traditional list scheduling [7]. First utilised with the assumption of zero task communication costs, e.g. [1,3,10,11,13], it was later used for non-zero task communication costs, e.g. [12,16,20,21,32]. In the following we consider non-zero task communication costs.

Like most scheduling algorithms, list scheduling assumes a fully connected homogeneous target system with a dedicated communication sub-system. The basic structure of list scheduling is rather simple. In the first phase a priority is attributed to every node. Based on this priority, the nodes are ordered into a list. In each step, the node with the highest priority among the ready nodes is chosen and added to the list. A node is said to be ready, if all its predecessors in the graph are already in the list. After adding the chosen node to the list, the new set of ready nodes is determined and the step is repeated until all nodes are contained in the list. Note that with this definition of list scheduling, the nodes' priorities are determined a priori, they do not depend on the status of the list or a partial schedule. An alternative approach, though with higher complexity, is to use dynamic priorities, which are updated during the second phase of scheduling. Here, we only consider static priorities and the interested reader should referred to e.g. [1,27].

In the second phase of list scheduling, the algorithm iterates over the list built in the first phase and schedules the nodes to the processors of the target system. To accomplish that, it determines the start time of each node in turn on every processor and chooses the processor that allows the earliest *start* time. The start time of a node is constrained by (i) the arrival time of the last incoming communication (DRT) and (ii) by the current finish time of the processor. The earliest start time of node n_i on processor P_k is

$$\mathrm{ST}_{\min}(n_i, P_k) = \max\{\mathrm{FT}(P_k), \mathrm{DRT}(n_i, P_k)\}, \tag{2}$$

and the earliest start time of node n_i on the target system is

$$\mathrm{ST}_{\min}(n_i) = \min_l\{\mathrm{ST}_{\min}(n_i, P_l)\}. \tag{3}$$

An alternative approach to determine the node's start time also considers idle time slots between already scheduled nodes in order to *insert* the node into a slot if appropriate [15]. This technique has higher complexity and we discuss it in [29]. Fig. 3 shows a possible partial schedule of the DAG presented in Fig. 1 and the utilised node list. Nodes a, b, e, d, c, g have already been scheduled to the target system comprising of three processors and the current node is node f. Processor P_3 allows its earliest starting time ($\mathrm{ST}(f, P_3) = 6$), since the incoming communication from node c is zeroed. On processor P_1 and P_2 f's earliest start time is 7, caused by the communication from node c on processor P_3.

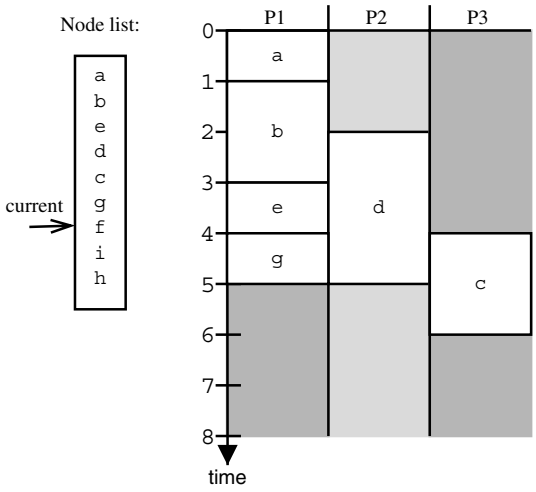

Fig. 3. List Scheduling for the DAG in Fig. 1.

The list scheduling heuristic can be summarised as follows:

(1) Determine the nodes' priorities. Order the nodes into a list according to their priorities, respecting their precedence constraints.
(2) Iterate over the node list from (1) and schedule each node to the processor that allows its earliest start time (Eq. (3)).

The above described list scheduling heuristic generates a valid schedule, which means that the precedence constraints among the nodes are respected. The first phase creates a list of nodes respecting the nodes' dependences and the second phase schedules the nodes in exactly this order, ensuring that all predecessors of a node to be scheduled have already been scheduled. Finally, the scheduling of a node according to Eq. (2) ensures the timely arrival of its incoming communications. In the second phase the finish time of every node is evaluated P times together with its incoming edges, once for every processor, resulting in a complexity of $O(P(V+E))$. The complexity of the first phase depends on which priority scheme is used (see Section 4).

3. List scheduling with communication contention

In this section we propose an extension to the concept of traditional list scheduling which supports arbitrary target systems and the consideration of communication contention. The target system is represented by the model described in Section 2, which allows arbitrary topologies with heterogeneous processors and communication links.

3.1. Edge scheduling and link heterogeneity

The basic underlying idea is to treat the communication edges in the same way as the nodes of the DAG: the edges are scheduled to the communication links in the same way the nodes are scheduled to the processors [27,29,30].

Corresponding to the node, we define $\mathrm{ST}(e_{ij}, L_{lk})$ to be e_{ij}'s start time on link L_{lk} and $\mathrm{FT}(e_{ij}, L_{lk})$ its finish time. Also, $\mathrm{FT}(L_{lk})$ is defined as the finish time of a communication link. While the start time of a node is constrained by the data ready time of its incoming communication, the start time of an edge is restricted by the finish time of its origin node. The scheduling of an edge differs further from that of a node, in that an edge might be scheduled on more than one link. A communication between two nodes, which are scheduled on two different but not adjacent processors, utilises all links of the communication route between the two processors. The edge representing this communication must be scheduled on each of the involved links. The method of data transmission on the links of a communication route can be somewhere in between two extremes: on the one end, data is transmitted on all links at the same time, i.e. when the transmission finishes on the first link it also finishes of the last link; on the other end, the data is transmitted on one link at a time and when finished it is forwarded to the next link. For today's parallel systems it is a realistic assumption that communication happens on all links of a route at the same time, since in general there is no significant delay between two adjacent links. As it will be seen below, introducing a delay for each "hop" is, however, easily accomplished.

For causality reasons, an edge's start time on a link cannot be earlier than the edge's start time on the first link. Further, the edge's finish time cannot be earlier than on the previous link. This matter becomes relevant for heterogeneous links as a fast link following a slow link cannot "overtake" the latter. We formalise this in the following.

Let $p = \langle L_1, L_2, \ldots, L_n \rangle$ be a path with n edges of a topology graph G_{T} that represents a communication route and the communication traverses the links in the order of the path. The earliest start time of the edge e_{ij} scheduled to the first link L_1 of this route and its finish time are

$$\begin{aligned} \mathrm{ST}_{\min}(e_{ij}, L_1) &= \max\{\mathrm{FT}(L_1), \mathrm{FT}(n_i)\}, \\ \mathrm{FT}(e_{ij}, L_1) &= \mathrm{ST}(e_{ij}, L_1) + w(e_{ij})/w(L_1). \end{aligned} \tag{4}$$

The earliest start time of the edge e_{ij} on all subsequent links and its finish time is defined by the following equations:

$$\begin{aligned} \mathrm{ST}_{\min}(e_{ij}, L_k) &= \max\{\mathrm{FT}(L_k), \mathrm{ST}(e_{ij}, L_1), \mathrm{FT}(e_{ij}, L_{k-1}) - w(e_{ij})/w(L_k)\}, \\ \mathrm{FT}(e_{ij}, L_k) &= \mathrm{ST}(e_{ij}, L_k) + w(e_{ij})/w(L_k) \end{aligned} \tag{5}$$

with $k = 2, \ldots, n$. Thus, on a subsequent link the edge e_{ij} can only start after the link is ready ($\mathrm{FT}(L_k)$) and late enough so that the edge does not finish earlier than on the previous link ($\mathrm{FT}(e_{ij}, L_{k-1}) - w(e_{ij})/w(L_k)$). For causality, edge e_{ij} must also start later than on the first link ($\mathrm{ST}(e_{ij}, L_1)$), otherwise parts of the communication could arrive earlier at the destination processors than they were sent.

Note, the communication costs are adapted to the link speed ($w(e_{ij})/w(L_k)$) for the calculation of the finish time. This simple approach allows an accurate modelling of the target system's link heterogeneity. Moreover, the inclusion of delays for communication via multiple links, as mentioned above, can be easily performed by introducing a delay in the calculation of the start times on subsequent links. For the utilisation of idle time slots between already scheduled edges, the determination of the start and finish times must be modified. This *insertion* scheduling has higher complexity [29] and is, likewise for node scheduling, not considered in this article.

The determination of the route, i.e. the path in the topology graph G_T, is done according to the routing policy of the target system. For a given pair of processors, the routing algorithm used for edge scheduling must return the sequence of links of the topology graph G_T that correspond to the links of the target system, which are used for communication between the two specified processors. Most of todays generic parallel systems employ a shortest path routing algorithm [6], which can be calculated in the topology graph G_T based on a breadth first search (BFS) [4].

To illustrate the scheduling of edges to links, Fig. 4 shows an example where edge e_{ij} is scheduled to the route $\langle L_1, L_2, L_3, L_4\rangle$ with $\mathrm{FT}(n_i) = 2$ for node n_i. Link L_2 has half of the other links' speed, $w(L_2) = 1$, and thus the communication of e_{ij} needs twice the time. Since communication on the subsequent link L_3 cannot finish earlier than on link L_2 (Eq. (5)), it starts later than on L_1, even though link L_3 is idle.

The scheduling of edges to links has practical consequences in the second phase of the list scheduling. The data ready time $\mathrm{DRT}(n_i, P_k)$ of node n_i on processor P_k is now calculated based on the finish time of the incoming edges on their assigned links, thus Eq. (1) becomes:

$$\mathrm{DRT}(n_i, P_k) = \max_j \{\mathrm{FT}(e_{ji}, L_{lk})\} \tag{6}$$

with $n_j \in V$ and $e_{ji} \in E$, $j = 1, \ldots, |V|$. If the origin node n_j of an incoming communication is scheduled on the same processor as node n_i, i.e. $l = k$, the communication time is assumed to have cost zero and we define $\mathrm{FT}(e_{ji}, L_{kk}) = \mathrm{FT}(n_j)$. For the

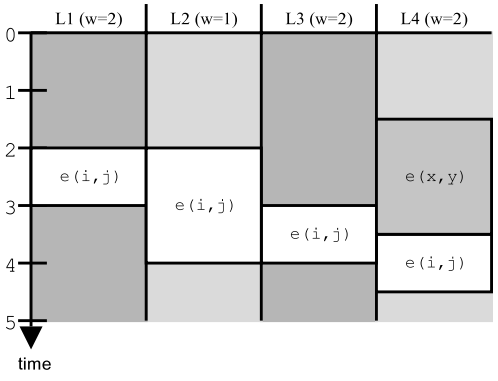

Fig. 4. Edge scheduled to links.

determination of the start time of a node on a processor, all of the node's incoming edges must be temporarily scheduled to the respective links. Only this guarantees a correct view of the communication times and the contention on the links.

3.2. Processor heterogeneity

The heterogeneity of the processors brings a change to the calculation of the finish time of the nodes, while the start time is calculated unchanged with Eq. (2). Likewise the determination of the edges finish time, the node finish time is now

$$\mathrm{FT}(n_i, P_k) = \mathrm{ST}(n_i, P_k) + w(n_i)/w(P_k), \tag{7}$$

that is the processor's speed is taken into account when calculating the execution time.

While for homogeneous systems the processor that permits the earliest start time of a node is automatically the processor that permits also its earliest finish time, the situation for heterogeneous systems is different. Due to varying processor speed, a processor that allows the earliest start time is not necessarily the processor that allows the earliest finish time for a node. To recognise this fact, our extended list scheduling approach schedules a node, in the second phase of the heuristic, to the processor that permits its earliest *finish* time.

3.3. Proposed extended list scheduling

The proposed extended list scheduling discussed in the previous sections can be summarised as follows:

(1) Determine the nodes' priorities. Order the nodes into a list according to their priorities, respecting their precedence constraints.
(2) Iterate over the node list from (1):
   (a) For each node, find the processor that allows its earliest *finish* time by temporarily scheduling the incoming edges to the links; the communication routes are determined according to the target system.
   (b) Schedule the node to the chosen processor and the incoming edges to the corresponding links.

In the second phase of the extended list scheduling, the finish time of every node is evaluated P times together with its incoming edges, once for every processor. This has a complexity of $\mathrm{O}(P(V+E))$, which is the complexity of the traditional list scheduling. For every temporarily scheduling of an edge, the links of the communication route on which the edge is scheduled must be determined. We designate the complexity of this $\mathrm{O}(\mathrm{routing})$. Thus, the complexity of the second phase of the extended list scheduling is $\mathrm{O}(P(V + E \cdot \mathrm{O}(\mathrm{routing})))$. As defined in edge scheduling (Section 3.1), the employed routing algorithm is target system dependent. Most systems employ a shortest path routing algorithms, which can be reflected in the topology graph G_T by a BFS based algorithm. For such systems, routing has a complexity

of $O(\text{routing}) = O(P + L)$, being the complexity of a BFS. Obviously, in regular topologies the complexity is lower, e.g. fully connected, bus: $O(\text{routing}) = O(1)$; ring: $O(\text{routing}) = O(P)$.

4. Priority attributions

After discussing the second phase of the extended list scheduling in the previous section, we now come back to the attribution of node priorities in the first phase, which is a significant issue of list scheduling.

The fundamental idea for the choice of a node priority is the node's relative importance. In list scheduling it is assumed, and shown by experiment [9,14,18], that the overall execution time of a program can be reduced if an important node is scheduled as soon as possible, i.e. before nodes with less importance. The earlier a node is considered for scheduling the earlier it can aquire a processor for its execution. The challenge is to find priorities that well reflect the nodes importance in the context of scheduling. In this article we compare various priority schemes for the scheduling of DAGs to systems with different topologies from the area of cluster computing.

A measure widely used for the attribution of node priorities are "levels" [7], which are path lengths of the graph. The bottom-level $\text{bl}(n_i)$ of node n_i denotes the length of the longest path starting with n_i. Complementary to the bottom-level, the top level $\text{tl}(n_i)$ of node n_i, is defined as the length of the longest path entering n_i. In the DAG of Fig. 1, node e has $\text{bl}(e) = 9$ and $\text{tl}(e) = 7$. The length of a path is defined as the sum of all its node and edge weights (weights are determined as described in Section 2). A level calculated only based on the node weights is called *computation* level—for example the computation bottom-level $\text{bl}_{\text{comp}}(n_i)$ (e.g. $\text{bl}_{\text{comp}}(e) = 4$). By induction, it can be easily shown that the longest path starting with a node always ends in a *sink node*, i.e. a node without leaving edges and that the longest path entering a node always starts in a *source node*, i.e. a node without entering edges.

A lower bound for the execution time of a DAG is its computation critical path, i.e. the longest computation path through the graph. The DAG of Fig. 1 has the computation critical path $\langle a, d, f, i\rangle$ of length 8. As all nodes of this path have to be executed in serial order, due to their precedence constraints, the length of the computation critical path is its minimum execution time achievable on a parallel machine. Note that this is only a lower bound for the execution time of the DAG and is in general not the optimal execution time. To consider communication, the critical path (CP) can be determined based on the computation and communication costs. In Fig. 1 the CP $\langle a, b, e, g, i\rangle$, length 16, is the emphasised path. When doing this, the length of the CP (cp) is

$$\text{cp} = \max_i\{\text{bl}(n_i) + \text{tl}(n_i)\},$$

being the maximum sum of a node's bottom and top-level. For nodes on the CP, the critical nodes, the sum of bottom-level and top-level equals cp. In particular for the

first node of the critical path $\mathrm{bl}(n_{c1}) = \mathrm{cp}$ ($\mathrm{bl}(a) = 16$), since the first node is a source node with $\mathrm{tl}(n_{c1}) = 0$. Thus, the node with the highest bottom level is the first node of the CP. After removing this node from the graph, the node with the highest bottom-level of the remaining graph is the first node of the CP of this remaining graph. So, by choosing always the node with the highest bottom-level, we choose the first node of a dominant sequence, that is the dynamic critical path of the remaining graph.

The top-level and the bottom-level of every node of the DAG can be determined in $O(V + E)$ using a depth first search (DFS) [4]. During the scheduling, the bottom-level of a node does not change in contrast to the top-level that might be affected by the zeroing of communication costs between nodes scheduled to the same processor.

For the experiments we employ node lists in decreasing bottom-level (BL) order and, for the purpose of comparison, in decreasing computation bottom-level (BLcomp) order. We expect that the bottom-level performs better since it includes communication costs. As shown in [30], the decreasing (computing) bottom-level order is also a topological order [4] of the nodes, that is an order that respects the nodes' dependences. Thus, the complexity to determine the node list is $O(V + E)$ to calculate the levels and $O(V \lg V)$ for the sorting (e.g. mergesort [4]), resulting in the total of $O(V \lg V + E)$.

When using the bottom-level as a measure for the priority, the incoming communications of the node considered are not taken into account. All ancestor nodes have already been scheduled to the target system, but not their outgoing edges. Two nodes, while having similar bottom-levels, can significantly differ in their incoming communication costs. The one with the more important communication should be scheduled first to have a larger resource choice. The bottom-level can be "extended in top direction" by adding the incoming communication costs. Two schemes are intuitive: (i) The priority assigned to node n_i is the sum of its bottom-level and the largest incoming communication, $\mathrm{pri}(n_i) = \mathrm{bl}(n_i) + \max_j\{w(e_{ji})\}$ (e.g. in Fig. 1, $\mathrm{pri}(g) = 5 + w(e_{bg}) = 9$). (ii) The priority assigned to node n_i is the sum of its bottom-level and the communication costs from its critical ancestor n_k, i.e. the parent node with the highest sum of top-level, computation cost and communication costs to n_i, $\mathrm{pri}(n_i) = \mathrm{bl}(n_i) + w(e_{ki})$. For example, the critical ancestor of g in Fig. 1 is e ($\mathrm{tl}(e) + w(e) + w(e_{eg}) = 7 + 1 + 3$ compared to $\mathrm{tl}(b) + w(b) + w(e_{bg}) = 2 + 2 + 4$), thus $\mathrm{pri}(g) = 5 + w(e_{eg}) = 8$.

The complexity to generate node lists using these priorities is as follows. Finding the largest or the critical communication of each node has a complexity of $O(V + E)$ using a DFS, likewise the calculation of the levels. Since nodes in decreasing priority order are in general not in topological order, we must maintain a set of ready nodes when creating the list. The complexity of adding and removing one node from this set is $O(\lg V)$, using a heap [4], and is done once for every node. Determining the set of ready nodes is $O(E)$ for all nodes. The total complexity of the node list determination is thus $O(V \lg V + E)$. For the experiments we employ both schemes (BL + MaxComm, BL + CriticalComm) and, as used in [27], the *computation* bottom-level plus the largest incoming communication (BLcomp + MaxComm).

Another technique to establish an order among the nodes is based on a critical path oriented traversal of the DAG. While the bottom-level, as described above,

orders the nodes of the DAG according to the current dominant sequence, this technique orders the nodes based on the global critical path [17]. The algorithm distinguishes between critical path nodes (CPN), in-branch nodes (IBN), which are nodes that have a path to a CPN, and out-branch nodes (OBN), which are all nodes that are neither CPNs nor IBNs. For example, the DAG of Fig. 1 has: the CPNs a, b, e, g, i; the IBNs c, d, f and only one OBN h. Defining CPN, IBN and then OBN as the order of importance, the aim is to schedule the CPN as soon as possible. This approach results in scheduling first the critical path nodes in order. If a CPN has ancestors not yet in the list (i.e. IBNs), these are recursively added to the list before this CPN. When all CPNs are scheduled, and therefore also all IBNs, the OBNs are added to the list.

To establish an order between the IBNs, e.g. between the IBNs c and d in the example DAG, the one with the largest communication is chosen first in [17] and the OBNs are added to the list in topological order. In a later article by the same authors [19], the parent with the highest bottom-level is chosen first, ties are broken by choosing the one with the smallest top-level. The OBNs are added in decreasing order of their bottom-levels. For the experiment we employ both approaches (CP_MaxComm, CP_BL_TL). Furthermore, we employ a third approach where the IBN with the highest top-level is chosen (CP_TL). This is the last node of the critical path of the sub-graph formed by a node's ancestor. In this approach, the OBNs are also added in decreasing order of their bottom-levels.

Finding the CP and calculating the levels has a complexity of $O(V + E)$. The traversal of the CPNs and the IBNs is $O(E)$, but determining the IBN with the highest priority is limited by $O(E \lg E)$ over all IBNs. Adding the OBNs in topological order is $O(V + E)$, in decreasing top-level order it is $O(V \lg V + E)$. Thus, CP_MaxComm is $O(V + E \lg E)$ and CP_BL_TL, as well as CP_TL, is $O(V \lg V + E \lg E)$.

As a reference we generate a node list according to the topological order of the nodes (TopoOrder). The topological order guarantees the compliance of the precedence constraints, but in terms of node's importance it establishes a random order. Table 1 lists the above discussed priority schemes with a short description.

Table 1
Summary of priority schemes

Priority scheme	Description
	Nodes ordered by
BL	Bottom-level
BLcomp	Computation bottom-level
BL + CriticalComm	Bottom-level + critical incoming communication
BL + MaxComm	Bottom-level + maximum incoming communication
BLcomp + MaxComm	Computation bottom-level + maximum incoming communication
	Critical path traversal: IBNs chosen by
CP_BL_TL	Bottom-level, ties broken by top-level; OBNs ordered by bottom-level
CP_TL	Top-level; OBNs ordered by bottom-level
CP_MaxComm	Maximum communication; OBNs ordered in topological order
TopoOrder	Topological order, i.e. random order respecting dependences

5. Experiments

In this section we present an experimental comparison of the priority schemes for our extended list scheduling, as described in the previous sections. For this purpose we have implemented the algorithms and generated random graphs, which were then scheduled by these algorithms to various target systems. The comparison is intended not only to present quantitative results but also to qualitatively analyse the results and to suggest explanations, for a better insight in the overall scheduling problem.

5.1. Experimental setup

For the generation of random graphs, which are commonly used to compare scheduling algorithms [9,14,18], two fundamental characteristics of the DAG are considered: (i) the communication to computation ratio (CCR) and (ii) the average number of edges per node. The CCR is defined as the sum of all communication costs (that is the weight of all edges) divided by the sum of all computation costs: CCRs of 0.1, 1 and 10 are used to simulate low, medium and high communication, respectively. For the average number of edges per node, we utilised values of 2 and 5. Graphs were generated for all combinations of the two above parameters with the number of nodes ranging between 100 and 1500, in steps of 200. Every possible edge (DAGs are acyclic) was created with the same probability, calculated based on the average number of edges per node. To obtain the desired CCR for a graph, node weights are taken randomly from a uniform distribution $[0.1, 1.9]$ around 1, thus the average node weight is 1. Edge weights are also taken from a uniform distribution, whose mean depends on the CCR and on the average number of edge weights (e.g. the mean is 5 for CCR = 10 and 2 edges per node). The relative deviation of the edge weights is identical to that of the node weights. Every set of the above parameters was used to generate several random graphs in order to avoid scattering effects. The results presented below are the average of the results obtained for these graphs.

The architectures employed for the experiments are aimed to reflect a wide range of typical cluster computing systems. Hence, we used the following architectures: fully connected processors, bus topology and ring topology. The bus topology does not only represent a SMP system, but also topologies of cluster computing, e.g. workstations connected via a hub (not a switch). The three different architectures were modelled with 8, 32 and 128 processors. In order to get results for heterogeneous systems, the fully connected architecture was also used with heterogeneous processors and links. For the simulation of a NUMA architecture, the processors of a fully connected system are divided into pairs and the speed of the link connecting the two processors is set to 10, i.e. 10 times faster than the unmodified links. This scenario corresponds to a cluster of dual processor machines connected via a switch, with a total of 8 and 32 processors. To represent a network of workstations (NOW), i.e. a cluster of in general heterogeneous systems, we modified the processors' speed

of 8 (all speeds are different taken from the interval $[0.6, 1.4]$) and 32 ($w(P_i) = 0.8$ with $1 \leqslant i \leqslant 16$, $w(P_j) = 1.2$ with $17 \leqslant j \leqslant 32$) processor fully connected machine, but in a way that the average speed remained 1, in order to obtain comparable results.

5.2. Results

5.2.1. Level based schemes

Due to the high number of different priority schemes, only the schemes based on the bottom-level and the computation bottom-level are initially compared. Fig. 5 shows the schedule length of the graphs, for CCR of 0.1 and 2 average edges per node, on 32 and 128 fully connected processors. The results on 128 processors are less stable than those on 32 processors due to the closer number of DAG nodes regarding the number of processors. At first, it can be surprising that there is no perceivable difference between the five schemes. Since communication is low it is not significant in the calculation of the priorities, thus all schemes behave like nodes sorted by the computation bottom level. However, the importance of the node list order becomes apparent by the fact that the topological order performs much worse. This was confirmed by all other experiments (in some experiments several times worse), therefore TopoOrder is omitted in the figures of the remaining paper.

With high communication, CCR = 10 in Fig. 6a, the differences between the schemes becomes significant. Worst performs BLcomp + MaxComm, which can be explained with the disproportional significance of the largest incoming communication in the priority. Of the remaining schemes, BL almost always yields better results than BLcomp, BL + MaxComm and BL + CriticalComm, with a maximum difference of about 20%. The results for the other experiments with the (computation) bottom-level schemes range qualitatively between the examples of Figs. 5 and 6a.

Surprising is here that the computation bottom-level is not much worse than the schemes that take communication costs into account. The node order established by

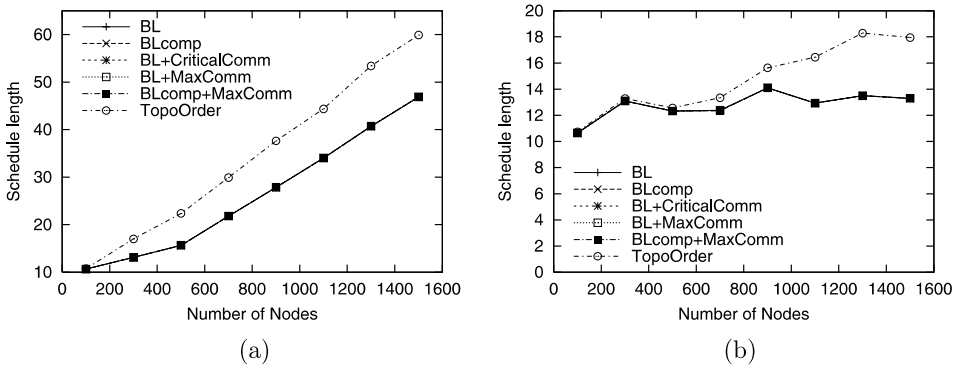

Fig. 5. Level based schemes; DAGs: edges/node = 2. (a) 32-fully connected, CCR = 0.1 and (b) 128-fully connected, CCR = 0.1.

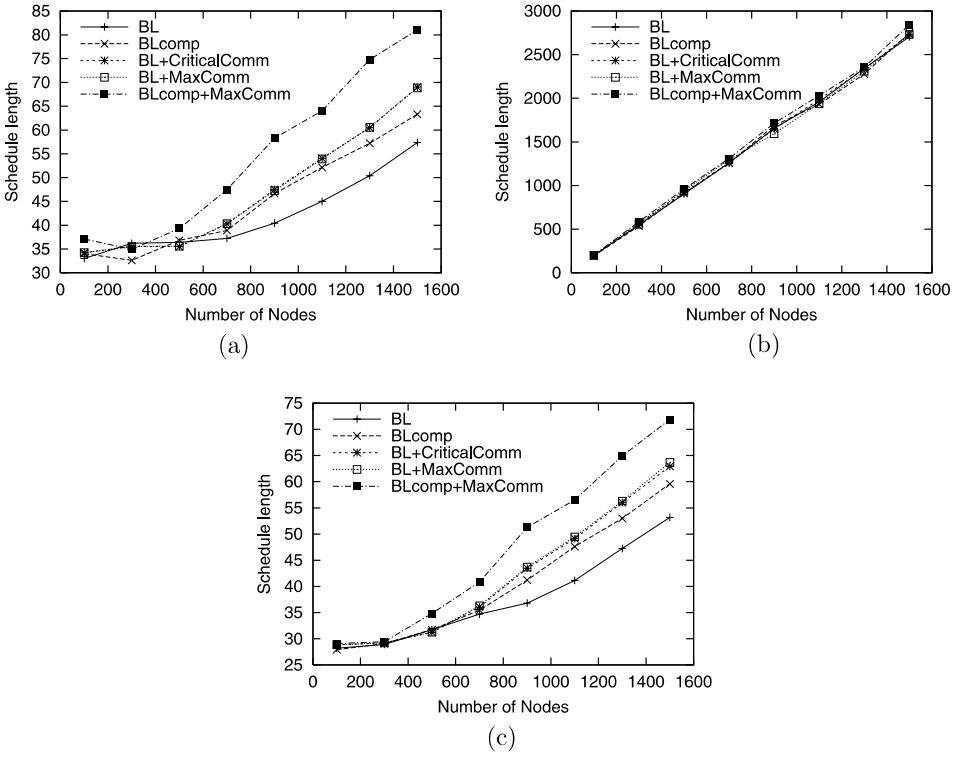

Fig. 6. Level based schemes; DAGs: edges/node = 2. (a) 32-fully connected, CCR = 10; (b) 8-ring, CCR = 10; and (c) 32 without contention, CCR = 10.

the computation bottom-level and the bottom-level differ significantly only for high CCRs. A possible explanation for these results is the zeroing of communication costs when the communicating nodes are scheduled to the same processor. Thus, the costs included in a priority due to communication might change after a node was scheduled, while the computation costs are fixed. Nevertheless, high CCRs show that it is important to consider communication and apparently the BL scheme is the best compromise.

It is interesting to notice that the difference between the level based schemes becomes more significant with the decrease of contention: for systems with few links (e.g. the ring used in Fig. 6b), there is only a small difference, while for fully connected systems the difference is significant (Fig. 6a). We confirmed this tendency by using these priority schemes for traditional list scheduling, without considering contention, for 8, 32 and 128 processors. Fig. 6c shows the results for 32 processors, CCR 10 and 2 average edges per node. For high communication the results are very similar to those obtained on the fully connected architecture considering contention (Fig. 6a). A possible explanation is that without considering contention the

communication costs of an edge reflect the communication time better than when contention comes into play.

5.2.2. CP based schemes

Next we compare the priority schemes based on the global critical path and we use BL, being the best of the previously compared schemes, as a reference. We repeat the experiments of Fig. 6a (32-fully connected, CCR = 10, average edges/node = 2) for these priority schemes and the results are presented in Fig. 7a. Among the three critical path schemes, CP_MaxComm clearly provides the worst performance. Using the largest communication as the criterion to choose the IBN is a local decision and cannot identify the importance of the IBN. The other two schemes CP_BL_TL and CP_TL perform similar and are very close to BL.

For low communication, the situation becomes different: Fig. 7b shows the results on a 32 processor bus machine with low communication (CCR = 0.1). CP_BL_TL and CP_TL perform notably worse than BL. Low communication needs a technique that can balance the load, which the schemes based on the CP apparently cannot. Even worse, in Fig. 7c the experiment of Fig. 7b is repeated but now with an average of 5 edges per node. All three critical path schemes yield results much worse than BL. Even the best of the three (CP_BL_TL) is up to 100% worse than BL. While in most experiments similar, CP_BL_TL performs in this experiment better than CP_TL. Overall, CP_BL_TL is the best of the critical path schemes. Again, results for the other experiments with the schemes based on the global critical path range qualitatively between the examples of Fig. 7a–c. For graphs with high communication and topologies with many links, CP_BL_TL and CP_TL have similar performance as BL, and are in some cases better than other schemes based on the bottom level. Increasing the number of edges per node, however, while fixing the CCR, always outcomes in worse results. A possible explanation is that for graphs with many edges there is no distinguished CP because several paths have similar length.

5.2.3. Cluster topologies

After the comparison of the priority schemes, we turn our attention to the behaviour of the extended list scheduling for the different systems' topologies. For better readability, the following figures show only four selected priority schemes, namely BL and BLcomp + MaxComm, and CP_BL_TL and CP_MaxComm, representing the best and the worst examined scheme of its kind, respectively. Fig. 8a depicts the results on a 32 processor ring architecture with medium communication (CCR = 1.0, average edges/node = 2). The schedule lengths obtained are up to several times as large as on the fully connected architecture (Figs. 6a and 7a), even though the CCR is lower than in those examples. So, the reduced speedup is due to the higher contention caused by the low number of communication links of the ring architecture. This effect is also observed on the bus machine, for example in Fig. 7b and c.

The effect of contention on architectures with few communication links can lead to schedule lengths higher than the sequential execution times. Fig. 8b shows such a case for high communication (CCR = 10, average edges/node = 5) on a eight proces-

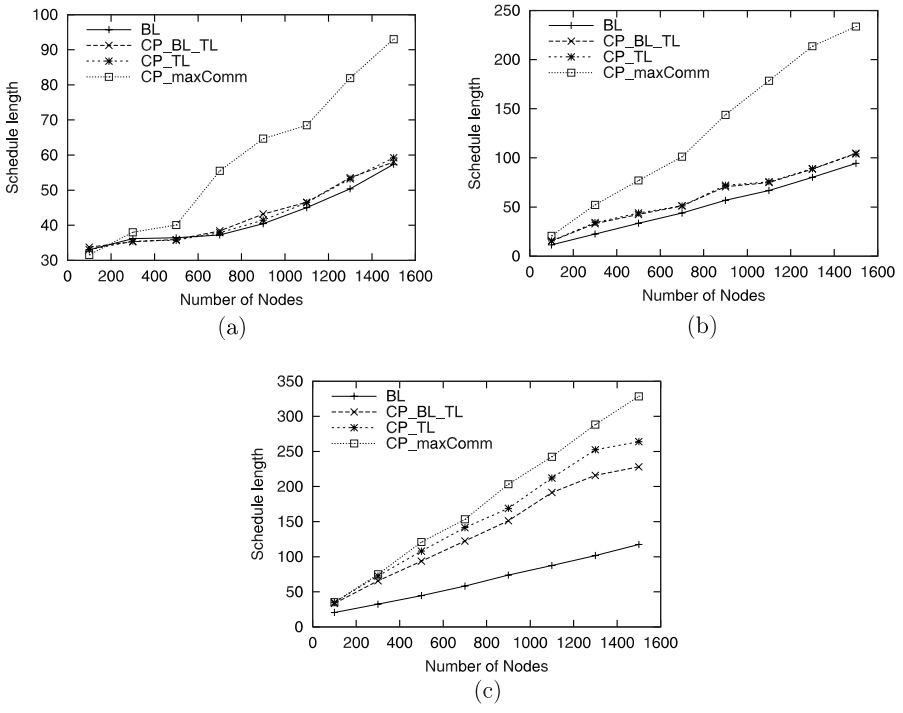

Fig. 7. CP based schemes. (a) 32-fully connected, CCR = 10, edges/node = 2; (b) 32-bus, CCR = 0.1, edges/node = 2; and (c) 32-bus, CCR = 0.1, edges/node = 5.

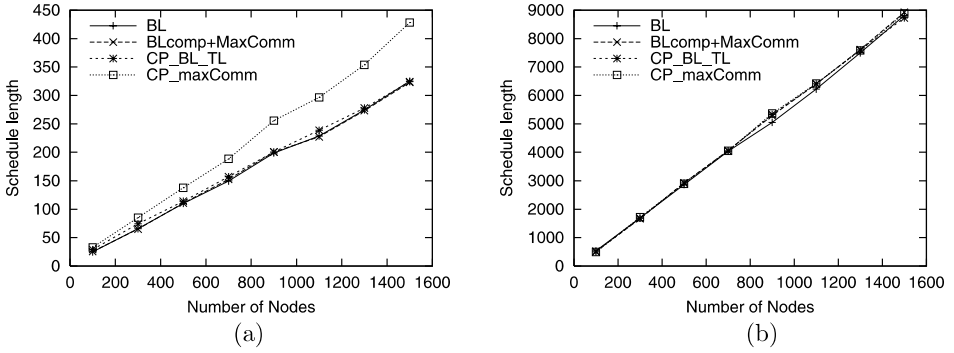

Fig. 8. High contention. (a) 32-ring, CCR = 1, edges/node = 2 and (b) 8-bus, CCR = 10, edges/node = 5.

sor bus machine. Another example are the results on a ring topology in Fig. 6b. Remember that the average node weight is 1, so the sequential execution time corresponds to the number of nodes of the DAG. In these cases the extended list

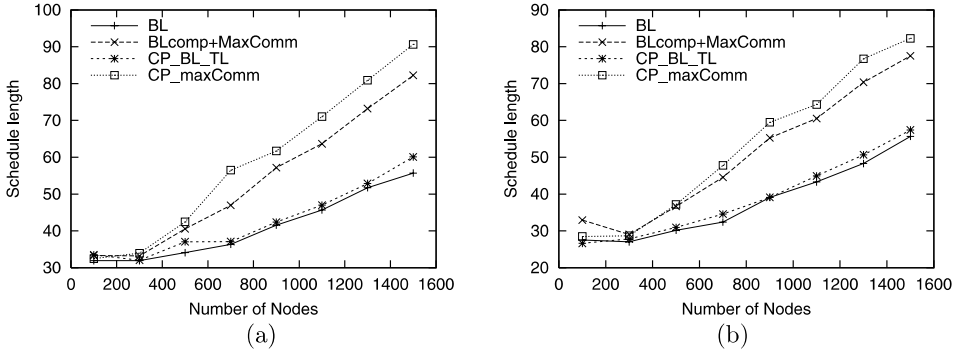

Fig. 9. Heterogeneity; DAGs: CCR = 10, edges/node = 2. (a) 32-fc, heterogeneous processors (b) 32-fc, fast linked processor pairs.

scheduling is not able to parallelise the graph efficiently. However, this situation can be easily identified when the schedule length is higher than the sequential execution time and the parallel schedule can then be substituted by the sequential schedule. In traditional list scheduling without considering contention this problem can only arise when the CCR is extremely high.

5.2.4. Heterogeneous cluster topologies

In the following, the appropriateness of our extended list scheduling for heterogeneous cluster architectures is analysed. In Fig. 9a graphs with a CCR of 10, and 2 average edges per node are scheduled by the selected schemes to the fully connected system with 32 heterogeneous processors. Since the DAGs are the same as for the experiments in Figs. 6a and 7a, where they are scheduled to a corresponding homogeneous system, the results can be directly compared with the results of those experiments. The differences are minimal and lead to the conclusion that our extended list scheduling is not affected by the heterogeneity (remember, the total processor speed is the same for both systems). This result is confirmed by the other experiments for heterogeneous systems.

In Fig. 9b we repeated the previous experiment on a 32 processor fully connected system with 16 faster links (10 times) each connecting a distinct pair of processors (NUMA architecture). For all priority schemes the schedule length is reduced in comparison to the homogenous system (Figs. 6a and 7a), so the extended list scheduling uses the advantage of the faster links, hence it is also appropriate for heterogeneous communication links. Once more, the results for the other systems with heterogeneous links are similar.

6. Conclusions

In this article traditional list scheduling was extended for the consideration of contention on communication links. The proposed extended list scheduling achieves

this goal by scheduling the edges of the DAG to the links of the target system, which may have an heterogeneous arbitrary architecture. Based on the extended list scheduling, we compared eight priority schemes for the generation of the nodes' schedule order. The schemes were employed for scheduling random graphs on systems taken from the area of cluster computing.

From the obtained results, the following can be concluded. Priority schemes based on the bottom-level are superior to schemes based on the global critical path. The difference between the schemes is even amplified with the increase of the average number of edges per node. The addition of communication costs to the bottom-level (BL + CriticalComm, BL + MaxComm) cannot improve the schedule length compared to the bottom-level only. Surprising is that the computation bottom-level (BLcomp) does not perform much worse than the bottom-level (BL) and the difference becomes even less significant with the increase of the average number of edges per node.

The qualitative behaviour of the schemes is not dependent on the topology of the target system, yet on the relative number of communication links. Few links emphasise the relevance of contention especially for DAGs with high communication. The differences between (computation) bottom-level based schemes decrease with the increase of contention, an interesting result confirmed with contention free list scheduling. This observation might lead to the conclusion that priority schemes for traditional and extended list scheduling should be chosen different. Nevertheless, the bottom-level scheme (BL) performed best for all experiments, with and without contention.

One shortcoming of the extended list scheduling is that it produces schedule lengths above the sequential execution times, when the communication to computation ratio of the DAG is high *and* the communication links of the target topology are sparse. However, an efficient parallel schedule might not be feasible inherent in the schedule problem and it is easy to substitute the result with the sequential version.

The results show that extended list scheduling is appropriate for scheduling DAGs to heterogeneous target systems with different topologies from the area of cluster computing. Heterogeneous systems are utilised as efficient as homogeneous systems and the schedule length benefits from faster communication links.

References

[1] T.L. Adam, K.M. Chandy, J.R. Dickson, A comparison of list schedules for parallel processing systems, Communications of the ACM 17 (1974) 685–689.
[2] C. Berge, Graphs and Hypergraphs, second ed., North-Holland, 1976.
[3] E.G. Coffman, R.L. Graham, Optimal scheduling for two-processor systems, Acta Informatica 1 (1972) 200–213.
[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, 1990.
[5] M. Cosnard, D. Trystram, Parallel Algorithms and Architectures, Int. Thomson Computer Press, London, UK, 1995.
[6] D.E. Culler, J.P. Singh, Parallel Computer Architecture, Morgan Kaufmann Publishers, 1999.

[7] A. Darte, Y. Robert, F. Vivien, Scheduling and Automatic Parallelization, Brinkhäuser, Boston, USA, 2000.
[8] H. El-Rewini, T.G. Lewis, Scheduling parallel program tasks onto arbitrary target machines, Journal of Parallel and Distributed Computing 9 (2) (1990) 138–153.
[9] A. Gerasoulis, T. Yang, A comparison of clustering heuristics for scheduling DAGs on multiprocessors, Journal of Parallel and Distributed Computing 16 (4) (1992) 276–291.
[10] R.L. Graham, Bounds for multiprocessing timing anomalies, SIAM Journal on Applied Mathematics 17 (2) (1969) 416–419.
[11] T. Hu, Parallel sequencing and assembly line problems, Operations Research 9 (1961) 841–848.
[12] J.J. Hwang, Y.C. Chow, F.D. Anger, C.Y. Lee, Scheduling precedence graphs in systems with interprocessor communication times, SIAM Journal of Computing 18 (2) (1989) 244–257.
[13] H. Kasahara, S. Narita, Practical multiprocessor scheduling algorithms for efficient parallel processing, IEEE Transactions on Computers C 33 (1984) 1023–1029.
[14] A.A. Khan, C.L. McCreary, M.S. Jones, A comparison of multiprocessor scheduling heuristics. in: Proceedings of International Conference on Parallel Processing, vol. 2, August 1994, pp. 243–250.
[15] B. Kruatrachue, Static task scheduling and grain packing in parallel processing systems, PhD Thesis, Oregon State University, USA, 1987.
[16] B. Kruatrachue, T. Lewis, Grain size determination for parallel processing, IEEE Software (January) (1988) 23–32.
[17] Y.-K. Kwok, I. Ahmad, Bubble scheduling: a quasi dynamic algorithm for static allocation of tasks to parallel architectures, in: Proceedings of Symposium on Parallel and Distributed Processing (SPDP), Dallas, TX, USA, October 1995, pp. 36–43.
[18] Y.-K. Kwok, I. Ahmad, Benchmarking the task graph scheduling algorithms, in: Proceedings of International Parallel Processing Symposium/Symposium on Parallel and Distributed Processing (IPPS/SPDP-98), Orlando, FL, USA, April 1998, pp. 531–537.
[19] Y.-K. Kwok, I. Ahmad, Link contention-constrained scheduling and mapping of tasks and messages to a network of heterogeneous processors, Cluster Computing: The Journal of Networks, Software Tools, and Applications 3 (2) (2000) 113–124.
[20] C.Y. Lee, J.J. Hwang, Y.C. Chow, F.D. Anger, Multiprocessor scheduling with interprocessor communication delays, Operations Research Letters 7 (3) (1988) 141–147.
[21] Z. Liu, A note on Graham's bound, Information Processing Letters 36 (October) (1990) 1–5.
[22] B.S. Macey, A.Y. Zomaya, A performance evaluation of CP list scheduling heuristics for communication intensive task graphs, in: Parallel Processing Symposium, 1998, Proceedings of IPPS/SPDP 1998, 1998, pp. 538 –541.
[23] M.A. Palis, J.-C. Liou, D.S.L. Wei, Task clustering and scheduling for distributed memory parallel architectures, IEEE Transactions on Parallel and Distributed Systems 7 (1) (1996) 46–55.
[24] C.H. Papadimitriou, M. Yannakakis, Towards an architecture-independent analysis of parallel algorithms, SIAM Journal of Computing 19 (2) (1990) 322–328.
[25] F.E. Sandnes, G.M. Megson, An evolutionary approach to static taskgraph scheduling with task duplication for minimised interprocessor traffic, in: Proceedings of the International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2001), Taipei, Taiwan, July 2001, Tamkang University Press, 2001, pp. 101–108.
[26] V. Sarkar, Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors, MIT Press, 1989.
[27] G.C. Sih, E.A. Lee, A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures, IEEE Transactions on Parallel and Distributed Systems 4 (2) (1993) 175–186.
[28] H. Singh, A. Youssef, Mapping and scheduling heterogeneous task graphs using genetic algorithms, in: Proceedings of Heterogeneous Computing Workshop (HCW'96), Honolulu, HI, April 1996, IEEE Computer Society, pp. 86–97.
[29] O. Sinnen, L. Sousa, Exploiting unused time slots in list-scheduling considering communication contention, in: Euro-Par 2001 Parallel Processing, Lecture Notes in Computer Science, vol. 2150, Springer-Verlag, 2001, pp. 166–170.

[30] O. Sinnen, L. Sousa, Scheduling task graphs on arbitrary processor architectures considering contention, in: High Performance Computing and Networking, Lecture Notes in Computer Science, vol. 2110, Springer-Verlag, 2001, pp. 373–382.
[31] L. Wang, H.J. Siegel, V.P. Roychowdhury, A.A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, Journal of Parallel and Distributed Computing 47 (November) (1997) 8–22.
[32] M.Y. Wu, D.D. Gajski, Hypertool: A programming aid for message-passing systems, IEEE Transactions on Parallel and Distributed Systems 1 (3) (1990) 330–343.