Exploiting Unused Time Slots in List Scheduling Considering Communication Contention

Oliver Sinnen and Leonel Sousa

Universidade Técnica de Lisboa, IST / INESC-ID
Rua Alves Redol 9, 1000 Lisboa, Portugal
{oliver.sinnen,las}@inesc.pt

Abstract. Static scheduling is the temporal and spatial mapping of a program to the resources of a parallel system. Scheduling algorithms use the Directed Acyclic Graph (DAG) to represent the sub-tasks and the precedence-constraints of the program to be parallelised. This article proposes an extention to the classical list scheduling heuristic that allows to schedule DAGs to arbitrary processor architectures considering link contention. In this extension, communication links are treated as shared resources likewise the processors and we improve the extended algorithm by exploiting unused time slots on the resources. The algorithm is experimentally compared to the existing heuristics BSA and DLS.

1 Introduction

The scheduling of a DAG (or task graph) is in its general form an NP-hard problem [1], i.e. an optimal solution cannot be calculated in polynomial time (unless $NP = P$). Many scheduling heuristics for near optimal solutions have been proposed in the past, e.g. surveys [2,3], following different approaches. Early scheduling algorithms did not take communication into account, but due to the increasing importance of communication for parallel performance, the consideration of the communication was included in the scheduling algorithms recently proposed. Most of these algorithms assume the target system as a homogenous system with fully connected processors and contention free communication resources. Very few algorithms model the target system as an arbitrary processor network and incorporate contention in the scheduling heuristic [4,5]. In [6], however, Macey and Zomaya showed that the consideration of link contention is significant to produce an accurate and efficient schedule.

This article presents an extension to the classical list scheduling heuristic [1] for scheduling on arbitrary processor networks with the consideration of link contention. The enhancement of the heuristic is achieved by treating communication links as shared resources likewise the processors. The new algorithm is further improved by exploiting unused time slots.

2 Models and Definitions

The DAG, as a graph theoretic representation of a program, is a directed and acyclic graph, where the nodes represent computation and the edges communi-

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 166–170, 2001.

cation. A weight $w(n_i)$ assigned to a node n_i represents its computation cost and a weight c_{ij} assigned to an edge e_{ij} represents its communication cost.

The topology of an arbitrary target system is represented as an undirected graph, where a vertex P_i represents a processor and an undirected edge L_{ij} represents a communication link between the incident processors P_i and P_j. Most scheduling algorithms, like the ones discussed here, assume a dedicated communication system.

We denote the start time of node n_i scheduled to processor P_k as $ST(n_i, P_k)$ and its finish time as $FT(n_i, P_k)$. The finish time of processor P_k is defined as $FT(P_k) = \max_i\{FT(n_i, P_k)\}$. After all nodes and edges of the DAG have been scheduled to the target system the schedule length is defined as $\max_k\{FT(P_k)\}$.

3 Extended List Scheduling

The first phase of list scheduling is the attribution of priorities to the nodes of the DAG and their ordering according to these priorities. Our Extended List Scheduling (ELS) heuristic uses the node's *bottom-level* $bl(n_i)$ as the measure for its priority . The bottom-level of a node is the longest path beginning with the node, where the length of a path is defined as the sum of the weights of its nodes and edges. The bottom-level has several properties beneficial for communication and contention aware list scheduling [7], and ordering the nodes by its bottom-levels in descending order automatically establishes an order according to the precedence-constraints of the graph [7]. Furthermore, in [8], eight priority schemes for contention aware list scheduling were experimentally compared and the bottom-level scheme performed best.

The main part of ELS iterates over the ordered node list and determines for every node the processor that allows its earliest *finish* time. Using the finish time instead of the starting time, takes the processor speed into account and allows the algorithm to be easily applied to heterogeneous target systems [8]. The node is scheduled on the chosen processor and the incoming edges of the node are scheduled on the corresponding communication links. By scheduling not only the nodes to the processors, but also scheduling the edges to the links, the algorithm achieves awareness of contention. Communication conflicts are detected as occupied links and the conflicting edge is delayed until the link is free. The route for the communication is determined by the static routing algorithm of the target architecture. Analogous to the processor, the start time and finish time of an edge on a link are denoted $ST(e_{ij}, L_{lk})$ and $FT(e_{ij}, L_{lk})$, respectively. Now, the data ready time $DRT(n_i, P_k)$ of a node n_i on processor P_k is defined as the time when the last communication from its parent nodes arrives, $DRT(n_i, P_k) = \max_j\{FT(e_{ji}, L_{lk})\}$. For a valid schedule, $ST(n_i, P_k) \geq DRT(n_i, P_k)$ must be true for all nodes.

The basic ELS algorithm determines the start and the finish time of a node on a processor at the end of the nodes already scheduled to that processor. The earliest start time of a node at the end is equal or greater to the finish time of

processor P. The node's finish time is then its start time plus its computation time.

$$ST(n_i, P) = \max\{FT(P), DRT(n_i, P)\}$$
$$FT(n_i, P) = ST(n_i, P) + w(n_i)$$

For the improvement of the basic ELS algorithm (called ELS-slot), we utilise unused time slots between nodes. To schedule a node n_i between two already scheduled nodes, a time slot large enough to accommodate n_i must be found. Assume the l nodes $n_{p_1}, n_{p_2}, \ldots n_{p_l}$ are scheduled in this order on the processor P. Let k_0 be the smallest k $(1 \leq k \leq l-1)$ that fulfils $w(n_i) \leq ST(n_{p_{k+1}}, P) - \max\{FT(n_{p_k}, P), DRT(n_i, P)\}$. The start and finish time of $_{n_i}$ are then

$$ST(n_i, P) = \max\{FT(n_{p_{k_0}}, P), DRT(n_i, P)\}$$
$$FT(n_i, P) = ST(n_i, P) + w(n_i).$$

If there is no such k_0, then the start and finish time are determined as in ELS, i.e. the node is scheduled at the end. k_0 corresponds to the node's earliest start and finish time.

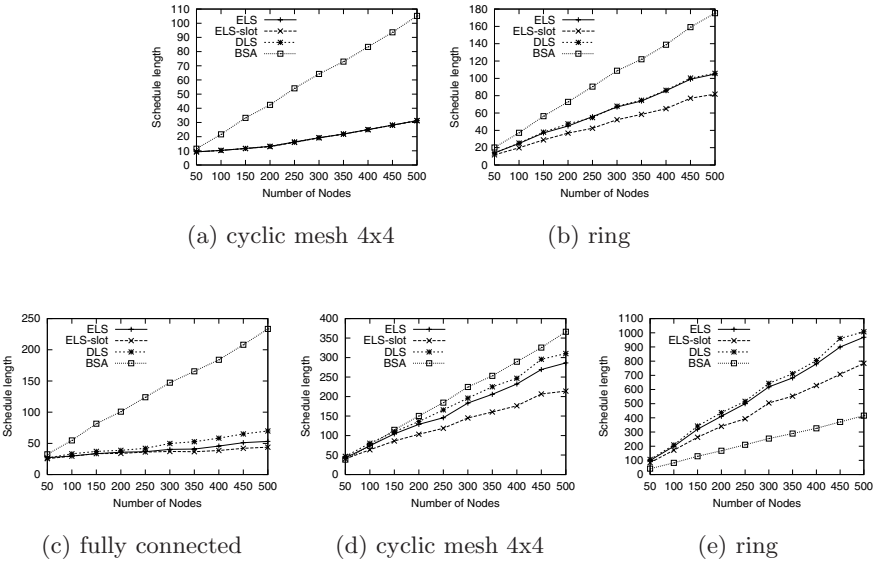
The start and the finish time of an edge on a link are determined correspondingly. ELS's complexity is $O(Vlg(V) + P(V + E \cdot O(Routing))$ and ELS-slot has a complexity of $O(V^2 + P \cdot E \cdot O(Routing) + P^2E^2)$, where $O(Routing)$ is the routing complexity of the target machine.

4 Experiments and Conclusions

ELS and ELS-slot were implemented and used to schedule random graphs to different architectures. The obtained schedule lengths are compared to the ones of BSA [4] and DLS [5] (streamlined version for homogeneous processors), which are also contention aware scheduling algorithms for arbitrary processor networks. DLS's complexity is similar to ELS's and BSA's complexity is similar to that of ELS-slot. The random graphs generated had the following characteristics: graph size 50-500 nodes; average number of edges per node 1.5, 2 or 5; node weight was uniformly distributed in [0.1, 1.9], i.e. average 1; edge weight was determined in a way that the global Communication-to-Computation Ratio (CCR) was 0.1, 1 or 10. The target machine architectures are (cyclic) meshes, rings and fully connected machines with 4, 8 and 16 processors. The results shown in Fig. 1 - every point corresponds to the average of 10 graphs - are for 16 processors and graphs with 2 average edges per node. The other results are similar in nature.

For low (CCR 0.1, Fig. 1a) and medium communication (CCR 1, Fig. 1b), BSA performs worse than the other algorithms ($> 300\%$ for CCR 0.1 and $> 60\%$ for CCR 1). The difference between ELS and DLS is negligible and ELS-slot performs best for CCR 1, with an improvement of about 30 % over ELS.

For high communication (CCR 10, Fig. 1c-1e), the relevance of the target system's topology increases and BSA yields the best results for the ring topology, where links are sparse (Fig. 1e), even though with a minimal speedup ($< 25\%$).



(a) cyclic mesh 4x4 (b) ring

(c) fully connected (d) cyclic mesh 4x4 (e) ring

Fig. 1. Schedule results with CCR 0.1 (a), CCR 1 (b), CCR 10 (c-e)

While the other algorithms yield here schedule lengths higher than the sequential execution time, they perform better than BSA on the other architectures (Fig. 1c, 1d). ELS performs always better than DLS (about 5%-20%), due to a better priority choice and ELS-slot can improve ELS by about 20%-30%.

References

1. M. Cosnard and D. Trystram. *Parallel Algorithms and Architectures*. Int. Thomson Computer Press, London, UK, 1995. 166
2. A. Gerasoulis and T. Yang. A comparison of clustering heuristics for scheduling DAGs on muliprocessors. *Journal of Parallel and Distributed Computing*, 16(4):276–291, December 1992. 166
3. Y. Kwok and I. Ahmad. Benchmarking the task graph scheduling algorithms. In *Proc. of Int. Par. Processing Symposium/Symposium on Par. and Distributed Processing (IPPS/SPDP-98)*, pages 531–537, Orlando, Florida, USA, April 1998. 166
4. Y. Kwok and I. Ahmad. Bubble Scheduling: A quasi dynamic algorithm for static allocation of tasks to parallel architectures. In *Proc. of Symposium on Parallel and Distributed Processing(SPDP)*, pages 36–43, Dallas, Texas, USA, October 1995. 166, 168
5. G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):175–186, February 1993. 166, 168

6. B. S. Macey and A. Y. Zomaya. A performance evaluation of CP list scheduling heuristics for communication intensive task graphs. In *Parallel Processing Symposium, 1998. Proc. of IPPS/SPDP 1998*, pages 538 –541, 1998. 166
7. O. Sinnen and L. Sousa. Scheduling task graphs on arbitrary processor architectures considering contention. In *Int. Conf. on High Perf. Computing and Networking*, Lecture Notes in Computer Science, Amsterdam, Netherlands, June 2001. 167
8. O. Sinnen and L. Sousa. Comparison of contention aware list scheduling heuristics for cluster computing. In *Workshop on Scheduling and Resource Management for Cluster Computing (ICPP 2001)*, Valencia, Spain, September 2001. IEEE Computer Society Press. (to be published). 167