Compile-Time Task Scheduling for Multi-Phase Programming

A. Benaini, D. Laiymani

Université de Franche-Comté - LIB
16 Route de Gray 25030 Besancon - France
e-mail : [benaini,laiymani]@comte.univ-fcomte.fr

Abstract. This paper presents a compile time scheduling algorithm dedicated to parallel machines in which the interconnection network can be altered during the execution of the same application. We propose an heuristic technique taking ideas from clustering and list scheduling algorithms to schedule a task graph on these machines. We give an upper bound to the performances of the algorithm that generalizes the one given by Gerasoulis and Yang [7].

1 Introduction

We present a compile-time scheduling technique that schedules precedence task graph onto a reconfigurable multiprocessor architecture. A machine is *reconfigurable* if its interconnexion network can be altered between different phases of the same algorithm execution. In this paper we consider reconfigurable machines with p processors and d communication links per processor. These multiprocessors systems yield a variety of possible topologies for the network where a possible topology is any one in which the number of connections per processor is less than or equal to d.
A parallel algorithm for a reconfigurable machine is implemented as a series of phases [6] (multi-phase programming). Each phase is then executed onto the processor graph that reflects, in the best way, the needs of the current data transfer pattern [1].

A schedule of a DAG for a reconfigurable machine (or a multi-phase scheduling) is then defined as the assignment to each task of a tuple <phase, processor, starting time>. A phase is a sub-graph of the initial DAG which can be executed, *without routing*, on a fixed topology of a reconfigurable machine. In order to obtain efficient multi-phase schedules, some conflicting goals have to be satisfied. First, a maximum of tasks and links must be assigned to each phase. In this way the inherent parallelism of the DAG will be kept as much as possible. Furthermore, for each phase, communication overheads have to be minimized by assigning to the same processor, tasks which have to exchange costly messages. Finally, it is necessary to reduce idle times between the execution of two consecutive phases.

2 Problem definition and general approach

Let a reconfigurable machine with p processors and d communication links per processor. The network is configured at the begining of each phase and execution of the series of phases is sequential. Reconfiguration overheads are assumed to be null [2].

Let a DAG $G = (V, E)$ where $V = \{n_j, j = 1, 2 \ldots v\}$ is the set of tasks and $E = \{e_{i,j} = (n_i, n_j)\}$ is the set of arcs. $c_{i,j}$ represents the communication overhead between n_i and n_j (which becomes zero if n_i and n_j are mapped to the same processor). τ_i is the computation cost of task n_i. The *length* of a path is the sum of all task computation and edge communication costs in the path. The *critical path* is the path with the longest length in G.

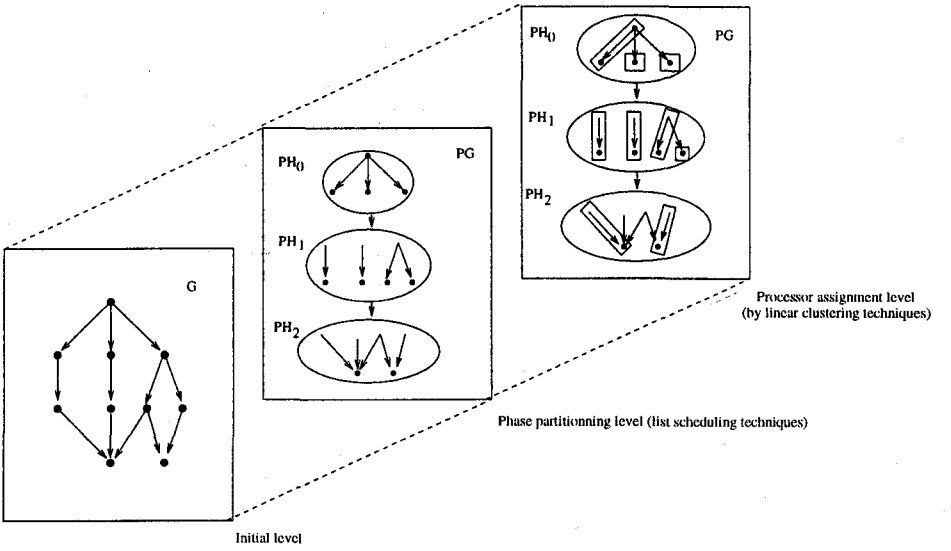

Fig. 1. The assignment levels of our scheduling method

Figure 1 shows the different levels of assignment that requires a scheduling for reconfigurable machines. The initial level is the tasks graph. The second level consists in partitionning G into a series of phases. The processor assignment level defines the assignation of tasks to processors for each phase. A phase $PH = (VP, EP)$ is a sub-set of G such that $VP \subseteq V$ and $EP \subseteq E$ (an arc in EP may have only in-node in VP, see figure 1). A multi-phase program is then defined as a sequence of *disjoint phases* PH_s, $0 \leq s < q$ such that

1. $\bigcup_{s=0}^{q-1} PH_s = G$,
2. the *phases graph* $PG = (GV, GE)$ with $GV = \{PH_s, s = 0, 1, \ldots q-1\}$ and $GE = \{(PH_s, PH_t) : \exists n_i \in PH_s, n_j \in PH_t, (n_i, n_j) \in E\}$ is a DAG (phases are executed sequentially),

3. the execution of each phase must be performed without routing cost on a fixed topology of a (p, d)-reconfigurable machine i.e. an edge in G will correspond to one communication link of the reconfigurable machine.

A *multi-phase* schedule is a partition of G into disjoint phases $PH_s, 0 \leq s < q-1$, a mapping of tasks to processors and the assignment of execution starting time for each task such that $\sum_{s=0}^{q-1} ET(PH_s)$ is minimized where $ST(n_i)$ is the starting time of n_i, $ET(PH_s)$ is the execution time of a phase PH_s and

$$ET(PH_s) = \max_{n_i \in VP_s} \{ST(n_i) + \tau_i\}$$

As phases are processed sequentially, the parallel execution time of the schedule is

$$PT = \max_{n_i \in V} \{ST(n_i) + \tau_i\} = \sum_{s=0}^{q-1} ET(PH_s)$$

Clearly this scheduling problem is NP-complete (this follows by a simple reduction from the *Scheduling on a Clique* [3]). Our heuristic algorithm produces efficient solutions in polynomial time. The algorithm builds the different phases one by one in a sequential way. It computes two assignment processes which are an assignment of tasks to phases and an assignment of tasks to processors. Informally, it scans the different tasks of G and according to a certain priority function it assigns the choosen task to the current phase. While a phase is processed we apply a modified clustering technique [4, 5, 7] in order to assign tasks to processors. Then we will obtain "clusterized" phases such that, clusters will correspond to processors. In this way we try to well match the physical properties of the target reconfigurable machine.

3 A scheduling algorithm for multi-phase programming

The algorithm is a succession of refinement steps in order to build phases, one by one, starting by phase PH_0. A refinement assigns a task to the current phase.

The evaluated finishing time of a task n_x, say $EFT(n_x)$, is equal to the length of the longest path in the "clusterized" phase PH_s from any entry node to n_x (including τ_x). In the same way, $EPT(PH_s)$ is the length of the longest path of phase PH_s. Now, let $n_1, n_2, \ldots, n_x$ be the set of ready tasks which are both candidates to a possible assignment in phase PH_s. We begin by selecting a task for possible assignment to PH_s. This task is the one minimizing $EPT(PH_s)$. A tie is broken by choosing the task which minimizes $EPT(PH_s) - EFT(n_x)$. This last criterion will reduce the waiting time between phase PH_s and PH_{s+1} induced by the assignment of a task to phase PH_s.

The assignment of a ready task n_k in phase PH_s is accepted if the linear clustering procedure generates a number m of clusters $m_0, m_1, \ldots m_{m-1}$ with $m \leq p$ and the degree of $m_i \leq d, \forall i = 0, 1, \ldots m-1$. When no assignment are possible the algorithm starts to build phase PH_{s+1}.

In this way *each phase* PH_s, $0 \leq s < q$ *is executed on a* (p,d)*-reconfigurable machine without routing cost* [2].

Now, we give a bound on the quality of the scheduling. In order to fit the particularities of multi-phase scheduling, we adapt the granularity definition given by Gerasoulis and Yang in [7] to the multi-phase model as follows. For a task $n_x \in V$ we define $g(n_x) = \tau_x / max_j(c_{jx}, c_{xj})$. Then, the granularity of G is given by $g(G) = min_{n_x \in V}(g(n_x))$.

Theorem 1. *Let* $PT_{opt}(PG)$ *denotes the optimal parallel time of the phase graph* PG *i.e. the length of the longest path in this graph and* PT_{mps} *the parallel time of the multi-phase scheduling algorithm. Then*

$$PT_{opt} \leq PT_{mps} \leq (1 + \frac{1}{g(G)}) L(PG) PT_{opt}(PG)$$

Moreover for coarse grain DAG $(g(G) \geq 1)$ *we have*

$$PT_{mps} \leq 2 \times L(PG) \times PT_{opt}(PG)$$

A proof of this theorem is given in [2]. If the number of processors is unbounded and if they are fully connected, the algorithm clearly generates an unique phase implying that, $PT_{mps} \leq (1 + \frac{1}{g(G)})) \times PT_{opt}$. Furthemore if G is coarse grain we obtain $PT_{mps} \leq 2 \times PT_{opt}$ which is the result given in [7].

Finally, to test our algorithm we are building a prototype named PHARAON (PHAse Reconfigurable Automatic cOde geNerator) that generates code for real reconfigurable architectures using the multi-phase programming paradigm.

References

1. J-M. Adamo and L. Trejo. Programming Environment for Phase-reconfigurable Parallel Programming on Supernode. *JPDC*, 23:278–292, 1994.
2. A. Benaini and D. Laiymani. Compile-time Task Graph Scheduling for Multi-phase Programming. Technical report, LIB - http://comte.univ-fcomte.fr, 1996.
3. M.R. Garey and D.S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.
4. M.A. Palis, J. Liou, and D.S.L. Wei. Task Custering and Scheduling for Distributed Memory Parallel Architectures. *IEEE Trans. on Parallel and Distributed Systems*, 7(1):46–55, 1996.
5. V. Sarkar. *Partitionning and Scheduling Parallel Programs for Execution on Multi-processors.* M.I.T. Press, 1989.
6. L. Snyder. Introduction to the Configurable, Highly Parallel Computer. *J. Par. Distr. Comp.*, 15:47–56, 1982.
7. T. Yang. *Scheduling and Code Generation for Parallel Architectures.* PhD thesis, Rutgers University, 1993.