Fast Scheduling and Partitioning Algorithm in the Multi-processor System with Redundant Communication Resources*

Eryk Laskowski

Institute of Computer Science
Polish Academy of Sciences
01-237 Warsaw, Ordona 21, Poland
laskowsk@ipipan.waw.pl

Abstract. Look-ahead dynamic inter-processor connection reconfiguration is a multi-processor architectural model, which has been proposed to eliminate connection reconfiguration time overheads. It consists in preparing link connections in advance in parallel with program execution. An application program is partitioned into sections, which are executed using redundant communication resources. Parallel program scheduling in such a kind of environment incorporates graph partitioning problem. The paper presents a scheduling algorithm for look-ahead reconfigurable multi-processor systems. It is based on list scheduling and utilizes a fast section clustering heuristic for graph partitioning. The experimental results are compared with results of a genetic graph partitioning algorithm.

1 Introduction

The aim of this paper is to present an efficient task scheduling algorithm for look-ahead reconfigurable multi-processor systems. In distributed memory multi-processor systems with message passing, link connection reconfiguration is a very promising alternative to fixed interconnection networks. Processors are connected using reconfigurable, direct, point-to-point connections thus retransmission of messages through intermediate nodes is eliminated. The topology and duration of inter-processor connections can be adjusted according to program needs.

Link connection reconfiguration in all known systems involves overheads in the communication execution time. To eliminate these overheads, a new approach called look-ahead dynamic link reconfiguration [1] has been proposed. It consists in preparing link connections in advance in parallel with program execution. An application program is partitioned into sections and link connections are prepared for the next program sections while previous sections are executed. A special architectural solution is necessary to provide connection reconfiguration in advance. In this paper, we investigate the system with multiple link switching devices.

* The paper partially sponsored by the National Grant KBN No. 7 T11C 015 20

R. Wyrzykowski et al. (Eds.): PPAM 2001, LNCS 2328, pp. 97–106, 2002.

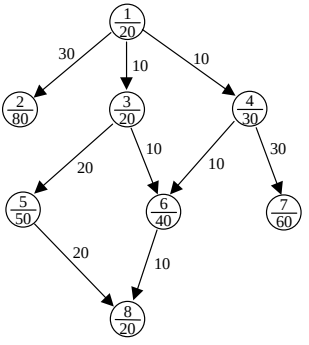

Fig. 1. An example of DAG.

The scheduling problem in reconfigurable multi-processor system has been previously considered only for statically established connection configurations (when connections are set before the execution of a program) or for single-switch multi-processor systems with dynamic on-request reconfiguration [4]. The algorithm presented in [3] has addressed a problem of program partitioning into sections. This paper presents scheduling algorithm in look-ahead reconfigurable multi-processor systems and the new fast heuristics of program graph partitioning into sections, which is shown to be more efficient than the algorithm presented in [3].

2 A Parallel Program Model

A parallel program is represented as a weighted Directed Acyclic Graph (DAG). A DAG is a tuple **G**=(**V**,**E T**, **C**), where

- **V**= $\{n_i : i = 1..v\}$ is a set of nodes, $v =\mid \mathbf{V} \mid$; a node represents an indivisible task of the program;
- **E**= $\{e_{i,j} : i = 1..v, j = 1..v\}$ is a set of directed edges; an edge $e_{i,j} \in \mathbf{E}$, which connects nodes n_i and n_j, represents a communication from the task n_i to the task n_j;
- **T**= $\{t_i : i = 1..v\}$ is a set of computation costs; the value $t_i \in \mathbf{T}$ is the cost of execution of the task n_i;
- **C**= $\{c_{i,j} : i = 1..v, j = 1..v\}$ is a set of communication costs; the value $c_{i,j}$ is the cost of communication from the task n_i to the task n_j.

Program is executed according to the *macro-dataflow* [5] model. An example of DAG is shown in Fig. 1.

3 The Look-Ahead Reconfigurable Multi-processor System

The look-ahead dynamic connection reconfiguration assumes anticipated connection setting in some redundant communication resources provided in the sys-

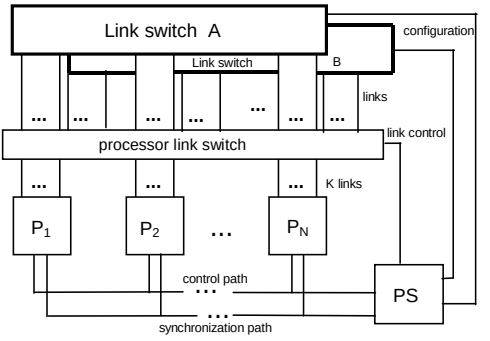

Fig. 2. The look-ahead reconfigurable multi-processor system.

tem. We investigate a system with two crossbars as redundant link connection switches, see Fig. 2. It is a multiprocessor system with distributed memory and with communication based on message passing. It consists of **N** identical worker processor subsystems $\mathbf{P_1}...\mathbf{P_N}$, control subsystem **PS**, control path, link connection switches (crossbars) **A** and **B**. Each working processor $\mathbf{P_i}$ has a **K** (**K<<N**) communication links $\mathbf{L_{i1}} \ldots \mathbf{L_{iK}}$ connected to the crossbars **A** and **B** through the processor link switch. Worker processor can execute a task and communicate at the same time. Connections are prepared by a global control subsystem **PS** in the switches **A** and **B** that are interchangeably used as the active and configured communication resources. Reconfiguration is done sequentially. Reconfiguration of the single connection takes **R** time units.

An application program is partitioned into sections which assume fixed direct inter-processor connections set in active communication resources. The connections for the next program sections are prepared in the configured communication resources in parallel with current program execution. At the section boundary processor links are switched between resources **A** and **B** in a very short time and the execution of next program sections is enabled. The program execution control is done by the **PS** subsystem. It collects messages on link occupation sent using control path and initializes link reconfiguration. In parallel with reconfiguration, the synchronization of the processors for execution of the next section is performed using synchronization path. The simplest implementation of such path is a bus, but more efficient solution can be done with hardware barrier synchronization. In this paper we are interested in the asynchronous processor-restrained control strategy, where connection switching is controlled at the level of dynamically defined processor clusters.

3.1 The Program Schedule

Task execution order is determined by the program schedule. As opposed to system with simple on-request reconfiguration [4], in the look-ahead reconfigurable environment the schedule also consists of program partitioning into sections.

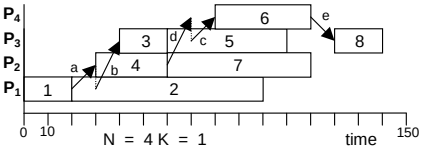

Fig. 3. A Gantt chart for the schedule of DAG from Fig. 1.

Schedule is defined as task-to-processor and communication-to-link assignment with specification of starting time of each task and each communication (see Fig. 3). Assignment preserves the precedence constraints coming from the program graph and from the assumed execution model. A processor can execute one task at the moment. A communication is asynchronous and non-blocking.

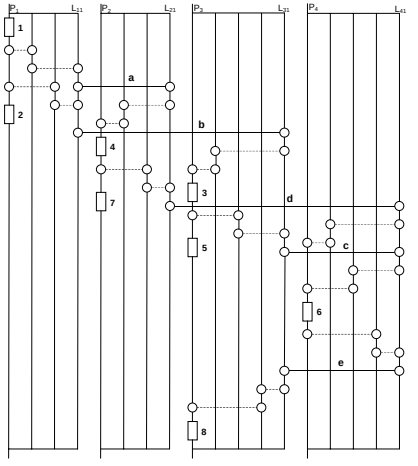

Fig. 4. Modeling of schedule from Fig. 3 in terms of the APG.

It occupies one link of sending processor and one link of receiving processor. A link can be used for transmission of one message at the moment. Intra-processor communication cost is negligible.

A program with specified schedule is expressed in terms of the Assigned Program Graph (APG), which assumes the synchronous communication model (CSP-like). It allows us to use some existing software tools for profiling parallel program execution time in the look-ahead inter-processor connection reconfiguration [3]. An APG consists of two kinds of nodes: the non-communicating code nodes (which correspond to tasks in DAG, shown as rectangles in Fig. 4) and communication instruction nodes (circles in Fig. 4). There are two kinds of edges in the graph: activation edges (vertical lines in Fig. 4) and communication

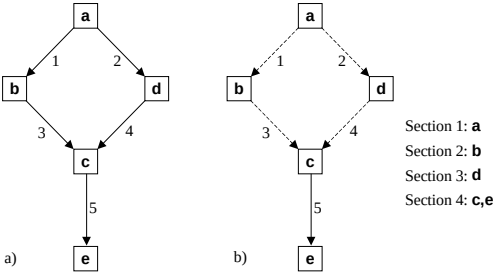

Fig. 5. a) Communication Activation Graph for APG from Fig. 4.
b) CAG graph partitioned into sections.

edges (horizontal lines, solid for inter-processor and dashed for intra-processor communications in Fig. 4). Two communication nodes and the connecting them inter-processor communication edge correspond to a communication in DAG. Each processor is assigned a subgraph built of several activation paths with nodes, which are scheduled on this processor. Asynchronous, non-blocking communications in a look-ahead reconfigurable environment requires modeling by CSP-like communications as used in APG, which is done as follows. Processor link behavior is modeled as a set of activation paths parallel to the computation path (marked as L_{i1} on Fig. 4) since each link works independently of processor and others links. Each communication is modeled as an activation path on the sender processor, which is used for sending a message to the link path. The communication is modeled in a similar manner on the receiver processor. Due to these paths, communication is sent without blocking the sending processor and, on the receiver side, link can be freed immediately after reception of the message. Program sections are defined by identification of such subgraphs in the APG that the validity conditions hold. They assure the correct execution of many sections in parallel in the system with the look-ahead created connections. A valid program graph partitioning for the processor-restrained strategy takes into account external communications only (labeled with **a...e** in Fig. 4) and has to fulfil the following conditions:

a) Section subgraphs corresponding to program sections are mutually disjoint in respect to external communication edges. Each communication edge belongs to one and only one section. Section boundaries can cross activation edges.
b) Section subgraphs are connected in respect to activation and communication edges.
c) Section subgraphs are complete in respect to activation paths. It means that a section contains all communication edges, which are incident to all communication nodes on all activation paths that can be found between any two communications nodes, which belong to the section.
d) A correct partition shows stability of inter-processor link connections inside sections. Processor link connections inside section subgraphs do not change.

To enable an easier partitioning analysis we introduce another program graph representation, which is called communication activation graph (CAG). This graph is composed of nodes, which correspond to external communication edges of the APG and of edges, which correspond to activation paths between communication edges of the APG, Fig. 5a. The rules for partitioning communication activation graphs are as described below.

- A partition of a program into sections defines a partition of the communication activation graph into disjoint subgraphs.
- The edges which connect nodes contained in a section subgraph in the communication activation graph define a connected subgraph when considered as undirected.
- All nodes on each path, which connects two nodes belonging to a section subgraph belong to the same section.

A partition of an exemplary program communication activation graph into sections is shown in Fig. 5b. Communication activation edges, which do not belong to any section are denoted by dashed lines.

4 Scheduling in the Look-Ahead Reconfigurable Systems

The goal of the scheduling algorithm, presented in this paper, is to produce schedules for the given DAG, such that execution time of the specified program is minimized. The proposed algorithm is based on the idea of list scheduling. It is a commonly used heuristic, since the scheduling problem has been proved to be NP-complete [5] for general task graphs with communication.

The proposed algorithm is improved versions of ETF (Earliest Task First) scheduling, proposed by Hwang et al. [2]. The algorithm schedules ready tasks. At each scheduling step, the earliest schedulable task is assigned first. The earliest starting time *est* of the task n_i is determined by preceding tasks allocations, their completion time, communication delays and the current time point. The main difference from original version of ETF is that instead of fixed inter-processor network topology, we investigate system with look-ahead dynamically created connections. It means that we have to take into account a limited number of links, links contention, connection reconfiguration and section activation time overheads. Modification of ETF consists in the new formula used for evaluation of the earliest starting time. The flow chart of *Ready* procedure used in modified ETF is given in Fig. 6. *Ready* (n_i, P_i) returns the time when the last message for task n_i will arrive at processor P_i. There are additional communication time overheads that simulate link reconfiguration control when network topology should be changed. Overheads are introduced into total communication time when link reconfiguration is necessary and there is no sufficiently long time gap after last communication to do reconfiguration in advance and without delaying program execution. The algorithm minimizes these overheads by reduction of the number of link reconfigurations.

```
Procedure Ready(n_i, P_i)
Time := 0
For each n_j ∈ Predecessors n_i
     T_Arrive := finishing time of task n_j
     P_j := processor which task n_j is scheduled on
     If P_j ≠ P_i Then
         T_Arrive := T_Arrive + c_j,i
         If P_i and P_j are connected Then
             send := link of P_j connected to P_i
             recv := link of P_i connected to P_j
         Else
             send := last recently used link of P_j
             recv := last recently used link of P_i
             If time since last use of link L_j,send or link L_i,recv
                 in previous configuration < R   Then
                     T_Arrive := T_Arrive + R
             EndIf
         EndIf
         Allocate communication e_j,i on links L_j,send and L_i,recv
     EndIf
     If Time < T_Arrive Then
         Time := T_Arrive
     EndIf
EndFor
Return Time
```

Fig. 6. The Ready procedure used in scheduling algorithm.

4.1 The Fast Graph Partitioning Algorithm

The graph partitioning algorithm (see Fig. 7) is a second phase of the scheduling procedure. The algorithm starts with initial partitioning, which consists of sections built of single communications. In every step, a vertex of CAG is visited and the algorithm tries to include current vertex to existing sections. The vertices are visited once in the order of their ascending finishing time, which is taken from the program schedule. Sections for clustering are determined by examination of incoming edges of the current vertex. The heuristic tries to find such a subset of those sections, which can be joint (including current vertex) into single section that doesn't break rules of graph partitioning (see Sect. 3.1) and thus, can be used in a section clustering. The subset, which gives the shortest program execution time, is selected. In case of a tie, the heuristic selects the biggest subset, which leads to reducing the number of sections and thus, could reduce section activation time overheads. When section clustering doesn't give any execution time improvement, the section of the current vertex is left untouched and one of two communication resources is assigned to it. As with section clustering, the choice of the resource depends on program execution time. In the next heuristics steps, this section could be joint to the vertex's successors section(s).

4.2 Evaluation of Program Execution Time

Program execution time is estimated by measuring the time of the symbolic execution of the program graph partitioned into sections. Program execution is simulated by the execution of the graph in a modeled look-ahead dynamically reconfigurable system [3]. The **APG** graph of the program is automatically extended at the section boundaries by subgraphs, which model the look-ahead

```
Begin
    T_best := max value of program execution time
    B := initial set of section, each section is composed of single
         communication
    For each vertex v of communication activation graph taken in
                              the order of ascending finishing time {1}
        P := set of all predecessors of v
        S := set of sections that contain communications from P, S ⊂ B
        M_best := ∅
        For each subset M ⊂ S
            C := set of all communications that are contained
                 in sections from M
            If v and C joint into single section
               preserve the conditions of valid graph partitioning {2}
            Then
                T := program execution time with
                     v and C placed in single section
                If T < T_best or (T = T_best and |M| > |M_best|) Then
                    T_best := T
                    M_best := M
                EndIf
            EndIf
        EndFor
        If M_best ≠ ∅ Then
            B := B - M_best
            Include to B a new section built of v and communications
                    that are contained in sections in M_best
        Else
            s := section that consists /only/ of communication v
            Assign communication resource to section s
        EndIf
    EndFor
End
```

Fig. 7. The general scheme of the fast graph partitioning algorithm.

reconfiguration control, see Fig. 8. The functioning of the control path, synchronization path and the global control subsystem PS is modeled with the use of program graph extensions. The bus is used as control and synchronization paths (see Fig. 2) and it is simulated as a subgraph, which contains concurrent processes corresponding to transmissions over the bus. The reconfiguration process is modeled by pairs of parallel input communications followed by a sequential process, which corresponds to crossbar reconfiguration latency. The activation of application processes after reconfiguration is done by a broadcast transmission over the bus.

5 Experimental Results

The results of the presented algorithm are compared to a scheduling algorithm, which applies genetic graph partitioning. This algorithm uses the same modified ETF heuristic in the first phase. The second phase, which consists of scheduled program graph partitioning by a genetic algorithm, has been presented in [3].

Evaluation results of the algorithm performance are shown in Fig. 9. It shows the speedup

$$S = (T_{\text{gen}} - T_{\text{fast}})/T_{\text{gen}} \times 100$$

in total execution time T_{fast} of a program partitioned by the fast algorithm against the execution time T_{gen} obtained with the use of the genetic heuristic.

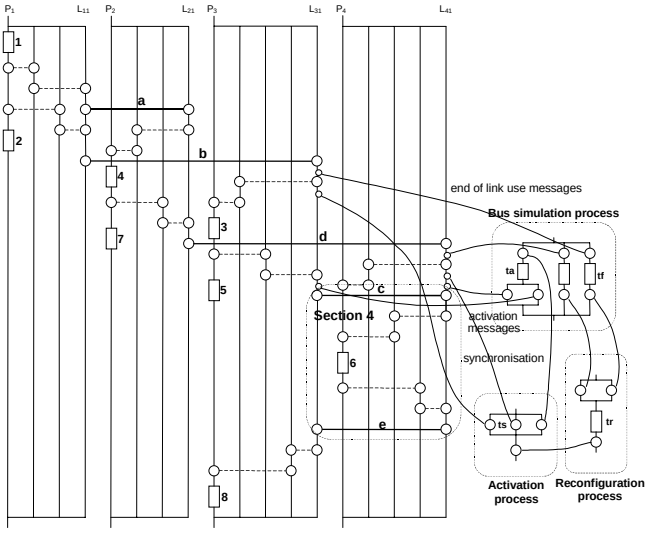

Fig. 8. Modeling the reconfiguration control in an APG.

The results are shown for two exemplary program graphs, executed in the look-ahead environment with the following program and system parameters (see Sect. 3 and [3] for definitions):

t1 : $N = 10, K = 2, a = 250, v = 45$, synchronization via bus;

t5 : $N = 6, K = 4, a = 800, v = 37$, hardware barrier synchronization.

The reconfiguration time of a single connection **tr** is in range 20..400, and section activation time $\mathbf{tv} = \mathbf{tf} + \mathbf{ts} + \mathbf{ta}$ is in range 5..300, where a is mean time interval between reconfigurations in the program.

The results are shown as a function of reconfiguration control efficiency $R = a/(\mathbf{tr} + \mathbf{tv})$ of the program executed on a given system, which is a ratio of the average time interval between reconfigurations of the same processor link in the program to the reconfiguration service time in the system.

We can see that the fast partitioning algorithm gives good speedup, with scheduled program execution times better or comparable than in the genetic approach. The results are better for programs executed in the system with bigger number of processor links. The explanation is that in such kind of system, the vertices of CAG have bigger number of incoming edges and thus, heuristic has a wider range of section for selection for clustering. When the reconfiguration control efficiency is low or the number of processor links is small, the genetic heuristic gives good results.

Another important performance factor is the algorithm time complexity. Our experiments show that execution of fast partitioning algorithm is one magnitude order faster than the genetic approach. The average algorithm execution times on the Dec Alpha 533Mhz work-station were over 4 min. for the genetic approach and about 30 sec. for the fast partitioning method proposed in this paper. Taking

into account result quality, it means that fast partitioning algorithm is a better solution in practical applications.

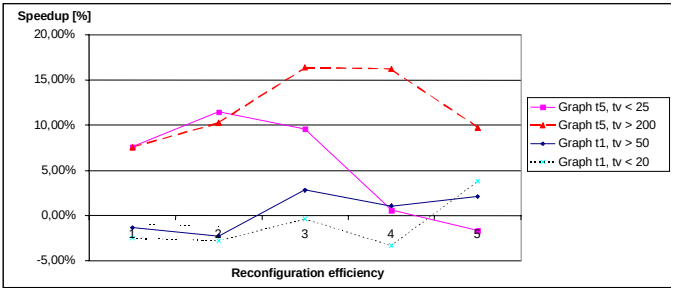

Fig. 9. The speedup of the fast partitioning algorithm against the genetic heuristics.

6 Summary

The paper presents task scheduling algorithm for the look-ahead reconfigurable multi-processor system. It is based on the ETF list scheduling heuristics and uses clustering approach to the program graph partitioning problem. The difference between presented algorithms has strong influence on the algorithm execution time. The presented algorithm is much faster than the one with genetic heuristics and gives better or comparable results, which has been proved by our experiments. The future works will focus on partitioning algorithm improvement and studies on others list scheduling strategies.

References

1. M. Tudruj, *Look-Ahead Dynamic Reconfiguration of Link Connections in Multi-Processor Architectures*, Parallel Computing '95, Gent, Sept. 1995, pp. 539-546.
2. J.-J. Hwang, Y.-Ch. Chow, F. D. Angers, Ch.-Y. Lee; *Scheduling Precedence Graphs in Systems with Interprocessor Communication Times*, Siam J. Comput., Vol. 18, No. 2, pp. 244-257, April 1989.
3. E. Laskowski, M. Tudruj, *A Testbed for Parallel Program Execution with Dynamic Look-Ahead Inter-Processor Connections*, Proc. of the 3rd Int. Conf. on Parallel Processing and Applied Mathematics PPAM '99, Sept. 1999, Kazimierz Dolny, pp. 427-436.
4. T. Kalinowski, *Program Execution Control in Dynamically Reconfigurable Multi-Processor Systems*, PhD Thesis, Institute of Computer Science PAS, Warsaw 1997.
5. El-Rewini H., Lewis T. G., Ali H. H. *Task Scheduling in Parallel and Distributed Systems*. Prentice Hall 1994