Embedded Parallel Systems Based on Dynamic Look-Ahead Reconfiguration in Redundant Communication Resources

Eryk Laskowski[1], Marek Tudruj[1,2]
[1] *Institute of Computer Science of the Polish Academy of Sciences*
01-237 Warsaw, Ordona 21, Poland
[2] *Polish-Japanese Institute of Information Technology*
02-801 Warsaw, Koszykowa 86, Poland
{laskowsk,tudruj}@ipipan.waw.pl

Abstract

A special parallel system architecture based on the look-ahead dynamic reconfiguration of inter-processor connections for execution of dedicated programs is discussed in the paper. The architecture assumes connection reconfiguration in multiple crossbar switches. Programs are structured into sections that use temporarily fixed inter-processor connections for communication. While some inter-processor connections in crossbar switches are used for execution of current sections, the connections necessary for execution of further sections are in advance configured some spare connection switches. Automatic program structuring for the assumed system architecture is proposed based on the analysis of parallel program graphs. The structuring algorithm finds the partition into sections that minimizes the execution time of a program executed with the look-ahead created connections. The program execution time is evaluated by simulated program graph execution with reconfiguration control modeled as an extension of the program graph.

1. Introduction

Parallel architectures based on run-time dynamic inter-processor connection reconfiguration enable adjusting a multi-processor system structure to program needs. This type of system structure can be optimized to reduce inter-processor connection reconfiguration time. This factor becomes more and more important in parallel systems based on user-level communication [7].

An environment, proposed in the paper, assumes new paradigm of point-to-point inter-processor connections setting, which neutralizes link connection reconfiguration time overheads by a special method applied at the level of system architecture and at the level of program execution control. This special method is called the look-ahead dynamic link connection reconfigurability [2]. Special architectural solutions for the look-ahead reconfigurability consist in increasing the number of hardware resources used for link connection setting (multiple crossbar switches) and using them interchangeably in parallel for program execution and run-time look-ahead reconfiguration.

Our approach can be used for synthesis of embedded parallel systems specified by interacting processes, which have been mapped on an architecture consisting of several processors connected by some communication resources. In the environment assumed in this paper process interaction is defined not only in terms of the flow of data (communication and processor interconnections) but also it captures the flow of control in the form of conditional instructions.

The paper consists of three parts. In the first part, the look-ahead inter-processor connection reconfiguration principles are presented. In the second part, the parallel program representations and features of the applied graph partitioning algorithm are discussed. In the last part, experimental results are shown and discussed.

2. The principle of the look-ahead connection reconfiguration

The look-ahead dynamic connection reconfiguration assumes anticipated inter-processor connection setting in communication resources provided in the system. An application program is partitioned into sections, which assume fixed direct inter-processor connections. Connections for next sections are prepared while current sections are executed. Before execution of a section, the

Proceedings of the 9th EUROMICRO Conference on Digital System Design (DSD'06)
0-7695-2609-8/06 $20.00 © 2006 IEEE

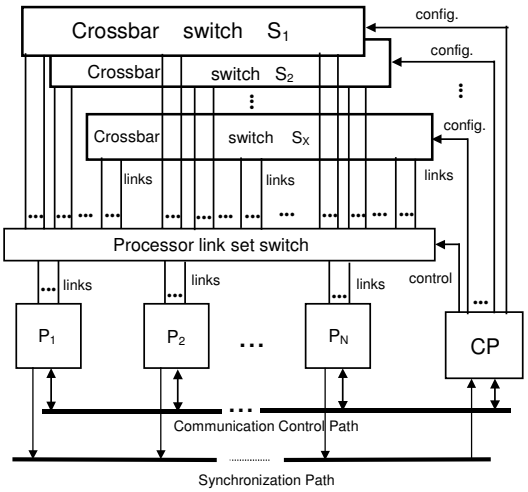


Fig. 1. Look-ahead reconfigurable system with multiple connection switches.

prepared connections are enabled for use in parallel and in a very short time. Thus, this method can provide inter-processor connections with almost no delay in the linear program execution time. In other words, it can provide time transparent control for dynamic link connection reconfiguration. The redundant resources can be link connection switches (crossbar switches, multistage connection networks), processor sets and processor link sets [2]. In the paper we investigate a system with multiple crossbar switches $S_1 \ldots S_X$ as redundant communication resources, Fig. 2. Connections between processors $P_1 \ldots P_N$ are set in crossbar switches in parallel with program execution and remain fixed during section execution. At sections boundaries, relevant processor's communication links are switched using Processor Link Set Switch to the crossbar switch(es) where the connections had been prepared. This switch is controlled by the Control Processor *CP*. We assume the asynchronous processor-restrained strategy, where inter-section connection switching is controlled at the level of dynamically defined worker processor clusters.

3. Parallel program representation

In the paper we use the Branching Task Graph (BTG), a new representation of data and control flow in parallel programs. The BTG inherits from several parallel program representations capturing data and control dependencies, which have been proposed in the system synthesis area [4, 5]. The BTG makes it possible to extend parallel program optimization [3] for execution in look-ahead reconfigurable systems, which now is not limited to classical DAGs with fully static control.

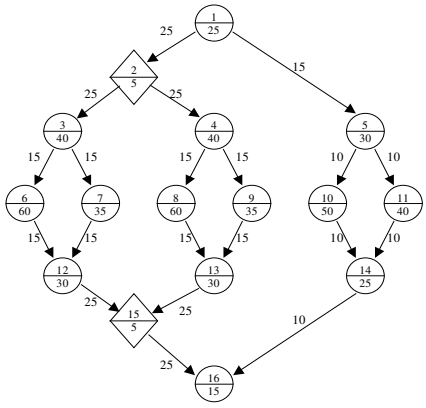


Fig. 2. An example of BTG.

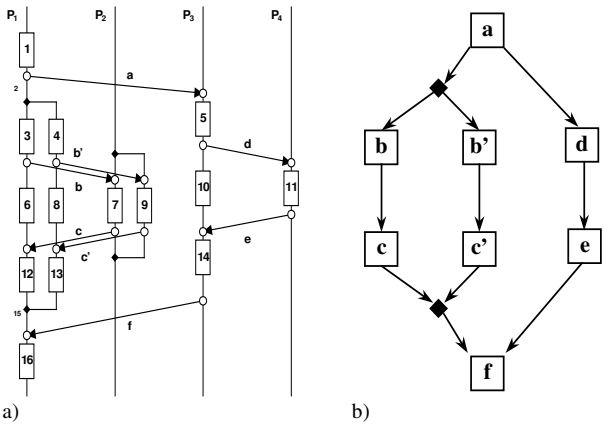
The BTG, is a weighted directed acyclic graph G*(V, V_C, E)*. The BTG consists of two kinds of nodes. Each node $n_i \in V$ represents a task, executed according to the *macro-dataflow* model. The weight of a node represents the task execution time. Each node $c_k \in V_C$, represents a conditional branch, in which the control flow of a program forks according to the value of a control statement, computed in the node. A directed edge $e_{ij} \in E$ represents communication that corresponds to data/ control dependencies among nodes n_i and n_j. The weight of an edge is the communication cost. Alternative paths starting from a conditional node meet in a conditional merge node $c_m \in V_C$. The graph is assumed to be static and deterministic. The probability distribution between control paths in each conditional node is known prior to the program execution.

A program with specified schedule is expressed in terms of an Extended Assigned Program Graph (XAPG), Fig 3a. XAPG assumes the synchronous communication model (CSP-like). There are two kinds of nodes in an XAPG: code nodes (which correspond to tasks in BTG, shown as rectangles in Fig. 3a) and communication nodes (circles in Fig. 3a).

The Extended Communication Activation Graph (XCAG) represents a program graph partition into sections. This graph is composed of nodes, which correspond to inter-processor communication edges of the XAPG program graph, and of edges, which correspond to activation paths between communication edges of the XAPG, Fig. 3b. Program sections are defined by identification of such subgraphs in the XCAG that validity conditions hold (see [3] for details).

4. Program structuring algorithms

Program structuring algorithms, presented in the paper, use a two-phase approach to obtain the schedule and partition of a program graph.



a) b)

Fig. 3. a) A schedule of a BTG from Fig. 2 in the form of XAPG, b) Extended Communication Activation Graph for the XAPG.

In the first phase, a list scheduling algorithm is applied to obtain a program schedule with reduced number of communications and minimized program execution time. In the second phase, scheduled program graph is partitioned into sections.

The scheduling algorithm is based on the ETF (Earliest Task First) heuristics [1]. Our algorithm differs from the original version of ETF in communication model assumed: instead of fixed inter-processor network topology, we investigate system with look-ahead dynamically created connections. We take into account a limited number of links and links contention. We have also introduced the new methodology of conditional branch scheduling. It includes *detection of mutually-exclusive paths* in the program graph [5]. During scheduling of a program graph, tasks and communications belonging to mutually exclusive conditional branch paths can share the same resource. It is accomplished by assigning them to the same time slot on a target system resource. To improve the quality of schedule, the structuring heuristics utilizes also the scheduling of *most-often-used paths* based on branch probabilities [6].

The goal of the graph partitioning algorithm is to find program graph partition into sections and to assign a crossbar switch to each section. The heuristics also finds the minimal number of switches, which allows program execution without reconfiguration time overheads. At the beginning of the algorithm, an initial partition consists of sections, each built of a single communication, which are assigned to the same crossbar switch. In each step, a vertex of XCAG is selected and then the algorithm tries to include this vertex to a union of existing sections determined by incoming edges of the current vertex. The union, which gives the shortest program execution time is selected. The vertices can be visited many times. The algorithm stops when all vertices have been visited and there hasn't been any program execution time improvement in a number of steps.

Estimation of the program execution time is based on simulated execution of the partitioned graph in a look-ahead reconfigurable system. For this purpose, a XAPG graph with a valid partition is extended by subgraphs, which model the look-ahead reconfiguration control. The functioning of the Communication Control Path, Synchronization Path and the Control Processor are modeled as subgraphs executed on additional virtual processors.

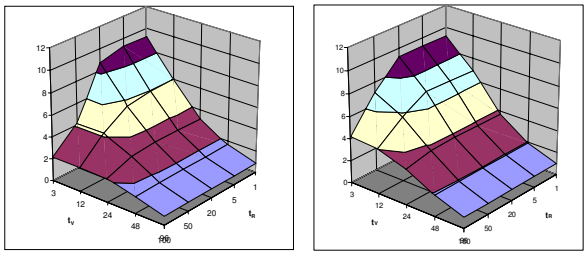
5. Experimental results

The described algorithm has been used for evaluation and comparative studies of execution efficiency of programs for different program execution control strategies and system parameters. The following parameters were used to characterize an application program in the experiments that were performed:

$\mathbf{t_R}$ - reconfiguration time of a single connection,
$\mathbf{t_V}$ - section activation time overhead,
a - average time between connection reconfigurations,
$\mathbf{R = a / (t_R + t_V)}$ – reconfiguration control efficiency.

Experiments with execution of exemplary program graphs have shown that the look-ahead connection reconfiguration behaves better (large program execution speedup with the look-ahead reconfiguration versus execution with the on-request reconfiguration) for systems in which the reconfiguration efficiency is poor (low values of the parameter **R**). This confirms the suitability of the look-ahead reconfiguration for fine-grain parallel systems. For systems with sufficiently fast connection reconfiguration control, the on-request reconfiguration can be sufficient (introduces lower reconfiguration control overhead).

We have examined program execution speedup versus parameters of reconfiguration control ($\mathbf{t_R}$ and $\mathbf{t_V}$), the number of crossbar switches, the number of processor links, for the parallel program of Strassen



a) 12 processors, 4 links, 2 x-bar b) 12 processors, 4 links, 4 x-bar

Fig. 4. Speedup for Strassen algorithm in the look-ahead environment.

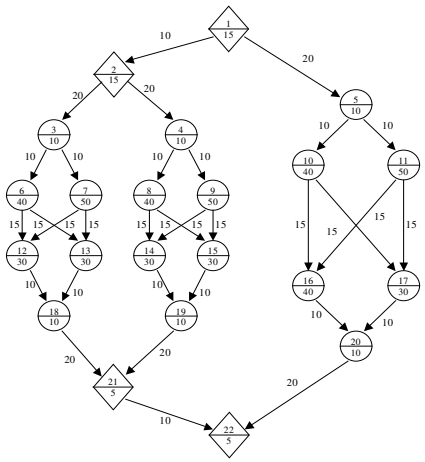


Fig. 5. The BTG graph of an exemplary parallel program.

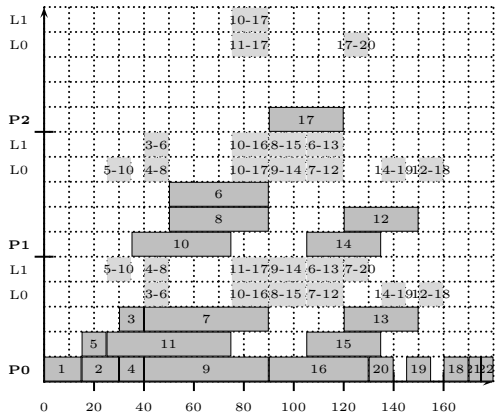


Fig. 6. Gantt chart of a schedule of an exemplary BTG from Fig. 5.

matrix multiplication (two recursion levels), Fig 4. With an increase in the number of crossbars, the high speedup area increases and is much larger than for the on-request reconfiguration. So, multiple crossbar switches used with the look-ahead control strongly reduce reconfiguration time overheads. The larger is the number of processor links, the look-ahead method is prevailing vs. on-request for a wider range of reconfiguration and activation time parameters.

We have compared schedules for programs with conditional branches obtained with our proposed algorithms with schedules based on the standard ETF heuristics, using an exemplary graph, Fig. 5. Some comparison results of program structuring by both methods are shown in the table below (see Fig. 6 for the Gantt chart of the schedule of the exemplary BTG):

algorithm	adapted ETF	proposed
program representation	DAG	BTG
number of communications	14	13
number of processors used	3	3
schedule length	245	180
reduction	–	26.5%

Compared with the standard ETF list scheduling algorithm, our structuring algorithm allows for more fine-grain analysis of parallel program graph and thus leads to better (shorter) program execution times. The analysis of experimental results for wider selection of program graphs is foreseen in the nearest future.

6. Conclusions

A new kind of the look-ahead reconfigurable multiprocessor embedded system together with the necessary program structuring algorithms have been presented in the paper. The new algorithm works on graph representation of programs, which captures data and control dependencies between tasks. It enables task scheduling of programs which include conditional branches in application programs. The presented algorithm is based on improved list scheduling and iterative section clustering heuristics, which have given better results than the one with greedy vertex selection heuristics and have the same time complexity. The assumed program representation enables applying new program optimization techniques based on detection of *mutually-exclusive paths* in programs and scheduling of the *most-often-used paths* based on branch paths taking probabilities.

References

[1] J.-J. Hwang, Y.-C. Chow, F. Angers, C.-Y. Lee; *Scheduling Precedence Graphs in Systems with Interprocessor Communication Times*, Siam J. Comput., Vol. 18, No. 2, 1989.

[2] M. Tudruj, *Look-Ahead Dynamic Reconfiguration of Link Connections in Multi-Processor Architectures*, Parallel Computing '95, Gent, Sept. 1995, pp. 539-546.

[3] E. Laskowski *Program Structuring Algorithms for Dynamically Reconfigurable Parallel Systems Based on Redundant Connection Switches*, Proc. of the 3rd Int. Sym. on Parallel and Distributed Computing, July 2004, Cork, Ireland.

[4] D. Wu, B. Al-Hashimi, P. Eles *Scheduling and Mapping of Conditional Task Graph for the Synthesis of Low Power Embedded Systems* IEEE Procs. – Computers and Digital Techniques, Vol. 150, Issue 5, Sept. 2003, pp. 303-312.

[5] Y. Xie, W. Wolf *Allocation and scheduling of conditional task graph in hardware/software co-synthesis.* DATE 2001.

[6] C. Murphy, X. Wang, *Most Often Used Path Scheduling Algorith,* Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2001), July 22-25, 2001, Orlando, Florida, USA. Vol. XII, pp.289-295.

[7] J.Chen, W. Watson, *User-level communication on Alpha Linux Systems* 2000 Intl. Symp. On Parallel Architectures, Algorithms and Networks, ISPAN, IEEE CS Press.