International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463
Vol. 3 Issue 4, April-2014, pp: (500-504), Impact Factor: 1.252, Available online at: www.erpublications.com.

Techniques for Custom Instruction Generation: A Survey

Parvathy Asokan[1], Kavitha V.[2]

[1] M. Tech., CMRIT, Bangalore, India
[2] Ph. D. Scholar, Jain University, India

Abstract: Efficiency and flexibility are critical in the design of embedded system. The Custom Instruction (CI) extraction is an important phase for implementing an application on an extensible processor. In this paper, we are discussing two techniques to extract the CIs for Reconfigurable Functional Unit of Extensible Processors: Temporal Partitioning Technique and Dynamic Critical Path (DCP) Algorithm. From the analysis, we came to a conclusion that the Dynamic Critical Path Algorithm provides high performance if used for Multiple processors on a chip.

Keywords: Extensible Processors, Horizontal Traversing Temporal Partitioning (HTTP), Vertical Traversing Temporal Partitioning (VTTP).

Introduction

General Purpose processors (GPP) are designed for general purpose computers. The most important concern of GPP is the computation speed. GPPs can be used in embedded systems because of their features such as flexibility, low cost design, programmability, tools availability, less time to market etc. The drawbacks of GPPs are their inefficiency for high performance computing. In order to accelerate the performance of GPP, an application specific instruction set extension can be added to the basic processor. Such a processor is called Application Specific Instruction Set Processors (ASIPs). In ASIP, the critical portions of the application can be executed on Custom Function Unit (CFU).Reconfigurable Instruction Set processors (RISP) consists of a microprocessor core that has been extended with the reconfigurable logic. It is similar to ASIP but instead of CFU, it contains Reconfigurable Functional Unit (RFU). Each task can be represented by a Data Flow Graph (DFG). The nodes of the DFGs are the critical instructions of the application and the edges represent the instruction dependencies. The critical path (CP) of a task graph is a set of nodes and edges, forming a path from an entry node to an exit node, of which the sum of the computation costs and communication costs is the maximum. Custom Instruction is a sequence of instructions that are extracted from hot basic blocks (HBBs). HBBs are basic blocks which are executed many number of times. A basic block is a sequence of instructions that are terminated by a control instruction. In this paper we are discussing two techniques to generate the Custom Instructions (CIs) for the RFU of extensible Processors- Temporal Partitioning technique and Dynamic Critical Path Algorithm (DCP).

Related Work

The Authors in [3] presented an adaptive dynamic extensible processor, AMBER. AMBER uses a coarse grain reconfigurable function unit with fixed resources. For temporal partitioning, several algorithms exist. Karthikeya et al. [4] proposed algorithms for large designs on area constrained reconfigurable hardware. SPARCS [5] is an integrated partitioning and synthesis framework, which has a temporal partitioning tool to temporally divide and schedule the DFG on a reconfigurable system. A Method to automatically detect recurring operation patterns to obtain custom instructions is presented in [6]. Instruction set selection refers to the problem of defining a custom instruction set that will result in the most efficient processing for a given application or domain. The work in [7] presents a technique for generating multicycle application- specific instructions for DSP applications.

Temporal Partitioning Technique

Integrated Framework is a CI generation approach that performs an integrated temporal partitioning and mapping process to generate mappable CIs. The flow diagram for Integrated Framework is shown in fig.1.

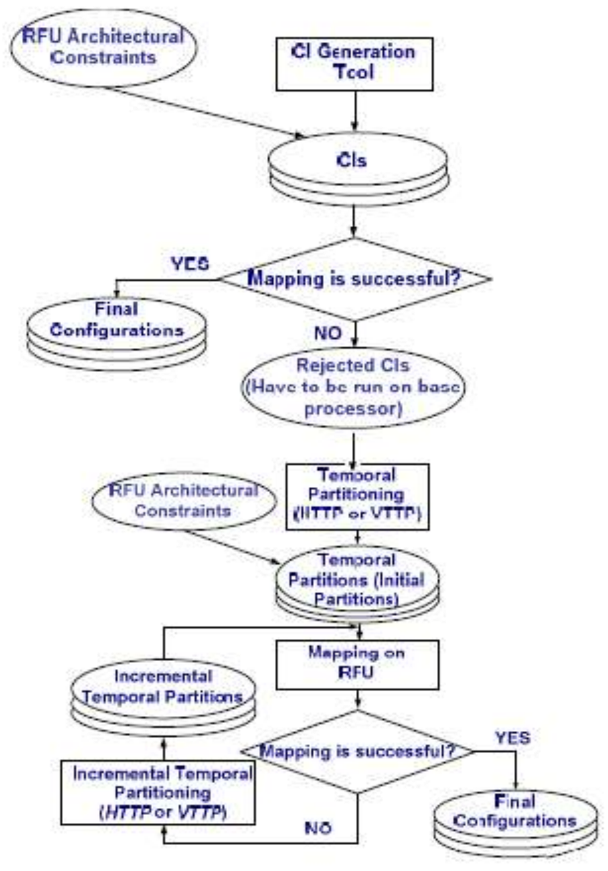
RFU Architectural Constraints
CI Generation Tool
CIs
YES
Mapping is successful?
Final Configurations
NO
Rejected CIs (Have to be run on base processor)
Temporal Partitioning (HTTP or VTTP)
RFU Architectural Constraints
Temporal Partitions (Initial Partitions)
Mapping on RFU
Incremental Temporal Partitions
Mapping is successful?
YES
Incremental Temporal Partitioning (HTTP or VTTP)
NO
Final Configurations

Fig.1: Design flow for Integrated Framework

In this design flow [1], the rejected CIs are partitioned to appropriate CIs with the capability of being mapped on the RFU. The design flow consists of the following steps:

- In the first stage, RFU primary constraints are considered to generate initial CIs by using CI generation tool.
- Then for each CI, mapping is done
- If they are successfully mapped, then the generated CIs are accepted and finalized.
- If not, an incremental temporal partitioning algorithm modifies the CI by moving some of the nodes .
- This is repeated until all the CIs are successfully mapped on RFU.

In the Integrated Framework, there are two algorithms for temporal partitioning- Horizontal Traversing Temporal Partitioning (HTTP) and Vertical Traversing Temporal Partitioning (VTTP). The first temporal partitioning algorithm in the Integrated Framework is HTTP. This algorithm traverses DFG nodes horizontally according to the As Soon As Possible (ASAP) level of nodes and adds them to the current partition while the architectural constraints are satisfied. The ASAP level of nodes represents their order to execute according to their dependencies. This algorithm increases parallelism for instruction execution. As a result, the intermediate data size is increased. The intermediate data size affects the data transferring rate and configuration memory size. The HTTP algorithm and its incremental version is shown in fig.2.

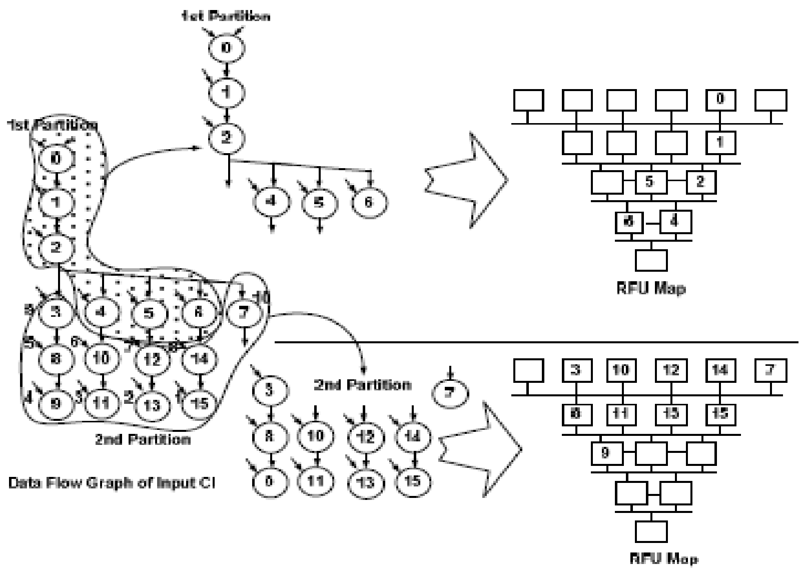

Fig.2. HTTP algorithm and its incremental version [1]

Incremental HTTP algorithm[1] selects a node and moves it to the next partition. The best choice for moving nodes is the nodes with the highest ASAP level. In fig. 2, the nodes 15, 13, 11,9,14,12,10,8,3 and 7 are selected in order and moved to the next partition. Another temporal partitioning algorithm is VTTP [1]. This algorithm traverses DFG nodes vertically. It creates partitions with longer critical paths and hence it reduces the intermediate data size. The VTTP algorithm and its incremental version is shown below.

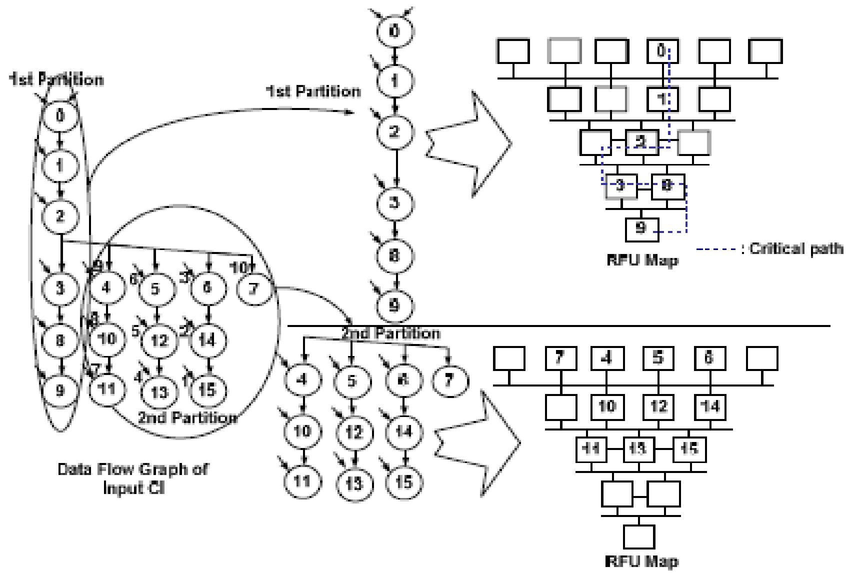

Fig. 3: VTTP algorithm and its incremental version [1]

The incremental VTTP algorithm extracts long paths during the traversing of DFG nodes. Firstly, nodes with highest ASAP level is selected and moved to the next partition. Then for modifying the same partition, the nodes are selected from the path where the previously moved node has been located. In fig.3. the nodes 15, 14, 6,13,12,5,11,10,4 and 7 are selected in order and moved to the next partition during incremental VTTP.
Mapping is defined as the placement of the DFG nodes on a RFU. The position of the DFG nodes on the RFU depends on the connection length and area. Minimum connection length and area are preferred. CIs can be assigned to the RFU on the priority basis. Slack of each node determines the priority for partitioning. Zero slack means the node is on critical path, and

is scheduled with highest priority. The position for each node determines the row number in the RFU. If the node is on the critical path, then the last row of the RFU is selected. After determining the row number, an appropriate column number is selected. The column number is determined according to the minimum connection length criterion. The main advantage of Integrated Framework is that it reduces the number of rejected CIs by partitioning them to appropriate CIs which can be mapped on RFU.

DCP Algorithm

The DCP algorithm [2] firstly determines the node to be scheduled. Then select the appropriate processor for scheduling that node. The critical path (CP) of a task graph determines the partial schedule length. The nodes on the critical path are scheduled properly in time and space. As the scheduling process continues the CP changes dynamically. In order to distinguish the intermediate CP from the original one, the intermediate CP is called as Dynamic Critical path (DCP). For identifying the nodes on the DCP, two attributes are obtained for each node: the lower bound and the upper bound on the start time of a node. The start time of nodes on a processor are not fixed until scheduling completes. The absolute earliest start time [2] of a node ni in a processor J is denoted by AEST (ni, J) and it is defined as

$$max_{1\le k\le p}\left\{AEST\left(n_{i_k}, PE\left(n_{i_k}\right)\right)+w\left(n_{i_k}\right)+r\left(PE\left(n_{i_k}\right), J\right)c_{i_k i}\right\}$$

where n_i has p parent nodes and n_{i_k} is the kth parent node.
$AEST(n_i, J) = 0$ *if it is an entry node, and*
$r\left(PE\left(n_{i_k}\right), J\right) = 1$ *if* $PE\left(n_{i_k}\right) \neq J$ *and zero otherwise.*

The AEST values can be computed by traversing the task graph in a breadth-first manner beginning from the entry nodes. All the AEST values of ni's parent nodes are available. The dynamic critical path length (DCPL) is defined as

$$max_i\left\{AEST\left(n_i, PE(n_i)\right)+w(n_i)\right\}$$

The absolute latest start time of a node ni in a processor J is denoted by ALST (ni, J) . It is defined as

$$min_{1\le m\le q}\left\{ALST\left(n_{i_m}, PE\left(n_{i_m}\right)\right)-r\left(PE\left(n_{i_m}\right), J\right)c_{ii_m}-w(n_i)\right\}$$

where n_i has q children nodes and n_{i_m} is the mth child node.
$ALST(n_i, J) = DCPL - w(n_i)$ *if it is an exit node, and*
$r\left(PE\left(n_{i_m}\right), J\right) = 1$ *if* $PE\left(n_{i_m}\right) \neq J$ *and zero otherwise.*

The value of ALST can also be computed by traversing the task graph in a breadth- first manner but in the reverse direction. The ALST values can be computed only after computing the DCPL. After assigning the AEST and ALST values, the nodes of the DCP can be identified. If the AEST and ALST values of a node are equal, then the node is on the DCP. After obtaining the DCP node, it is necessary to select the processor for scheduling that node. The selection of the processor can be done by using two rules.

RULE I. *A node n_i can be inserted into a processor J, which contains the sequence of nodes* $\left\{n_{J_1}, n_{J_2}, \ldots, n_{J_m}\right\}$, *if there exists some "k" such that*

$$min\left\{ALST(n_i, J)+w(n_i), ALST\left(N_{J_{k+1}}, J\right)\right\} - max\left\{AEST(n_i, J), AEST\left(n_{J_k}, J\right)+w\left(n_{J_k}\right)\right\} \ge w(n_i)$$

where, $k = 0, \ldots, m$, $ALST\left(n_{J_{m+1}}, J\right) = \infty$, *and;*
$AEST\left(n_{J_0}, J\right)+w\left(n_{J_0}\right) = 0$ *provided none of the nodes in* $\left\{n_{J_1}, n_{J_2}, \ldots, n_{J_k}\right\}$ *is a descendant node of n_i and none of the nodes in* $\left\{n_{J_{k+1}}, n_{J_2}, \ldots, n_{J_m}\right\}$ *is an ancestor node of n_i.*

This rule states that ni can be inserted into a processor if it has a sufficiently large idle time slot. After ni is inserted into the processor, the communication costs among the nodes in the processor are set to zero.

RULE II. *If a node n_i is inserted to the processor J, then*

$$AEST(n_i, J) = max\{AEST(n_i, J), AEST(n_{J_l}, J) + w(n_{J_l})\}$$

and

$$ALST(n_i, J) = min\{ALST(n_i, J), + w(n_i), ALST(n_{J_{l+1}}, J)\}$$

where l is a value of k satisfying Rule I.

The DCP algorithm schedules all the DCP nodes first. Then it updates the AEST and ALST values dynamically after each scheduling step so as to obtain the next DCP node. At last, it assigns the actual start times of each node to be the AEST of that node. The advantages of DCP algorithm are

- Assigns dynamic priorities to the nodes at each step based on DCP, so that the schedule length can be monotonically reduced.
- Selects suitable processor for a node based on the start time of the node's critical child node.
- Scheduled unimportant nodes to the processors already in use in order to waste processors.

Conclusion

Analysis has been done on two techniques to extract the Custom Instructions for the Reconfigurable Functional Unit of Extensible Processors. It is obtained that, the DCP algorithm works better while scheduling the task graphs onto multiprocessors as compared to Temporal Partitioning Technique.

Acknowledgement

I would like to express my heartfelt gratitude to my Guide **Mrs. Kavitha. V.**, Associate Prof,, Electronics And Communication Engineering, CMR Institute of Technology, for her timely advice on the technical support and regular assistance throughout the review work.

References

[1]. Farhad Mehdipour, Hamid Noori, Morteza Saheb Zamani, Kazuaki Murakami, Koji Inoue Mehdi, " Custom instruction Generation Using Temporal Partitioning Techniques for a Reconfigurable Functional Unit" Embedded and Ubiquitous Computing,, pp. 722-731, 2006.
[2]. Yu-Kwong Kwok, Ahmad, I. "Dynamic Critical- Path Scheduling: An effective Technique for Allocating Task Graphs to Multiprocessors" Parallel and Distributed Systems, vol.7, no. 5, pp. 506-521, May 1996.
[3]. Noori, H., Murakami, K., Inoue, K., General Overview of an Adaptive Dynamic Extensible Processor Architecture, Workshop on Introspective Architecture (WISA'2006), 2006.
[4]. Karthikeya, M., Gajjala, P., Dinesh, B., Temporal partitioning and scheduling data flow graphs for reconfigurable computer, IEEE Transactions on Computers, vol. 48, no. 6, 1999, pp.579–590.
[5]. Ouaiss, I., Govindarajan, S., Srinivasan, V., Kaul M., Vemuri R., An integrated partitioning and synthesis system for dynamically reconfigurable multi-FPGA architectures, In Proceedings of the Reconfigurable Architecture Workshop, 1998, pp. 31-36.
[6]. M. Arnold and H. Corporaal, "Automatic detection of recurring operation patterns," in Proc. Int. Symp. Hardware/Software Codesign, May 1999, pp. 22–26.
[7]. H. Choi, J.-S. Kim, C.-W. Yoon, I.-C. Park, S. H. Hwang, and C.-M. Kyung, "Synthesis of application specific instructions for embedded DSP software," IEEE Trans. Comput., vol. 48, pp. 603–614, June 1999.