Synthesis of Application Specific Multiprocessor Architectures for Process Networks

Basant Kumar Dwivedi, Anshul Kumar and M. Balakrishnan
Dept. of Computer Science and Engineering
Indian Institute of Technology Delhi, New Delhi, India
{basant, anshul, mbala}@cse.iitd.ernet.in

Abstract

In this paper, we address the problem of synthesis of application specific multiprocessor SoC architectures for process networks of streaming applications. An application is modeled as Kahn Process Network (KPN) which makes the parallelism present in the application explicit. The synthesis process involves selection of computation modules, memory modules, communication architecture and mapping of processes of KPN on compute units and channels on memory modules. Our solution minimizes hardware cost while taking into account the performance constraints. One of the salient features of our work is that it takes into account the additional overheads because of data communication conflicts. Our method uses average processing requirements of KPN to handle data dependent behavior of processes and cycles within the KPN. In contrast to others, we do not perform static scheduling, only mapping and synthesis is done.

1. Introduction

This paper addresses the problem of mapping a process network with data dependent behavior and soft real time constraints onto the heterogeneous multiprocessor System on Chip architectures. Many streaming applications which can be represented as Kahn Process Networks (KPNs) [7, 5] show data dependent behavior with soft real time constraints. For example, variable length decoding and motion compensation processes of MPEG2 decoder are very much dependent on type of images in the video sequence and there is also a requirement on number of frames to be decoded per second as a real time constraint. Moreover, instead of communicating at the beginning or at the end of computation, many of such applications have computation and communication interleaved. These applications can be mapped onto homogeneous multiprocessor architectures, but performance, power, area constraints etc. forces designer to customize the architecture heavily by exploiting computation and communication properties of the application. This makes the architecture heterogeneous.

Traditionally the problem of mapping the application onto the architecture has been viewed as a scheduling problem. There has been considerable work in the direction of scheduling task graphs with begin-end type of communication property and constant processing time requirements. This problem has been formulated as an MILP problem in [10, 2] with heterogeneous multiprocessors as the target. Dave et al. [4] also address the problem of scheduling of periodic task graphs with begin-end type of communication property. The problem of scheduling task graphs which have communication and computation interleaved, has been dealt in [3]. The target architecture is heterogeneous in nature in both the cases. All the above works suffer from the limitation that the processing requirement of a task has been assumed to be constant and independent of input.

Data dependent behavior rules out the possibility of static scheduling of the application because this is based on the assumption that the processing time of a task/process will not change for different inputs. Apart from this, previous works on scheduling of process networks [9, 1] also suggest that static scheduling of applications modeled as KPN cannot be done.

The main contribution of our work is an MILP based approach which can be used to map the KPN of streaming applications with data dependent behavior and interleaved computation and communication. The mapping takes place alongwith the synthesis of application specific multiprocessor SoC architecture for the given application. Static scheduling is not done at this stage. We assume that in the final implementation of the system, a dynamic scheduler will be present at the compute units. Since, static scheduling is not being done, exact instance of data communication conflicts are not known. Hence, our approach also estimates possible data communication conflicts and takes these into account in the mapping process which is the other salient feature of this work. Our approach also allows one to synthesize the interconnection architecture either along-

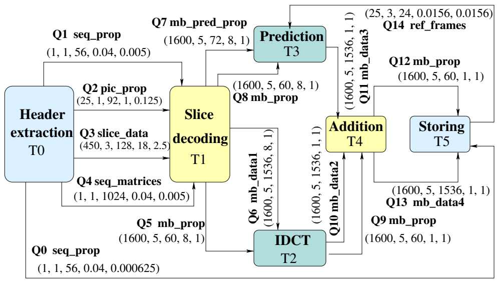

Figure 1. Application KPN

T_i, Q_j	i^{th} process, j^{th} channel
T_{qwr_j}, T_{qrd_j}	writer process of channel Q_j, reader process of channel Q_j
sz_j, len_j	size in bytes of the token of Q_j, length of Q_j
n_{ij}	number of tokens produced/consumed per iteration by T_i on/from Q_j
thr_j	throughput constraint in terms of number of tokens to be produced/consumed per second on Q_j
$iter_i$	iterations/second must be made by T_i

Table 1. Notations for application parameters

with the mapping process or separately in a post processing stage. Organization of rest of the paper is as follows. Section 2 provides details of application and architecture models. Section 3 provides details of how we perform data communication estimation. In Section 4 various MILP formulations have been presented. Section 5 gives details of experiments and Section 6 concludes.

2. Application Model and Architectural Component Library

As shown in Figure 1, we consider applications modeled as KPN. We assume that the processes are iterative in nature and perform computation as soon as required data is available at their inputs. This is a reasonable assumption for the streaming applications. The KPN model can have more than one channel between two processes. It can also have cycles. Unlike begin-end type of task graphs, here computation and communication are interleaved. Hence in our application model, an arc only means that there is some communication from one process to another during the course of computation. Various application parameters are given in table 1. As per this notation, in Figure 1, the tuple next to each arch is $(thr_j,\ len_j,\ sz_j,\ nrtoken_{qwr_jj},\ nrtoken_{qrd_jj})$.

We assume an architecture component library which contains a number of compute units, memory modules and interconnection components. A compute unit could be a non programmable unit like ASIC or a programmable unit such as a RISC or DSP processor. Alongwith each compute unit, there is a local memory. There could also be a number of shared memories which are connected to the compute units through interconnection network. Various architecture parameters are given in table 1.

CU_k	k^{th} compute unit
cpt_k, $freq_k$	cost of CU_k, frequency of CU_k
$ncycontxt_k$	number of cycles consumed during a context switch on CU_k
ncy_{ik}	number of cycles taken by the process T_i when mapped onto compute unit CU_k for one iteration without memory conflicts and context switch overheads
LM_k, SM_l	local memory alongwith CU_k, l^{th} shared memory
$cost\_base_k$	base cost of LM_k
$cost\_base_l$	base cost of SM_l
$bwmm_l$	bandwidth of SM_l in bytes/sec
SW_m	m^{th} switch
M_m, N_m	number of processor side ports, number of memory side ports
$swty_m$	type of switch SW_m. Value 1 denotes a bus and 0 denotes cross-bar switch
$cost\_sw_m$	overall cost of SW_m
$cost\_sw\_in_m$	cost of a link associated with SW_m
$bwsw_m$	bandwidth of SW_m

Table 2. Notations for architecture parameters

Interconnection components consist of a number of switches. In our formulation, buses are also called switches. The difference between a cross-bar switch and a bus is that bus provides low bandwidth low cost solution compared to a cross-bar switch. The main difference between single bus and multiple bus is that latter has higher bandwidth and a component attached to it will have as many connections to it as many buses in it. $cost\_sw\_in_m$ captures this higher link cost. We also associate bandwidth parameter $bwsw_m$ with every switch to take into account the bandwidth limitations of buses. Motivation of having cross-bar switches in the component library is based on the observation that in a process network, each process communicates with only some of the other processes. If the process network mapping onto the architecture is properly done, then a number of smaller switches can be employed to provide low cost high bandwidth solution.

3. Estimation of Data Communication Conflicts

If more than one channel are mapped onto the same memory module, it is possible that some request on one channel is in service when request on the other channel arrives. Inter-arrival period of requests on these channels depend on where readers and writers of these channels are mapped. Moreover because of mapping, shared memory request rate from each processor is not equal, memories are referenced non-uniformly and nature of shared memory access is that of burst. Except in [11] problem of heterogeneous non-uniform burst mode memory access has not been discussed elsewhere to the best of our knowledge. However, work in [11] doesn't take into account the slowing down of processors because of blocking delays due to contention.

In [6], we have proposed a method to estimate the mutual interference delays caused because of requests on two channels in the pre-processing stage for heterogeneous non-uniform burst mode memory access. If $rrcon_{j_1 j_2 k_1 k_2}$ is the delay seen by Q_{j_1} due to interference of Q_{j_2} when access on Q_{j_1} is being made from compute unit CU_{k_1} and on Q_{j_2} from CU_{k_2} for reads and rnp_{jk} is the number of time units during read inter-access on Q_j from CU_k, $rrcon_{j_1 j_2 k_1 k_2}$ can be given as:

$$rrcon_{j_2 j_1 k_2 k_1} = \frac{sz_{j_1} \times (sz_{j_1} + 1)}{2 \times (rnp_{j_1 k_1} + rrcon_{j_1 j_2 k_1 k_2})} \tag{1}$$

Similar equations are derived for other type of communication as well. Though, equation 1 leads to some over-estimation of data communication conflicts, it still allows one to estimate the communication conflicts and also keeps the number of variables introduced in MILP in limit.

4. Overall Synthesis

There are two aspects of the synthesis: selection of compute units and memories from the library and interconnection architecture synthesis. We have done formulation in such a manner that it allows to either perform the above two together or interconnection architecture synthesis can be performed in the post processing phase. This section describes these two aspects. Due to lack of space exact equations are omitted. Readers are referred to [6] for details.

4.1. MILP Formulation for Mapping

Decision Variables
There are basic binary variables which define the mapping of processes to compute units and channels to memories. Other variables are derived from these and correspond to connectivity of compute units to memories and amount of data communication conflicts.

Basic Mapping Constraints
These are the constraints which define mapping of process network and architecture instance.

1. A process can be mapped to only one compute unit.
2. ASIC can accommodate only one process.
3. A queue is mapped onto a local memory only when its reading and writing processes are mapped to the same compute unit.
4. A queue is mapped to either local memory of a compute unit or shared memory module.
5. A compute unit CU_k will communicate with a memory module SM_l when some reader or writer of a queue Q_j is mapped onto CU_k and queue itself is mapped onto SM_l.
6. A compute unit CU_k is utilized only if some process T_i is mapped onto CU_k.

Performance Constraints

1. Bandwidth of shared memory module SM_l should be larger than arrival rate.
2. A compute unit offers number of time units equal to its clock frequency (cycles per second). This must accommodate computation overheads of processes mapped, context switch overheads and waiting time due to data communication interferences.

Objective function
The objective is to minimize hardware cost. In the mapping stage, it essentially consists of cost of compute units used, local memory modules and shared memory modules.

4.2. ILP for Communication Architecture

Synthesis of communication architecture can either be done alongwith the previous stage (mapping) or as a post processing step when mapping is already known. Former leads to an overall minimum cost solution. Latter significantly simplifies the MILP of mapping stage, but this might lead to overall higher cost solution.

Decision Variables in this case define paths $CU_k - SW_m - SM_l$. These are further used to derive usage of a particular IN component.

Constraints

1. A switch of type bus cannot be used if it does not meet bandwidth requirement.
2. Number of compute units connected to a switch should not be greater than number of processor side ports of the switch. Similarly number of memory modules connected to a switch should not be greater than number of memory side ports of a switch.
3. Compute unit CU_k is connected to switch SW_m if there is at least one communication path from CU_k to some memory module SM_l and vice versa.
4. A switch is utilized only if some compute units and some memory module are connected to it.

Objective function in communication architecture synthesis is to minimize the total cost of switches and associated interconnections links.

Number of cycles taken by a process T_i per iteration on compute unit CU_k for $\forall k$

ncy_{0k}	ncy_{1k}	ncy_{2k}	ncy_{3k}	ncy_{4k}	ncy_{5k}
1043416	717331	75248	54564	23030	65134

Compute unit parameters

CU_k	$contxt_k$	TYP_t	cpt_t	$freq_k$	$cost\_base_k$
CU_0, CU_2	200	1	300000	100e+06	10
CU_1	200	2	700000	500e+06	10
CU_3	200	3	500000	400e+06	10

Interconnection network parameters

SW_m	$cost\_sw_m$	$cost\_sw\_in_m$	sp_m	smm_m	$bwsw_m$	$swty_m$
SW_0	20000	2400	4	4	3e+06	1
SW_1	40000	4800	4	4	6e+06	1
SW_4	320000	2400	4	4	24e+06	0
SW_5	160000	2400	4	3	12e+06	0
SW_6	120000	2400	3	2	9e+06	0

Table 3. Inputs

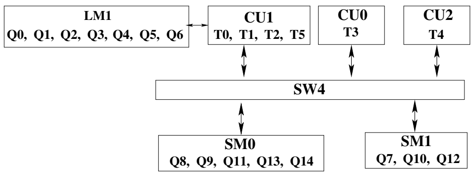

Figure 2. Synthesized architecture and mapping

5. Experiments and Results

We performed the experiments on MPEG2 Decoder obtained from mediabench [8]. This benchmark is ideal as it is streaming in nature, shows data dependent behavior for different video sequences, has multiple channels between two processes and also has a cycle. We implemented a library using which a C application can be modeled as KPN. We converted the sequential code of MPEG2 decoder into the KPN as shown in Figure 1. Functionalities of various processes are obvious from their names.

First part of the Table 3 shows process parameters of application KPN. These parameters were obtained by following the procedure discussed in [6]. Second part of this table shows compute unit attributes and the last part of it provides details of interconnection network (IN) parameters. In this experiment, presence of three memory units was assumed with bandwidth of 800000 and base cost of 5. The synthesized architecture and process network mapping is shown in Figure 2 after running the MILP. This example took less than 5 minutes on a workstation having Intel 2.20GHz XEON CPU alongwith 1GB RAM.

6. Conclusions

An MILP based approach for synthesis and mapping of process networks onto heterogeneous multiprocessor architecture has been presented. Our approach also provides an estimation of the data communication conflicts arising due to mapping of multiple channels of process network on the same shared memory module. Communication architecture synthesis can be done either alongwith mapping or in a post processing stage. Our MILP is extendible and optimizations such as synthesis of low power architectures can also be performed based on power consumption during each iteration of process and each transaction on channel. We have performed experiments on a real life example MPEG2 decoder from mediabench and results for one problem instance has been presented. Our approach can be effectively used to generate application specific multiprocessor architectures for applications modeled as KPN which show data dependent behavior. Our approach is not restricted to KPN in the FORM of DAG, but also allows cycles within it.

References

[1] T. Basten and J. Hoogerbrugge. Efficient Execution of Process Networks. In *Communicating Process Architectures*, 2001.

[2] A. Bender. MILP Based Task Mapping for Heterogeneous Multiprocessor Systems. In *Proc. EURO DAC*, 1996.

[3] J.-M. Chang and M. Pedram. Codex-dp: Co-design of Communicating Systems Using Dynamic Programming. *IEEE Trans. on CAD*, 19(7):732–744, July 2000.

[4] B. P. Dave, G. Lakshminarayana, and N. K. Jha. COSYN: Hardware Software Cosynthesis of Heterogeneous Distributed Embedded Systems. *IEEE Trans. on VLSI Systems*, 7(1):92–104, Mar. 1999.

[5] E. A. de Kock, G. Essink, W. J. M. Smits, P. van der Wolf, J. Y. Brunel, W. M. Kruijtzer, P. Lieverse, and K. A. Vissers. YAPI:Application Modeling for Signal Processing Systems. In *Proc. 37th Design Automation Conference (DAC'00)*, pages 402–405, June 2000.

[6] B. K. Dwivedi, A. Kumar, and M. Balakrishnan. Synthesis of Application Specific Multiprocessor Architectures for Process Networks . Technical report, Dept. of Computer Science & Engg., Indian Institute of Technology Delhi, 2003. http://www.cse.iitd.ernet.in/esproject/.

[7] G. Kahn. The Semantics of a Simple Language for Parallel Programming. In *Proc. IFIP Congress 74*. North Holland Publishing Co, 1974.

[8] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *International Symposium on Microarchitecture*, pages 330–335, 1997.

[9] T. M. Parks. *Bounded Scheduling of Process Networks*. PhD thesis, University of California, EECS Dept. Berkeley, 1995.

[10] S. Prakash and A. C. Parker. SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems. *J. Parallel & Distributed Computing*, pages 338–351, Dec. 1992.

[11] D. J. Sorin, J. L. Lemon, D. L. Eager, and M. K. Vernon. Analytic Evaluation of Shared-Memory Architectures. *IEEE Transaction on Parallel and Distributed Systems*, 14(2):166–180, Feb. 2003.