ISSN: 2229-6948 (ONLINE)
DOI: 10.21917/ijct.2010.0010

A HETEROGENEOUS MULTIPROCESSOR SYSTEM-ON-CHIP ARCHITECTURE INCORPORATING MEMORY ALLOCATION

T.Thillaikkarasi[1], A. Jagadeesan[2] and K.Duraiswamy[3]
Bannari Amman Institute of Technology, Tamil Nadu, India
Email: mails4thillai@gmail.com[1], jagadeesan_a@yahoo.co.in[2]
[3]*K.S.R.College of Technology, Tiruchengode, Tamil Nadu, India*
Email: ukdswamy@rediffmail.com

Abstract
This paper describes the development of a Multiprocessor System-on-Chip (MPSoC) with a novel interconnect architecture incorporating memory allocation. It addresses the problem of mapping a process network with data dependent behavior and soft real time constraints onto the heterogeneous multiprocessor System on Chip (SoC) architectures and focuses on a memory allocation step which is based on an integer linear programming model. An application is modeled as Kahn Process Network (KPN) which makes the parallelism present in the application explicit. 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. 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. It permits to obtain an optimal distributed shared memory architecture minimizing the global cost to access the shared data in the application, and the memory cost. Our approach allows automatic generation of an architecture-level specification of the application.

Keywords:
Application Specific Multiprocessors, Integer Linear Programming, Kahn Process Networks, System on Chip, Memory Allocation.

1. INTRODUCTION

In embedded systems the memory architecture can be chosen more or less freely, it is only constrained by the application requirements. Different choices can lead to solutions with a very different cost, which means that it is important to make the right choice. For this reason, the allocation of the memory blocs major steps in the SoC design flow. The goal of the memory allocation and assignment is to make use of the memory architecture freedom to minimize the cost related to background memory storage and transfers. Many applications in fields such as multi-media (audio and video) and image processing handle bulky and strongly dependent data. They consequently require the integration of a great number of memories of various types (local private, local distributed and on-chip global shared memory). Moreover, up to 70% of the chip area is dedicated to memory. Unfortunately, nowadays, there is not a complete and automatic method allowing designers to integrate all these memory types (particularly the shared memory) technological and market trends, including the ability to produce in the SoC from a high abstraction level. In this paper, we address the problem of synthesis of application specific multiprocessor SoC architectures for process networks of streaming applications. Many streaming applications which can be represented as Kahn Process Networks (KPNs) show data dependent behavior with soft real time constraints.

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 with heterogeneous multiprocessors as the target. All the works suffer from the limitation that the processing requirement of a task has been assumed to be constant and independent of input.

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 along with the synthesis of application specific multiprocessor SoC architecture for the given application. Static scheduling is not done at this stage. Our approach also allows one to synthesize the interconnection architecture either along 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 previous work and contribution. Section 3 provides details of application and architecture models. In Section 4 various MILP formulations have been presented. Section 5 gives details of memory allocation. Section 6 gives the analysis and Section 7 concludes.

2. PREVIOUS WORK & CONTRIBUTION

We have found a lot of works concerning memory integration and optimization in the field of single or multiprocessor system design. Some of them deal with memory allocation.

In this work we only deal with SoC. These are different from classic general purpose architectures because they target a specific application which makes the memory architecture and the communication network specific to the application and then simpler. For instance in most of these applications data regularity is quite trivial or non existing and thus no sophisticated data cache is required.

The contribution of our work is a full systematic approach allowing an optimal distributed shared memory allocation for application-specific SoCs, and automatic architecture-level application-code generation.

3. APPLICATION & ARCHITECTURE MODELS

As shown in Fig.1, we consider applications modeled as KPN.

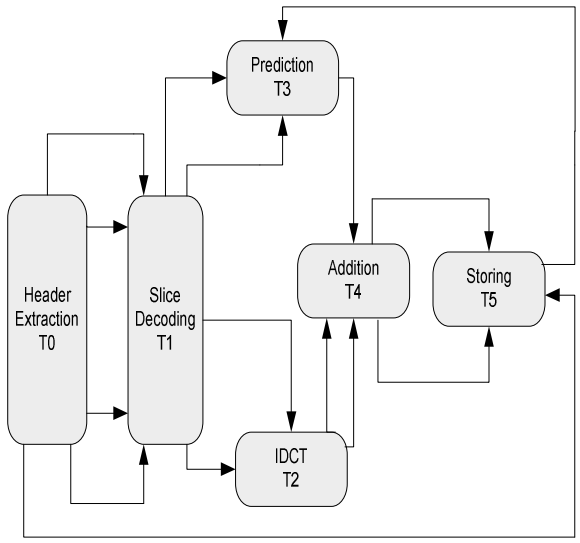

Fig.1. Application KPN

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.

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. Along with 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.

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. 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.

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.

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 along with 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 CUk – SWm – SMl. 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 CUk is connected to switch SWm if there is at least one communication path from CU_k to some memory module SMl 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.

5. MEMORY ALLOCATION STEP

Our memory allocation flow (Fig.2) takes as input a system level specification of the application (after processor allocation), a generic architecture model and libraries containing the estimated access time of each processor to memories and memory costs. This flow is mainly composed of three parts. The first consists in extracting parameters from the application code. The second carries out the memory allocation using an integer linear program (generated automatically). The third reads/writes primitives of the shared data in the application code (taking into account the memory allocation results), and generates an architecture-level description of the application. These three parts will be detailed in this section.

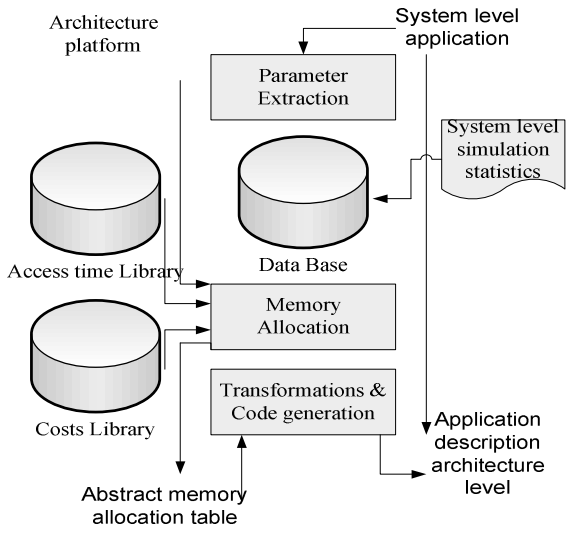

Fig.2. Memory allocation and code generation flow

5.1. PARAMETER EXTRACTION

This stage consists in extracting from the system level description of the application some information about the handled data, such as their names, sizes (types) and the use of the communication channels (for each communication channel connecting two processors, we determine the variables exchanged through this channel).

5.2. ALLOCATION

Using the parameters extracted in the previous step and the results of the system-level simulation of the application, we generate automatically an integer linear program. This program gives an exact solution for the memory blocs allocation minimizing the memory cost and the global time to access the shared data in the application as follows.

The objective function "F" consists in minimizing the total access time (reading and writing) of the processors accessing the shared data ((1), (2)) + Costs due to the memories sizes and to their integration in the system ((3)).

"F" has to be minimized subject to the following constraints:

The sum of data sizes assigned to the memory k has to be smaller or equal to the size of the memory k. This constraint allows us to compute the memory sizes (4).

If the memory k is not included in the architecture, its size must be equal to zero (5).

If the memory k is included in the architecture, its size must be bigger than zero (6).

Each variable must be assigned to one and only one memory (7).

- The variables X_{kj} and Y_k are binary; TM_k is integer.

Thus, we obtain:

Min F = $\sum_{i=1}^{p}\sum_{j=1}^{v} nb\_read(i,j)\sum_{k=1}^{m} T\_read(i,k)\,X_{kj}$ (1)

$\sum_{i=1}^{p}\sum_{j=1}^{v} nb\_write(i,j)\sum_{k=1}^{m} T\_write(i,k)\,X_{kj}$ (2)

$\sum_{k=1}^{n} CBM_k TM_k + \sum_{k=1}^{m} CUM_k Y_k$ (3)

Subject to:

$\sum_{i=1}^{p} Taille\_var(j)\,X_{kj} \le TM_k$ (4)

$TM_k \le AY_k$ k=1,….,m (5)

$TM_k \le AY_k - A + 1$ k=1,….,m (6)

$\sum_{k\in Sj}^{p} X_{kj} = 1$ j=1,…….v (7)

$X_{kj} \in \{0,1\} TM_k \in N$ j=1,……v et k=1……m (8)

$Y_k \in \{0,1\}$ and $TM_k \in N$ k=1……m (9)

P = number of processors; V = number of variables

- M = number of memories (<= P+1); TMk = size of the memory k

- Sj = set of indexes of memories associated with processors using the data j, plus the index of the shared memory

- Xjk = 1 if and only if the variable j is assigned to the memory k

- Yk = 1 if and only if the memory k is integrated to the architecture

- Nb_read (i, j) (resp. Nb_write (i, j)) = the number of processor i's read (resp. write) accesses to the variable j (obtained by a system level simulation)

- T_read (i, k) (resp. T_write (i, k)) = estimated read (resp. write) access time of the processor i to the memory k

- Taille_var (j) = size of the variable j

- CBMk (resp. CUMk) is the cost due to the memory k's size (resp. average cost of the integration of the memory k in the architecture).

Model complexity

For a problem instance we obtain:

- at least $[(P + 1).V]+ [P + 1] + [P + 1] = (P + 1) (V + 2)$ variables, With:

a = number of Xjk variables,

b = number of Yk variables,

c = number of TMk variables.

- NB_constraintes = (P + 1) + (P + 1) + (P + 1) + V constraintes

Note that in the computation of the number of variables we supposed that each one of the V shared data is used by all the processors, this is still very theoretical. In real applications that we know the NB_Variables tends generally to (P + 1) (V + 2)/2.

5.3. CODE TRANSFORMATION

This third module avoids the boring and error prone task of analyzing all the application description files in order to insert the shared memory module, and to carry out the necessary code modifications. Indeed, it rewrites the application description at the architecture level and adds a new module which is the shared memory and its controller. All the read and write operations on the shared data are changed by explicit read/write primitives on the shared memory.

6. ANALYSIS

This application was described at the functional level mainly in 4 interface files and 4 implementation files. Automatic refinement adds to the specification 4 files (2 interfaces and 2 implementations) corresponding to the memory body and to the memory's controller (200 lines at the functional level). The interfaces of the 4 processors were modified automatically in order to connect them to the global shared memory, and all the accesses to the data stocked in this memory were modified. Then, we obtained the application code at the architecture-level with a shared memory architecture in a complete automatic way.

The modeling of the memory allocation problem by an integer linear programming approach presents some major advantages as:

- it is an exact method, which contrary to the heuristic based methods, gives an optimal solution,

- it is a very generic model which allows the integration of all the memory types (local private, local distributed and global shared memories) in the architecture,

- it resolves two problems: allocation of the memory blocs, and the data assignation into these blocs,

- there are many available tools which permit the resolution of such a model.

Since some variables in our model are Boolean, the resolution step can be slow depending on the number of such variables. So, for the applications integrating lot of processors, we recommend the use of stochastic methods instead of the linear model.

Our future works will consist of the development of algorithms allowing the optimization of the variables placement in a given memory and the automatic interface generation. These algorithms will have to take into account the physical characteristics of the memories and the access modes.

7. CONCLUSIONS

An MILP based approach for synthesis and mapping of process networks onto heterogeneous multiprocessor architecture has been presented. We use an exact method to resolve the memory allocation problem for the fixed criteria (total access time to the shared data and the cost of the memory architecture). The proposed methodology permits a systematic generation of generic memory architecture for multiprocessor embedded SoC, from a high abstraction level distributed specification of the application. 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. 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] De Kock, E. A., Essink, G., Smits, W. J. M., vander Wolf, P., Brunel, J. Y., Kruijtzer, W. M., Lieverse, P. and Vissers, K. A (2000) 'YAPI:Application Modeling for Signal Processing Systems', In Proc. 37th Design Automation Conference (DAC'00), pp. 402–405.

[2] Basten, T. and Hoogerbrugge, J. (2001) 'Efficient Execution of Process Networks', In Communicating Process Architecture.

[3] Baghdadi, A. Lyonnard, D. Zergainoh, N-E. Jerraya, A.A. (2001) 'An Efficient Architecture Model for Systematic Design of Application-Specific Multiprocessor SoC', Proc. of DATE 2001.

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

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

[6] Singh, A. Chhabra, A. Gangwar, A. Dwivedi, B. K. Balakrishnan M.and. Kumar. A ,(2003)'SoC Synthesis With Automatic Interface Generation' In Proc. 16th International Conference on VLSI Design (VLSI-2003), New Delhi, India, pp. 585–590.

[7] Dwivedi, B. K. Kumar, A. and Balakrishnan, M.(2004), 'Synthesis of Application Specific Multiprocessor Architectures for Process Networks', In Proc. 17th International Conference on VLSI Design, Mumbai, India, pp. 780–783.

[8] Dwivedi, B. K. Kumar, A and Balakrishnan, M.,(2004), 'Automatic Synthesis of System on Chip Multiprocessor Architectures for Process Networks. In Proc. Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2004), Stockholm, Sweden, pp. 60–65.

[9] Dwivedi, B. K. Dhand, H. Balakrishnan M. and Kumar. A., (2005) 'RPNG: A Tool for Random Process Network Generation', In Proc. Asia and South Pacific International Conference in Embedded SoCs (ASPICES-2005), Bangalore, India, July.

[10] Basant K. Dwivedi, Arun Kejariwal, Balakrishnan, M. and Anshul Kumar. (2006) 'Rapid Resource-Constrained Hardware Performance Estimation', In Proc. International Workshop on Rapid System Prototyping (RSP06), Chania, Crete, Greece.