2011 IEEE Computer Society Annual Symposium on VLSI

Preprocessing-based Run-time Mapping of Applications on NoC-based MPSoCs

Samarth Kaushik, Amit Kumar Singh, Thambipillai Srikanthan
Centre for High Performance Embedded Systems, Nanyang Technological University, Singapore
{samarth2, amit0011, astsrikan}@ ntu.edu.sg

***Abstract*— Design-time strategies are suited only for mapping predefined set of applications and thus cannot predict dynamic behavior. This dynamism demands run-time mapping of application tasks to maintain a critical balance between performance and resource optimization. This paper proposes a run-time heuristic that intelligently distributes the application tasks among multiple processors taking communication overhead, computation load and resource utilization in consideration.**

***Keywords:* Multiprocessor System-on-Chip (MPSoC), Network-on-Chip (NoC), Mapping Algorithms.**

I. INTRODUCTION

System-on-Chip (SoC) design is experiencing a radical shift from uni-processor architecture to multi-processor architecture in order to adjust with the ever increasing demand for high performance. The rising complexity of real-life applications cannot be addressed by simply trying to make single-core processors run faster, instead it requires multiple processors, connected with a Network-on-Chip (NoC), which can cohesively communicate and provide increased concurrency [1].

The challenge is to map parallelized tasks of an application onto MPSoC platform, which entails a judicious mechanism of mapping these tasks on various processing elements (PEs), either at design-time or at run-time. Numerous design-time mapping techniques have been developed but they are limited to predefined set of applications and are unaware of run-time resource management [2], whereas run-time mapping techniques can be employed to large number of applications and incorporate run-time resource management. In [3], Holzenspies et al. propose a run-time strategy for mapping inherently parallel streaming applications on MPSoC. Singh et al. [4] describe a communication aware run-time mapping heuristic for MPSoC platforms accommodating multiple tasks on a single PE. The heuristic tries to minimize the communication overhead between two highly communicating tasks by mapping them on the same PE. However, existing heuristics does not attempt to balance the computation load on each PE utilized for mapping and also involves a restricted approach for minimizing communication overhead.

We present a run-time task mapping technique that reduces computation load variance and delineates substantial performance improvements along with efficient resource utilization.

II. PROPOSED ALGORITHM

Our technique performs pre-processing of the application graph before actual mapping is done in order to reduce the communication overhead and improve the load balancing on various platform PEs, taking available memory on PEs into consideration.

Application Model. An application is modeled as a set of communicating parallel processes represented as a task graph. The task graph is denoted as a directed graph $ATG = (T, E)$, where T is a set of application tasks and E is the set of all edges in the application, connecting the tasks and representing their communication as shown in Figure 1. A task $t_i \in T$ is represented as (t_{id}, t_{comp}), where t_{id} is the task identifier and t_{comp} is the task computation load in cycles. An edge $e_i \in E$ connecting the two tasks contains zcommunication information (t_{comm}) between the tasks. t_{comm} represents the number of cycles taken for transferring a single token when full channel bandwidth is available.

Platform Model. The MPSoC architecture is a graph $AG = (P, C)$, where P is the set of PEs identified by its identifier p_{id} and C represents the on chip communication channels for interconnecting the PEs. The PEs are connected in 4×4 mesh topology by a NoC. Among the available PEs, one is used as Manager Processor that is responsible for managing task operations and resources usage, including run-time management of task loads.

Mapping. Task mapping is represented by function *mpg*: $t_i \in T \rightarrow p_i \in P$, which maps each task of the application on the platform PEs.

A. Pre-Processing

The technique tries to minimize communication latencies among various tasks of the application while simultaneously trying to balance the processing load on various PEs. The scheme starts by targeting the communication intensive edges in the application and attempts to merge these highly communicating tasks on the same PE. The merging operation takes place only if memory constraint of the involved PE is satisfied, i.e., the PE must have sufficient memory to accommodate both the tasks and shared memory for their local communication. The shared memory is required by communication data on the edge of the connecting tasks. The proposed strategy forms a global approach as complete application graph is seen in entirety for removing communication bottlenecks, in contrast to mapping technique in [4] where merging of communicating tasks takes place during execution. The main purpose of

978-0-7695-4447-2/11 $26.00 © 2011 IEEE
DOI 10.1109/ISVLSI.2011.43

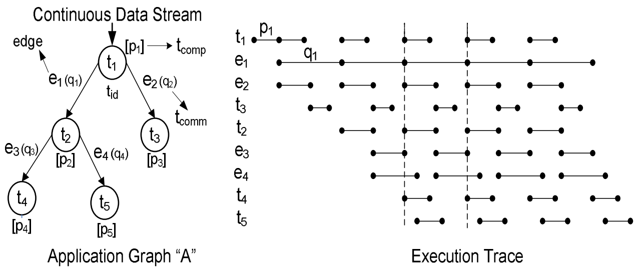

Figure 1: Execution Trace of Application Graph A

pre-processing is to remove any bottleneck that may arise due to overhead of transferring data among communicating tasks which can be understood by examining the execution trace of an application graph. In Figure 1, execution trace of application graph A is shown, where p_i represents the trace for computation time of each task $t_i \in T_i$ and q_i represents the trace for communication time of each edge $e_i \in E_i$. This representation expresses the available parallelism that can be exploited to achieve high performance by removing the bottlenecks, for example, edge e_1 appears to be the main bottleneck. If the communicating tasks of edge e_1 are merged together on a single PE, then e_1 no longer remains the active bottleneck of the system and the whole execution trace will shrink leading to faster execution. The same process is repeated with next highest bottlenecked edge till a processor becomes the bottleneck.

During pre-processing, once the processor becomes the bottleneck, resource optimization is carried out by merging the tasks with minimum computation load such that after merging, the communication overhead or computation load does not overshoot the computation bottleneck determined in the earlier step. This step not only enhances resource utilization but also tries to balance the computation load among several PEs by bringing the computation load of each PE as close as possible to computation bottleneck.

B. Mapping

The optimized application graph thus obtained by the pre-processing can be mapped using a run-time mapping heuristic. Nearest Neighbor heuristic proposed in [5] has been employed where communicating tasks are mapped on the neighboring PEs.

III. EXPERIMENTS AND RESULTS

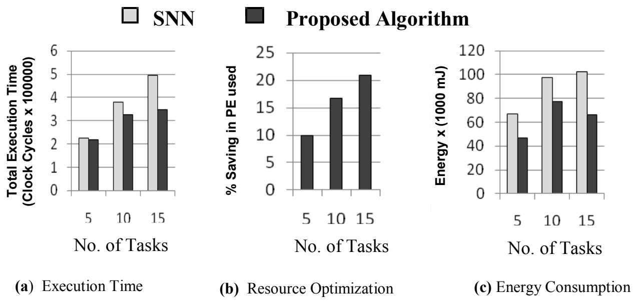

Figure 2: Computation Load Variance for Apps with 5, 10 & 15 Tasks

A Model-Sim simulator same as used in [4] has been adopted for performing experiments in co-simulation.

We have evaluated scenarios with random, pipeline & tree like streaming applications having 5, 10 and 15 tasks. Platform PEs are homogeneous processors.

We have measured execution time, resource utilization, energy consumption and computation load variance for mapping applications onto the platform. The total execution time includes time for pre-processing, mapping, configuration, processing and communication. The result obtained with our approach is compared with the Smart Nearest Neighbor (SNN) heuristic proposed in [4]. Figure 2 (a) compares the overall execution time. Resource utilization is measured as the percentage usage of PEs in the mapping. Figure 2(b) shows saving in resource usage for applications with various number of tasks. Energy consumption is measured as sum of communication and computation energy [6]. Figure 2(c) compares average energy consumption. Computation Load Variance represents the probability of computation load on each PE. Figure 3 shows the average computation load variance for different applications with 10 and 15 tasks.

IV. CONCLUSIONS

This paper describes a new mapping strategy, where placement for a task is found to balance the computation load on different PEs and to reduce communication overhead in the multi-tasking MPSoC platform. The improvements are clearly enunciated in the experiments and results section.

REFERENCES

[1] A. Jerraya et al., Guest editors' introduction: multiprocessor systems-on-chips, Computer 38 (7) (2005) 36–40.

[2] L.-Y. Lin et al., Communication-driven task binding for multiprocessor with latency insensitive network-on-chip, in: Proceedings of ASP-DAC, 2005, pp. 39–44.

[3] P. K. F. H¨olzenspies et al., Run-time spatial mapping of streaming applications to a heterogeneous multiprocessor system-on-chip (mpsoc), in: Proceedings of DATE, 2008, pp. 212–217.

[4] A. K. Singh et al.,"Run-time mapping of multiple communicating tasks on MPSoC platforms", International Conference on Computational Science, ICCS 2010.

[5] Carvalho, E.; Moraes, F. Congestion-aware task mapping in heterogeneous MPSoCs. System-on-Chip (SoC), 2008.

[6] A. K. Singh et al., "Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms," Journal of Systems Architecture,vol. 56, no. 7, 2010.

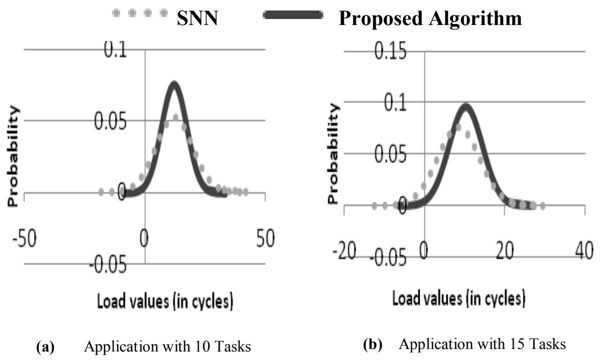

Figure 3: Computation Load Variance for App with 10 & 15 Tasks