Longest parallel path mapping algorithm for heterogeneous MPSoCs

Nikolina Frid, Vlado Sruk
University of Zagreb, Faculty of Electrical Engineering and Computing
Zagreb, Croatia
nikolina.frid@fer.hr, vlado.sruk@fer.hr

1. Introduction

Fast and efficient design space exploration for multi – processor System – on – Chip (MPSoC) is the key for reduction of time – to – market in embedded system design. Scheduling and mapping software to hardware on heterogeneous platforms is NP – complete problem, and finding near optimal solution in reasonable amount of time is challenging.

Traditionally this problem is solved using Linear Programming model [Lin et al. 2011] but high complexity of this approach and extremely high computational requirements significantly increase design time. There have been several attempts to solve this problem, including greedy deterministic [Gajski, Vivekanandarakaj] and evolutionary non-deterministic approaches [Palesi et al. 2002]. Greedy approach is the most common and delivers the final mapping solution much faster than evolutionary approach but always gives a suboptimal solution.

2. Automatic Mapping Tool

The authors propose the following heuristics based mapping algorithm: application is modelled using a parallel model of computation and represented as a Directed Acyclic Graph (DAG), the vertices being the tasks and the edges representing communication between tasks. Vertex weight is the number of operations executed in the process and the weight of the edge is the amount of data transacted both as reported by a profiler. The MPSoC is also modelled as a directed graph, each vertex representing a processing element – PE (weight is the computation speed) and each edge representing connection between PEs (weight is the throughput).

This algorithm’s objective is to minimize the overall execution time of the application. The mapping process, as shown on Figure 1, consists of four stages. In the first stage, as long as there are free PEs (sorted in descending order by computation speed), using a modified version of Dijkstra algorithm, the algorithm identifies a critical path and maps all tasks in the path to the same PE. Thus, communication costs between tasks on the same path are minimized.

Next, a preliminary schedule is made for mapped tasks, which enables the search for free gaps in PE task list where unmapped tasks will be mapped in stage 3. Mapping remaining tasks starts with PEs sorted ascending by computation speed. For each unmapped task a time frame is calculated - it begins when all of task’s mapped predecessors have finished and ends before the first successor has to start. Starting with the slowest PE, mapper checks if it is free during time frame and fast enough to compute the task in time; if it is not, the overflow for that task and that PE are stored for later evaluation. When the fit PE is found the task is mapped to that PE – a perfect fit, and in case when there is no perfect fit the task is mapped to PE with least overflow. The final schedule is calculated in the last stage of the mapping process.

Based on the algorithm discussed above, Automatic Mapping Tool has been developed in Java programming language and preliminary results are discussed in the next chapter.

3. Results and future work

This solution can also be classified as greedy and thus provides in most cases a suboptimal solution but in comparison with existing greedy algorithms: Load Balancing (LB) and Longest Processing Time (LPT) algorithms [Gajski et al. 2008] it exploits application concurrence. Final solution is accelerated proportionally to number of PEs in time proportional to $E \log V$ (in the worst case).

In future research a more “system – aware” approach with an in-depth knowledge of internal structure of each task is needed in order to find the most suitable among platform PEs. Also, systems with different bus architectures need to be explored. Another direction of development could address the case when MPSoC doesn’t provide fixed set of components and both the platform structure and mapping scheme could be recommended.

References


