Work-in-Progress: Code-Size-Aware Mapping for Synchronous Dataflow Graphs on Multicore Systems

Mingze Ma and Rizos Sakellariou
School of Computer Science, The University of Manchester, UK
{mingze.ma,rizos}@manchester.ac.uk

ABSTRACT
Synchronous Dataflow Graphs (SDFGs) are widely used to model streaming applications (e.g., digital signal processing applications), which are commonly executed by embedded systems. The usage of on-chip resources is always strictly constrained in embedded systems. As the cost of instruction memory is a significant part of on-chip resource costs, code size reduction is an effective way to control the overall costs of on-chip resources. In this work, a code-size-aware mapping heuristic is proposed to decrease the code size for SDFGs on multicore systems. The mapping heuristic is jointly used with a self-timed scheduling heuristic to decrease the code size of the original schedule. In preliminary experiments, the proposed heuristic achieves significant code size reduction for all the tested SDFGs without affecting throughput.

1 INTRODUCTION
Applications running on embedded systems are always strictly restricted by their memory usage, especially for programmable digital signal processors which always have extremely limited on-chip memory space [1]. Off-chip memory which can lead to extra cost, higher energy consumption and lower speed is often not a good option to solve the problem of limited memory. Therefore, minimizing memory usage becomes a critical objective for the optimization of applications on these systems.

Digital signal processing applications are commonly modeled by dataflow graphs. A widely used dataflow model is called Synchronous Dataflow Graph (SDFG). The memory usage of an SDFG is significantly decided by its scheduling process. Therefore, much work [1, 2] is done to reduce the memory usage by both data and code during scheduling on single core systems. Nevertheless, for multicore systems, most work only focuses on decreasing the memory usage by data [6]. However, as illustrated in [1], for realistic SDFGs, compared to minimizing the usage of data, minimizing code size often achieves much more reduction of memory usage. Thus, an interesting but not well-studied problem is how to reduce the code size on multicore systems by scheduling.

An SDFG is a directed cyclic graph which consists of nodes and directed edges. The nodes (actors) are used to model the computational modules of an application. The edges (channels) represent the data communication between actors. The communication data is represented by tokens transferred through first-in-first-out buffers on channels. The execution of an actor consumes tokens on its input channels and produces tokens on its output channels. SDFGs are executed iteratively. An actor in an SDFG always executes multiple times within an iteration due to various token producing/consuming rates of actors. The number of executions of an actor within one iteration is the repetition number of the actor. The average computation time per iteration is referred to as iteration period (IP). The speed of a streaming system is represented by throughput, which is the reciprocal of IP.

The target multicore platform in this paper is assumed to have a set of homogeneous processing cores which are connected by an ideal interconnection network, which means the communication overheads between cores are assumed to be zero. The code of a core is stored in its private memory and cannot be shared with other cores. This means if an actor is mapped on multiple cores, then multiple copies of the code of the actor should be stored separately on the private memory of the corresponding cores. No off-chip memory accessing is considered. The platform assumption is fit for some programmable multicore digital signal processors like [5].

Scheduling of an SDFG on a multicore system consists of mapping onto different cores and deciding the execution of sequences of the actors on different cores. Scheduling can be actor-to-core binding based [4], which means an actor can only be mapped on one core, or actor duplication-enabled [6], which means an actor can be mapped on one or more cores. For SDFG models, duplication-enabled scheduling enables multiple replications of an actor to be executed on more than one processing core, making the parallelism within an application to be better utilized. Therefore, duplication-enabled scheduling is preferable for the SDFG model in terms of improving throughput. For the target platform in this paper, the code size of the application is very likely to increase by actor duplication as an actor has to be mapped onto multiple cores; hence, code size reduction approaches are desirable for duplication-enabled scheduling.

Problem Definition Optimizing the mapping process of duplication-enabled schedules for SDFGs on the target multicore systems. The objective of the mapping process is to reduce the extra code size caused by the duplication of actors while maintaining the throughput of the schedule.

Promising Solution A code-size-aware mapping heuristic is proposed, which is combined with an existing self-timed scheduling heuristic to form Code-Size-Aware Scheduling (CSAS). Preliminary experiments show that compared with the original self-timed scheduling heuristic, our mapping heuristic achieves significant code size reduction for all the SDFGs in the experiment without decreasing the throughput of the outcome.
2 CODE-SIZE-AWARE MAPPING

The proposed code-size-aware mapping heuristic is shown in Algorithm 1. The duplication-enabled scheduling heuristic introduced in [6] named Schp(G, P) \textit{(desh)} is jointly used with the proposed mapping heuristic to form the csas algorithm. In each scheduling step of desh, a set of actors which are ready to execute \textit{RA}, a set of unoccupied cores \textit{AC} and a vector of current actor allocation on cores \textit{AL} are used as the input of the proposed mapping heuristic. An actor-to-core mapping pair \((a_p, c_p)\) is the output of the mapping heuristic. Then, desh executes the actor \(a_p\) on the core \(c_p\).

As shown in line 1 of the algorithm, a prior core of an actor is a core that the actor has been mapped on before. If an actor has no prior core, then all cores can be viewed as a prior core for the actor. Line 2 of the algorithm gives the actor with the highest priority \(a_p\). The priority of actors is successively decided by the number of prior cores, the quotient of the repetition number and the code size and the execution time of the actors. Then, line 3 gets the core with the highest priority \(c_p\) for \(a_p\). The prior cores of \(a_p\) are prior to be chosen as \(c_p\). The size of \textit{AL}(c) is used to break the tie. The worst-case time complexities of lines 1, 2 and 3 are \(O(NM)\), \(O(N)\) and \(O(M)\), respectively, where \(N\) is the number of actors of an SDFG and \(M\) is the number of cores. Therefore, the complexity of the mapping heuristic is \(O(NM)\). However, since the number of ready actors and available cores is rather small, in reality, the runtime of the heuristic is very short.

3 SIMPLE EVALUATION

The dataset used in our experiment consists of eight typical SDFGs which are all obtained from a zipped file in the Download/Benchmark section of the website [3]. The description of the file is “Explore throughput/storage-space trade-offs”. Some of these SDFGs are commonly studied artificial SDFGs like \textit{bipartite}, \textit{example}, \textit{fig8} and \textit{buffercycle}. The remaining SDFGs in the file are realistic streaming applications in digital signal processing domain like a \textit{modem}, a satellite receiver (\textit{satellite}) and a sample-rate converter (\textit{samplerate}), or in the multimedia domain like an \textit{H.263 decoder} (\textit{h263decoder}). The code size information is not provided by the source of these SDFGs, therefore we set the code sizes of the actors in these SDFGs as uniformly distributed random numbers between 10 and 100. The algorithms used in the experiment are implemented and tested in SDF3 [3].

REFERENCES

[5] Zhuyi Yu, Kaidi You, Ruijin Xiao, Heng Quan, Peng Ou, Yan Ying, Haofan Yang, Ming’e Jing, and Xiaoyang Zeng. 2012. An 800MHz 320mW 16-core processor with message-passing and shared-memory inter-core communication mechanisms. In ISSCC.