Task Scheduling for SoC systems with Data Communication on the Fly

extended abstract

Łukasz Maśko[1], Marek Tudruj[1,2]

[1] Institute of Computer Science of the Polish Academy of Sciences
ul. Ordona 21, 01–237 Warsaw, Poland
[2] Polish–Japanese Institute of Information Technology
ul. Koszykowa 86, 02–008 Warsaw, Poland
{masko, tudruj}@ ipipan.waw.pl

System on chip (SoC) computing [1] and emerging ultra high scale integration technology are currently susceptible to provide SoC modules embedding dozens of processors, inter–processor communication network and shared memory modules in a single chip. At the same time, the interconnection–centric SoC design [2] is seeking for new efficient inter-processor communication mechanisms, which condition reasonable use of so many processors on a chip. The success at the architectural and technological levels can not be consumed without adequate tools for automatic program design, particulary for parallel program task scheduling.

1 Dynamic SMP clusters based on SoC technology

This paper concerns scheduling of parallel programs for a class of clustered shared memory multiprocessor (SMP) systems, which are built of modules implemented in SoC technology, connected via a global network. Dynamic SMP clusters are created inside SoC modules around busses, which connect processors with data memory banks. Processors can be dynamically switched between clusters for program–defined time. The efficiency of the assumed architecture for typical numerical computations has been demonstrated by numerous simulation experiments [6].

Fig. 1a presents the general structure of the proposed system. Its basic elements are SoC modules interconnected by a global (peer to peer) network. A SoC containes processors P and memory modules M interconnected through local networks. Fig. 1b presents the general structure of a SoC module that uses a local data exchange network to connect processors with memory modules. Each processor is equipped with many data cache modules, which provide multi–ported access to/from memory modules. All memory modules are also connected to the external peer to peer global network.

Inter–processor data transfers inside a SoC module can be performed using reads on the fly, which consist in capturing data to one or more processor's data cache(s), while the data are present on a local cluster network (usually being written to memory by another processor). It allows to avoid multiple, sequencial reads of the same data from the shared memory to processors' data caches. Processor switching between clusters consists in connecting a processor to a new cluster (i.e. its local network). A processor switched to a cluster can bring in its cache data, which are useful for the cluster. When the processor writes data to memory, processors in the target cluster can read data on the

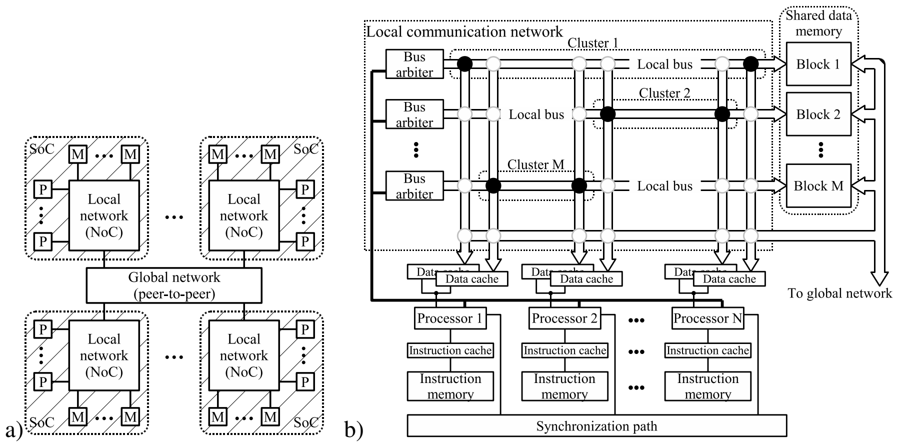

Fig. 1. General system structure a) and architecture of a single SoC module b)

fly. Exact synchronization of process, which writes with reading processes is necessary. Special inter–processor synchronization hardware has to be included in the system to enable parallel execution of many synchronization operations for the program.

Tasks in programs are so built that they do not require data cache reloading during their execution. All data have to be prefetched to processor's data cache before a task begins. Current task results are sent to the cluster memory module only after task completes. Such program execution paradigm, called cache–controlled macro data–flow principle, completely prevents data cache thrashing. The single assignment rule avoids cache consistency problem.

An extended macro dataflow graph representation (EMDG) is used to specify programs for this architecture. It is based on a standard macro dataflow program graph with new nodes representing memory module bus arbiters, the global data network arbiter, reads from memory modules to processor's data caches, writes from data caches to memory modules, processor switches between busses and barriers.

2 The Scheduling Algorithm

Program scheduling for parallel systems with a limited number of resources is in general, an NP–complete problem [4]. In the assumed architecture, it must take into account the cache–driven macro data flow execution paradigm as well as multi–level communication network. The paper presents a new algorithm, which takes into account the limited number of processors and is especially oriented towards modularity of target systems. The presented algorithm contains two phases: mapping of program graphs nodes to processors and processors to SoC modules and structuralization of communication inside SoC modules.

The first phase is a fusion of a genetic algorithm and a list scheduling with an ETF (Earliest Time First) heuristics [3, 4]. In this step, nodes of the initial program macro dataflow graph are mapped to a limited number of processors available in a system. The set of processors may be mapped on SoC modules in many ways. The processors are interconnected via a full network, in which the links between processors from the same SoC module are fast (local communication), while links between different SoCs are slow (global communication).

The mapping of computation nodes to processors is performed using a list scheduling algorithm with an ETF heuristics modified in such a way, that it respects the fact of division of a set of all processors in the system to SoC modules. A mapping routine is provided with the mapping vector $(a_1, \ldots, a_{MN}), 1 \leq a_i \leq M$, where N is the number of processors in a SoC module and M is the number of SoC modules in the system. Values in such vector mean that a logical processor i is mapped to a SoC module a_i. Communication between processors i and j is performed through the global interconnection network, if $a_i \neq a_j$, and through local communication network otherwise, which implies a possible usage of data transfers on the fly. To find the best mapping, the optimal mapping vector must be provided. To determine such vector, we use a genetic search algorithm. Each mapping vector (corresponding to one processor distribution scheme) constitutes one chromosome. The fitness function of an individual is computed based on execution time of a program graph scheduled with the ETF algorithm under the constraionts imposed by the chromosome (mapping vector).

The second phase transforms and schedules computations and communications between mapped computation nodes. It transforms the inter–processor communication inside SoC modules into data transfers on the fly and dynamic processor switching between memory modules. This step accounts for all other system features such as the number of busses and memory modules in a single SoC module, the number of ports and the size of processor data caches. This process is based on atomic subgraphs and their basic transformations, as described in [5].

Finally, the graph is tuned to include the last constraint — the size of the processors' data caches. The nodes and their influence on cache occupancy for each processor, are examined in the order determined by a simulated execution of the graph. Transformations include introduction of additional steering edges, writes from processors data cache to a shared memory block and reloading of this data back to processor's data cache, when this information is required.

The proposed scheduling algorithm has been implemented as a program package combined with a cycle accurate simulator of execution of program extended macro–data flow graphs in the proposed architecture. Simulation results of numerical programs graphs which asses particular features of the proposed scheduling algorithm are presented in the full paper.

3 Experimental results

As a testbed, a set of automatically generated *semi–regular* program graphs was used. The graphs consisted of 4 communication–intensive subgraphs executed in parallel. Each subgraph contained 7 layers of nodes, 3 to 4 nodes in each layer. The communication inside a subgraph took place only between consecutive layers. Additional rare random communication between the subgraphs was introduced. All the computing nodes had the same weights. Other parameters that varied between graph sets were the input degree of graph nodes and the weight of communication edges. There were 6 parallel

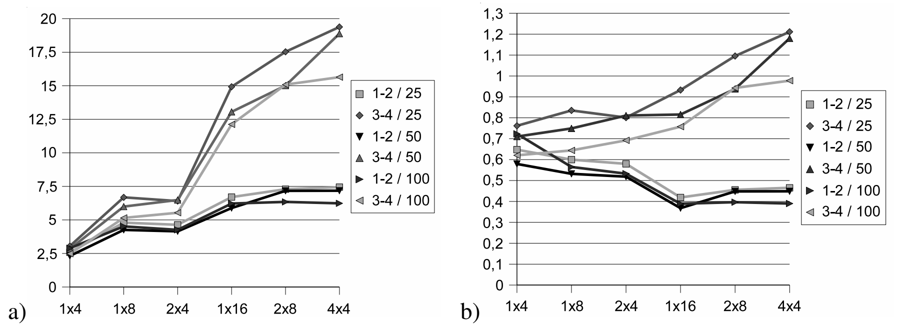

Fig. 2. Speedup a) and parallel processor utilisation efficiency b) against sequential execution for experimental graphs

configurations tested: 1 SoC module with 4 processors (1x4), 1 SoC module with 8 processors (1x8), 2 SoC modules with 4 processors each (2x4), 1 SoC modules with 16 processors (1x16), 2 SoC modules with 8 processors each (2x8) and 4 SoC modules with 4 processors in each (4x4). The executive system contained twice as many shared memory modules and local busses as the number of processors. The size of processors' data cache depended on the scheduled program graph and was the same for each tested hardware configuration and was equal to 2 times the maximal data size required by any of graph nodes. Communication–to–computation speed ratio was set to 4. The obtained speedups against execution of tested graphs on a single processor are presented in Fig.2a. Parallelization efficiency (parallel speedup per processor) is presented in Fig.2b.

References

1. Ch. Rowen, Engineering the Complex SOC, Fast, Flexible Design with Configurable Processors, Prentice Hall PTR, 2004.
2. J. Nurmi, H. Tenhunen, J. Isoaho, A. Jantsch, Interconnect–Centric Design for Advanced SoC and NoC, Springer Verlag, 2004.
3. J.-J. Hwang, Y.-C. Chow, F.D. Anger, C.-Y. Lee, Scheduling precedence graphs in systems with interprocessor communication times, *SIAM Journal on Computing*, 18(2), 1989.
4. J.Y-T. Leung, *Handbook of scheduling. Algorithms, models and performance analysis*, Chapman and Hall, 2004.
5. Ł. Maśko, Atomic operations for task scheduling for systems based on communication on–the–fly between SMP clusters, 2nd *International Symposium on Parallel and Distributed Computing, ISPDC 2003, Ljubljana, Slovenia*, October 2003, IEEE CS Press.
6. M. Tudruj, Ł. Maśko, Dynamic SMP Clusters in SoC Technology - Towards Massively Parallel Fine Grain Numerics, 11th *Proceedings of the* 6th *International Conference on Parallel Processing and Applied Mathematics PPAM 2005*, Poznań, Poland, September 2005, LNCS 3911, Springer Verlag, 2006.
7. A. Tchernykh, J. Ramírez, A. Avetisyan, N. Kuzjurin, D. Grushin, S. Zhuk, Two Level Job–Scheduling Strategies for a Computational Grid. In Parallel Processing and Applied Mathematics, *Proceedings of the* 6th *International Conference on Parallel Processing and Applied Mathematics, PPAM 2005. Poznan, Poland*, September 2005, LNCS 3911, Springer-Verlag.
8. O. Sinnen, L.A. Sousa, F.E. Sandnes, Towards a Realistic Task Scheduling Model, *IEEE Transactions on Parallel and Distributed Systems*, vol.17, no.3, March 2006.