Dynamic Task Allocation Strategies in MPSoC for Soft Real-time Applications

Eduardo Wenzel Brião, Daniel Barcelos, Flávio Rech Wagner
Universidade Federal do Rio Grande do Sul - UFRGS
Instituto de Informática
Porto Alegre, RS, Brazil
{ewbriao, danielb, flavio}@inf.ufrgs.br

Abstract

This work evaluates task allocation strategies based on bin-packing algorithms in the context of multiprocessor systems-on-chip (MPSoCs) with task migration capabilities, running soft real-time applications. The task migration model assumes that the whole code and data of the tasks are transferred from an origin node to the chosen destination node. We combine two types of algorithms to obtain better allocation results. Experimental results show that there is a trade-off between deadline misses and system energy consumption when applying bin-packing and linear clustering algorithms. In order to save energy, our system turns off idle processors and applies Dynamic Voltage Scaling to processors with slack. Depending on the algorithm selection and on the application, it is possible to obtain a reduction on deadline misses from 30% to 100% and energy consumption savings from 60% to 80%.

1 Introduction

The complexity of electronic embedded systems design has been increasing due to the technological evolution that allows the integration of a complete system on a single chip (SoC – System-on-Chip). In order to reduce design costs and time-to-market, systems are built by assembling pre-designed functional modules (processors, memory, dedicated hardware blocks), called IP (Intellectual Property) cores [1]. They can be reused from previous designs or acquired from third-party vendors.

Nowadays, the most commonly used communication architectures are not suitable for the communication requirements of future SoCs, such as scalability and performance. Networks-on-Chip (NoC) arise as a solution to fulfill these requirements [2].

Several applications of the same or different domains can be loaded and executed on an MPSoC platform, since it provides appropriate resources to allow the simultaneous execution of several applications. However, the system must know where and when allocating tasks that compose the application.

In distributed computing systems, tasks must be allocated as soon as they are available, such as to minimize the impact of system degradation, due to a bad workload distribution. Mechanisms for task migration are needed to provide the infrastructure that allows dynamic load balancing or concentration. They may enable resource savings, since resources would be typically planned for worst-case conditions that rarely occur. With this in mind, there are on-line algorithms that may be combined to decrease the communication among tasks and to allocate tasks to processors such that timing constraints are met. Linear clusterization [3] may be applied such that clustered tasks are allocated to a single processor in order to minimize inter-processor communication. This approach may be combined with bin-packing algorithms [4], which apply heuristics to allocate tasks to processors, considering the processor utilization. These algorithms present a low overhead and may be efficiently applied in a dynamic context, in embedded systems. Various combinations of bin-packing algorithms may explore different trade-offs between energy, power, performance, and real-time constraints.

This work presents the evaluation of several task allocation strategies based on bin-packing algorithms in the perspective of MPSoCs. We show different trade-offs between energy consumption and deadline misses when applying bin-packing algorithms for task allocation. We show that, even using a task migration mechanism with high overhead, it may be applied in embedded systems based on NoC architectures. The use of task migration is justified since it pays off the performance and energy costs involved in the system. Results show that task migration may greatly improve the fulfillment of task deadlines in soft real-time systems and decrease the energy consumption. In our work, the task migration is triggered when the allocation heuristic is executed. Afterwards, the task migration will be executed again only when a new application is loaded by the user.

The remaining of this paper is organized as follows. Section 2 discusses related work, while Section 3 shows our energy and task models. Section 4 presents our simulator and task migration support. Section 5 presents experimental results, and, finally, Section 6 draws main conclusions and addresses future work.

2 Related Work

Task allocation may be modeled as a bin-packing problem [4]. In bin-packing, the objective is to pack a set of items with given sizes into bins that have a fixed

capacity, and items whose total size exceeds this capacity cannot be assigned to the bins. There are four main bin-packing heuristics, but we concentrate on two of them: Best-Fit (BF) and Worst-Fit (WF). WF generates a task distribution with load balancing, while BF generates a distribution that is concentrated in some bins.

There are methods, such as clustering, that try either to minimize the inter-process communication or to explore the parallelism. Linear clustering totally explores the parallelism of the task graph, while non-linear clustering reduces the parallelism, by serializing independent tasks in order to reduce communication costs [3].

Few works cover task migration in the context of embedded systems. Bertozzi et al. [5] propose a user-managed migration scheme based on code checkpointing and a characterization methodology for task migration overhead in a shared memory. However, in this work, no figures for energy and power were measured. Wronski et al. [6] present a TLM SystemC NoC-based simulator, which executes clustering and bin-packing algorithms for task partitioning. The authors conclude that the combination of load concentration (BF) with DPM may result in lower energy consumption. However, the usage of load balancing (WF) minimizes the number of deadline misses. This work does not consider the task migration costs.

Our work, in turn, considers task migration overhead in a dynamic environment and shows the evaluation of bin packing algorithms in the context of NoC-based MPSoCs, in terms of energy and soft real-time constraints.

3 Energy and Task Models

The dynamic power consumption of the network routers and links is calculated with help of the Orion library [7]. The energy spent by a data phit to be transferred between two routers is defined as:

$$E_{phit} = E_{wrt} + E_{arb} + E_{read} + E_{xb} + E_{link} \quad [1]$$

where E_{wrt}, E_{arb}, E_{read}, E_{xb}, and E_{link} represent, respectively, the energy spent in writing the phit in the buffer, selecting the input channel, reading the phit from buffer, crossbar, and output channel. The static consumption of the memory is estimated by a model similar to [8]. In this work, the only component with power management capabilities is the memory and, in fact, it is the only one that really affects the results.

Each application is a directed acyclic graph $T = G(K, A)$, where each node $k_i \in K$ is a periodic task and each arc $a_{i,j} \in A$ is a dependency and flow of messages between tasks k_i and k_j. The arc weight $a^w_{i,j}$ represents the amount of bits to be transferred between the tasks. Each task $k_i \in K$ is a tuple $\{C, T, S, D, \alpha\}$, where C is the worst case execution time, T is the task period, S is the task size in bytes (including program size and data size), D is the task deadline, and α is the average number of gate switchings per cycle of the task in the core.

4 Simulator and Task Migration

4.1 Serpens Simulator

The SystemC NoC-based Serpens simulator has been developed to simulate the behavior of systems that run sets of synthetic tasks, which are dynamically loaded. The simulator also executes clustering and bin-packing algorithms for task partitioning and implements an on-line scheduling for tasks that are mapped to the same processor. DVS and DPM mechanisms are implemented to minimize energy consumption. The DAR (Dynamic Average Rate) algorithm [12] was used for DVS. The system is based on Java processors [9], each one with its private memory and on the NoC presented in [10], both of which have been developed in our research group. The Java processor implements an execution engine for Java in hardware through a stack machine compatible with the Java Virtual Machine (JVM) specification. We use the pipelined version of Femtojava. Each processor has its own local scheduler – we use EDF.

The simulator uses the Orion library to evaluate the power consumption. An in-house tool [11] is used to calibrate the processor power values. The simulation model uses NoC routers [10] that have been designed for the synthesis of low power and low area NoC-based embedded systems Each router has 5 bi-directional ports with input buffer size of 4 phits. The phit size is 4 bytes. More information about the simulator can be found in [6].

4.2 Task migration infrastructure

The task migration mechanism adopted in this work is based on a *copy* model. This model is very simple, with highest overhead, since the whole context (code, data, stack, and contents of internal registers) is migrated and there is no task execution during the transfer. Costs of task shut-off (deallocation of kernel-level data structures and user level-memory space) and task re-spawning (task creation system calls) are not taken into account. Of course they could bring an additional contribution to the migration cost.

In each processor, there are mechanisms for inter-process communication based on messages (send/receive primitives). Figure 1 illustrates the scheduling sequence for a task migration between two cores P0 and P3. The origin core P0 is executing a task T0. An interrupt happens, and the system decides to migrate a task T1 from P0 to P3. The label "MS" in the scheduling indicates that P0 (origin core) is sending T1 to P3. Processor P3 is in idle until receiving the migration packets sent from P0 ("MR"). When the last packet was received by P3, the scheduler "S" in P3 monitors if there is some new task, and the new task is released through "R" ("R"

inserts the new task in the ready queue of the core). The migrated task T1 then starts to run.

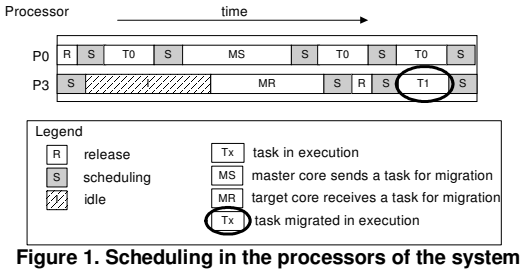

Figure 1. Scheduling in the processors of the system

It is assumed that the packet transmission time is much larger than the time of copying memory data. Therefore, in our migration scheme, we afford to neglect the time of copying memory data. To go through a router, the packet transmission time is 40 cycles, and one packet has 4 bytes.

5 Experimental Results

This section presents several experiments, based on three case studies and different network sizes, which demonstrate the feasibility and efficiency of the task migration mechanism.

5.1 Case studies

Experiments use two distinct applications, shown in Table 1. The first application is part of the embedded system synthesis benchmark suite (E3S) [13] and comes from the telecom domain. The 'telecom' application has a small number of tasks and a large volume of communication. The second application is a synthetic one. It has a small amount of communication, but a large number of tasks. As shown in Table 1, the average context size of each task is 1.3 KB and 1.1 KB for the 'telecom' and 'synthetic' applications, respectively.

Table 1. Case studies used in the experiments

Application	# Task	# Edges	# Avg. context size	#Comm.
Telecom	30	18	1.3 KB	850 KB
Synthetic	64	52	1.1 KB	0.052 KB

5.2 Simulation strategy

We implemented the WF and BF bin-packing algorithms, and they are combined with linear clusterization (LR). LR groups tasks with large communication with each other and can minimize the cost of communication among tasks, thus increasing the options of an appropriate algorithm to minimize figures such as energy consumption and deadline misses. In both case studies, we first simulate the applications with tasks allocated in an ad-hoc way. We simulate separately 10 different random task allocations and calculate the average for deadline misses and energy consumption. Afterwards, we consider that task migration has been required by some higher-level mechanism and apply the algorithms to decide on the task allocation after migration. The simulated time for all experiments was 200 ms. The processor frequency varies from 100 MHz to 600 MHz, and the voltage varies accordingly from 1.3 V to 2.0 V using DVS. The NoC runs at 266 MHz, and the network size varies from 4x4 to 7x7 cores. The memory size of each core is 64 KB.

5.3 Experiments

Tasks are considered to be periodic. They are scheduled and later on concluded, and then we count one *task finalization*, even if the task is terminated after its period. If this occurs, we count one *deadline miss*.

Figures 2 and 3 present, respectively, the deadline misses ratio of synthetic and telecom applications for each network size. These figures compare the situation before (ad-hoc allocation) and after task migration. As shown in Figure 2, there has been no significant improvement in the deadline misses ratio when the BF + LR allocation was applied, when compared to the ad-hoc allocation. Even grouping tasks with heavier communication between them, this algorithm could not decrease the number of deadline misses, since the cluster allocation exceeded 100% of processor capacity in some processors.

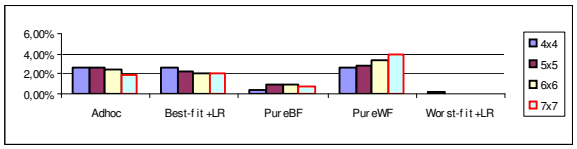

Figure 2. Deadline misses ratio of the synthetic application

The pure BF algorithm allocates single tasks. Deadline misses decreased when compared to an ad-hoc allocation, since tasks did not exceed processor capacity. The pure WF algorithm distributes tasks among processors. However, since there are communication dependencies between tasks, deadline misses increased. For larger NoCs, deadline misses increase, since there is a larger number of hops in sending messages. Even though, we have at most 4% of deadline misses, a reasonable rate for soft-real time systems. Finally, the WF + LR algorithm could reduce to 0% the deadline misses ratio for most NoC sizes. Because of the linear clustering process, communication between clusters is very low, thus reducing congestion and helping avoid deadlines.

Figure 3 shows the deadline misses ratio for the telecom application. We see an overall behavior similar to that of Figure 2. However, since communication volumes are higher than in the synthetic application, there is a larger ratio of deadline misses. In this case, even the WF + LR algorithm could not reduce deadline misses to 0%. We notice that, in the WF + LR algorithm, the NoC size does not impact the ratio of deadline misses, since for this application the number of clusters is less than 16, such that networks larger than 4x4 do not bring any considerable advantage.

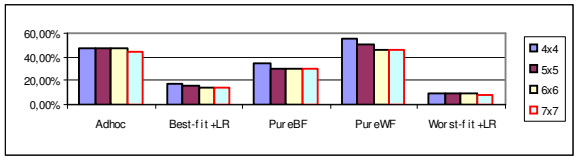

Figure 3. Deadline misses ratio of the telecom application

These experiments consider soft real-time applications, where a small ratio of deadline misses can be tolerated. Of course, deadline misses found in Figures 2 and 3 could be further reduced by increasing system throughput, especially network bandwidth and processors' frequencies, although increasing energy consumption. But the application of the WF + LR allocation algorithm significantly reduces the required increase in system throughput to achieve a given ratio of deadline misses.

Figure 4 shows the energy consumption resulting from all allocation algorithms applied to the synthetic application. We notice that the BF and BF + LR algorithms showed smaller energy consumption, since tasks and task clusters are concentrated on a smaller number of processors, while the remaining processors and their memories may be switched off. The WF + LR algorithm presents a constant energy consumption for 5x5 and larger NoCs, since cluster sizes are small than 25 and, even increasing the NoC size. The number of active processors will remain constant.

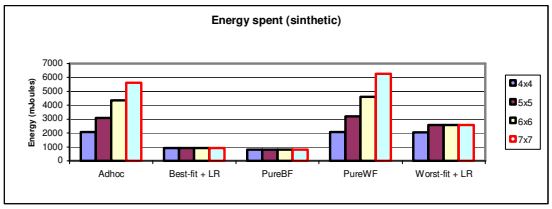

Figure 4. Energy spent in the synthetic application

Figure 5 presents the energy consumption, for a simulation time of 200 ms, resulting from all allocation algorithms applied to the telecom application. Algorithms BF and BF + LR present less energy consumption, while WF + LR presents less consumption than pure WF, since LR groups tasks with heavier communication between them.

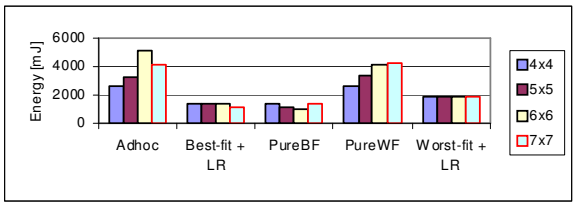

Figure 5. Energy spent in the telecom application

Results also show that migrations take less than 1% of the execution time and are responsible for approximately 2% of the total spent energy, on average. This means that migration costs can be easily amortized by the system, since energy consumption and deadline misses can be significantly improved after migration.

6 Conclusions and Future Work

This work demonstrated by experimental results that there is a trade-off between deadline misses and system energy consumption when applying bin-packing and linear clustering algorithms in MPSoCs where processors are interconnected by a network-on-chip. A small execution time of the task allocation heuristics is essential in order that the overall cost of the task migration may be amortized, and the combination of bin-packing and linear clustering shows this property. Our group is now working on a new task admission control algorithm, which already considers the communication costs in the scheduling phase.

As future work, we will measure the overhead imposed by on-line monitoring resources running on the processors. This monitoring will be basic to more advanced allocation algorithms.

References

[1] R.Bergamaschi, W.Lee. "Designing System-on-Chip using Cores". In: 37 th *Design Automation Conference* (DAC), USA, 2000, pp. 420-425.

[2] L.Benini, G.DeMicheli. "Networks on chips: a new SoC paradigm". *IEEE Computer*, vol.35, n.1, pp. 70-78. 2002.

[3] A.Gerasoulis, T.Yang. "On the Granularity and Clustering of Directed Acyclic Task Graphs". *IEEE Trans. Parallel Distrib. Syst*., v.4, n.6,. 1993. pp. 686-701

[4] D.Johnson. Near-optimal Bin-packing Algorithms. Cambridge, Mass. 1973

[5] S.Bertozzi, A.Acquaviva, D.Bertozzi, A.Poggiali. "Supporting Task Migration in Multi-processor Systems-on-chip: a Feasibility Study". In: *Design, Automation and Test in Europe Conference* (DATE), Munich, Germany, 2006, pp. 15-20.

[6] F.Wronski, E.Brião, F.R.Wagner. "Evaluating Energy-Aware Task Allocation Strategies for MPSoCs". In: IFIP Working Conference on Distributed and Parallel Embedded Systems (DIPES). Braga, Portugal, October 2006.

[7] H.Wang. "Orion: A Power-performance Simulator for Interconnection Networks". In: *ACM MICRO*. Istanbul, Turkey,. 2002. pp. 294-305

[8] J.Butts, G.Shoi. "A Static Power Model for Architects". In: *International Symposium on Microarchitecture*, *ACM Press*, California, 2000, pp. 191-201.

[9] S.A. Ito; L. Carro; R.P. Jacobi. "Making Java Work for Microcontroller Applications". IEEE Design & Test of Computers vol 18, Issue 5, 2001, pp. 100-110.

[10] C.Zeferino, A.Susin. "SoCIN: a Parametric and Scalable Network-on-Chip". In: Symposium on Integrated Circuits and Systems Design (SBCCI), 2003, pp.169–174.

[11] A. Beck, J. Mattos,; F. Wagner; L. Carro. "CACO-PS: A General Purpose Cycle-Accurate Configurable Power Simulator". In: Symposium On Integrated Circuits And Systems Design, 2003, São Paulo, Brasil. IEEE Computer Society Press, 2003. p.349-354.

[12] J. Zhuo, and C. Chakrabarti. "An Efficient Dynamic Task Scheduling Algorithm for Battery Powered DVS Systems" In: *Design Automation Conference Asia and South Pacific* (ASP-DAC), 2005, pp.846-849.

[13] EEMBC. The Embedded Microprocessor Benchmark Consortium. http://www.eembc.org 2006.