Multi-Objective Aware Communication Optimization for Resource-Restricted Embedded Systems

Olaf Neugebauer, Peter Marwedel
TU Dortmund University
Dortmund, Germany
firstname.lastname@tu-dortmund.de

Michael Engel
Leeds Beckett University
Leeds, United Kingdom
M.Engel@leedsbeckett.ac.uk

***Abstract*—Creating efficient parallel software for current embedded multicore systems is a complex and error-prone task. While automatic parallelization tools help to exploit the performance of multicores, most of these systems waste optimization opportunities since they neglect to consider hardware details such as communication performance and memory hierarchies. In addition, most tools do not allow multi-criterial optimization for objectives such as performance and energy. These approaches are especially relevant in the embedded domain.**

In this paper we present PICO, an approach that enables multi-objective optimization of embedded parallel programs. In combination with a state-of-the-art parallelization approach for sequential C code, PICO uses high-level models and simulators for performance and energy consumption optimization. As a result, PICO generates a set of Pareto-optimal solutions using a genetic algorithm-based optimization. These solutions allow an embedded system designer to choose a parallelization solution which exhibits a suitable trade-off between the required speedup and the resulting energy consumption according to a given system's requirements.

Using PICO, we were able to reduce energy consumption by about 35% compared to the sequential execution for a heterogeneous architecture. Further, runtime reductions by roughly 55% were achieved for a benchmark on a homogeneous platform.

***Index Terms*—Parallel programming, Parallel processing, Multiprocessing systems, Embedded software**

I. Introduction

Creating efficient parallel software for current embedded multicore systems is a complex and error-prone task. Numerous approaches that try to parallelize and map sequential applications to a multicore platform waste significant optimization potential. Parameters critical for the performance of software on such a platform, like communication performance between cores and the speed of different memories in the memory hierarchy, are often not considered in existing publications.

One reason for this negligence is that precise cost models for critical hardware components, such as latencies, throughput, and energy consumption of memories or other communication channels, were either not available or far too costly to evaluate. An optimization approach should be integrated into development cycles. Thus, an approach is required which constrains the required evaluation overhead for large numbers of hardware parameters by using high-level cost models.

In this paper we present a *Parallelism Implementer* and *Communication Optimizer* (PICO) infrastructure that enables the multi-criterial optimization of parallelized sequential C programs using a set of configurable high-level performance and energy models for the most performance-critical components of a given embedded multicore system. This use of high-level models enables its integration into state-of-the-art embedded software development toolflows and allows the evaluation of numerous potential parallelization configurations for a sequential program w.r.t. relevant system parameters.

The work presented in this paper concentrates on multi-objective optimization especially relevant for embedded systems. Apart from the achievable execution speed on a parallel platform, energy is the most important optimization criterion, especially for mobile platforms. Thus, in general there is no single optimal solution. Rather, each solution fulfilling a necessary minimal criterion, e.g., a required speedup, comes with one or more additional criteria, such as energy consumption. As detailed later in this paper, our multi-objective optimization approach employs genetic algorithm-based techniques to determine a Pareto-front of possible solutions for a given platform and parallelized application, which allows a developer to select the configuration most suitable for the task at hand.

Using PICO in conjunction with the PAXES parallelizer [1], [2] we detail the advantage of integrated multi-objective optimization in this paper. We use a set of standard embedded systems benchmarks, e.g. JPEG2000, and multiple platforms of homogeneous and heterogeneous nature to evaluate our approach. We observed that generated solutions utilize different data exchange capabilities of the platform which improve performance in terms of runtime and energy consumption.

To summarize, the main contributions of this paper are:

- A multi-objective communication optimization of parallelized C programs considering various communication mechanisms provided by the target platforms
- Optimization of performance and energy consumption using high-level cost models
- Integration of the optimization into a state-of-the-art parallelization and compiler toolflow

The rest of this paper is structured as follows. Section II presents related work. Section III describes the fundamentals of our approach. Detailed insight into our optimization algorithm is given in Section IV. Section V describes our framework. We evaluated our algorithm and provide a discussion of the results in Section VI. Section VII concludes this paper with a summary and gives directions for future work.

ISBN 978-3-8007-3657-7

II. Related Work

Application creation and parallelization techniques have been investigated in the last decades. Park et al. [3] presented a survey on software development approaches for multicore platforms. Ceng et al. published a semi-automatic user controlled parallelization framework [4] for sequential C code. Communication is modeled as logical channels between processing elements and the cost for communication is calculated by the size of the data and the estimated transfer time. Only a fixed communication type is used.

Process networks, especially Kahn process networks (KPN), are a common technique to model parallel applications in the embedded system domain. Processes communicate through unbounded FIFO queues. Verdoolaege et al. [5] presented a method to extract process networks for static affine nested loop programs. The tool is able to determine FIFO capacities and to optimize communication by removing channels and reducing the communication volume. Automatic buffer sizing was also analyzed by Cheung et al. [6]. Their approach determines the required channel size in KPNs to improve the performance.

In the last years research especially in the embedded system domain focused on heterogeneous multiprocessor systems on chip (MPSoC). Cordes et al. [1], [2] proposed several approaches to parallelize legacy C applications. The extraction of parallelism is able to consider the heterogeneity and resource-restrictions of embedded systems. Nevertheless, this approach only considers communication as a fixed overhead.

Modern MPSoCs provide several ways to exchange data between processors, e.g. hardware FIFOs or different types of memories. Nadezhkin et al. [7] presented an approach to map KPN applications onto a Cell BE platform. They analyzed different software implementations for FIFO communications in absence of hardware FIFOs. Realizing communication efficiently using windowed FIFOs was proposed by Haid et al. [8]. Their approach reduces the copy operations required to exchange data between processors.

Ferrandi et al. [9] presented a combined heuristic for mapping and scheduling of tasks and communication onto heterogeneous multiprocessor systems. The main objective is make-span and energy is not considered. Castrillon et al. [10] presented a KPN mapping technique onto heterogeneous MPSoCs. The Group-Based Mapping (GBM) heuristic is able to consider different communication resources and maps processes and communication at the same time. By analyzing different communication techniques in a state-of-the-art MPSoC, Odendahl et al. proposed a new communication cost model [11]. They split the communication cost into a sender and a receiver value and compared their approach with the single-cost model of the GBM heuristic.

In the area of design space exploration, Erbas et al. [12] proposed a genetic algorithm-based mapping approach which also considers communication between tasks. This approach uses a high-level representation of the application for optimization and simulation. In addition, only on specific communication type is modeled.

III. System Model

This section describes the target platform architecture used in this paper. Further, the programming model we apply is presented. Finally, the automatic parallelization algorithm is sketched and the drawbacks in case of communication optimization between tasks are discussed.

A. Target Platform

Modern MPSoCs comprise different memory types like fast and small scratch pad memories (SPM) or large DRAMs. In addition, some systems provide hardware support for data exchange like hardware FIFOs. To further increase the performance or to fulfill special requirements, processors with different characteristics are nowadays combined into a heterogeneous system. To abstract from real systems and to create platform-independent approaches, we use a high-level representation. Processing elements are connected through a communication infrastructure among themselves and the memories. Processors or accelerators are examples for processing elements and a bus or a NoC is an example for a communication infrastructure. In case of heterogeneous systems, we group processing elements by characteristics into logical groups.

B. Programming Model

In this paper parallel applications follow the *fork-join model* where a task can fork child tasks and suspends its own execution until all child tasks have finished their execution. Data is exchanged at the beginning and end of the tasks. Further, it is possible to transfer data between concurrently running tasks with a FIFO-style communication.

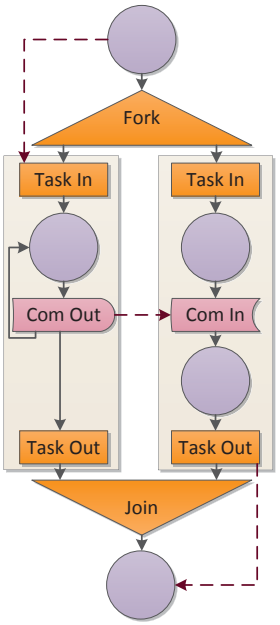


Fig. 1. Graphical representation of a parallel program

In the following, we use a high-level representation of a parallel program as shown in Figure 1. The graph consists of computations (circles), task management (triangles), task input and output, and communication nodes. Control flow is represented by solid directed edges whereas data which is communicated between tasks is drawn in dashed lines. For each variable which needs to be transferred a data edge is inserted. As shown in the example, the first computation node produces data used by a subtask. The concurrently executed tasks exchange data through communication nodes and one subtask generates data used in the computation of its parent task. For the optimization presented in this paper, a unique identifier is assigned to all task input/output and communication nodes. To simplify this model, *communication in* and *out nodes* share the same properties like FIFO size.

ISBN 978-3-8007-3657-7

C. Parallelization Process

For our parallelization approaches [1], [2] we use a hierarchical task graph to extract parallelism from sequential programs employing ILP-based and evolutionary algorithm-based approaches which can be applied to heterogeneous systems as well. During the parallelization process, communication is considered in a static FIFO-style. Thus, only one type of communication is used and the process cannot benefit from modern MPSoC communication optimization opportunities which might lead to infeasible or suboptimal solutions. A set of Pareto-optimal parallel solutions following the fork-join model (see Section III-B) is returned to the user.

In this paper we investigate opportunities of modern heterogeneous MPSoCs communication mechanisms coupled with this state-of-the-art parallelization process. We assume a parallelized application and analyze how various communication techniques influence the performance on different platforms. Thus, we identify the main problem of choosing communication techniques and their mapping onto the system. Transferred to the high-level representation, the challenge is to find a good mapping of task input/output and communication nodes to available hardware/software implementations with respect to resource restrictions. In the following, we present a genetic algorithm-based optimization approach using different communication mechanism prototypes.

IV. Genetic Algorithm-based Communication Optimization

To improve parallel application performance this paper investigates how various communication techniques influence performance on resource-restricted systems. First, we present the analyzed communication mechanisms followed by a detailed description of our optimization algorithm.

A. Communication Mechanism Prototypes

To analyze the benefits of using various communication mechanisms we implemented a set of different software FIFO prototypes distinguished by unique identifiers. All implementations use a common API which makes it easy to add new communication techniques into our approach. In the following, a brief overview of the characteristics is given.

In our model, global data structures encapsulate data transmitted from a parent task to its child and back. FIFO queues realize the communication between concurrently running tasks. We are using four different implementation prototypes but our approach is not limited to those implementations. Currently, all prototypes copy data from the sender to the FIFO and then to the receiver. Reading/writing is blocking if the FIFO is empty/full. The first implementation is very simple and performs busy polling on the FIFO status to check if access to a FIFO is possible. The second prototype utilizes a communication mechanism provided by the operating system. In our case RTEMS [13] is used which provides message queues to exchange data between threads. Those queues can only transport small data objects but FIFO management is done by the operating system. The third implementation is based on the first prototype and uses interrupts to check if data and space in the queue are available, respectively. This allows the processor to go to idle state which reduces energy consumption. Up to now all implementations only transmit one variable and multiple FIFOs are used to exchange variables between tasks. Transmitting each variable with its own FIFO implies a lot of management overhead. The last prototype implementation bundles the communication channels if possible.

As mentioned above we assume systems with different memories, which can be used for data exchange. Our approach can be extended easily to consider hardware supported communication mechanisms as well. During the implementation of the communication, a good mapping of channels onto memory with respect to resource-restrictions like available memory space is necessary. Thus we employ a genetic algorithm which finds good solutions for this problem.

B. Optimization Algorithm

We developed a genetic algorithm-based (GA) approach to evaluate possible communication-dependent optimization opportunities in our parallelization process. GAs are a method to find solutions (individuals) for a multi-objective optimization problem. As input, a GA requires a comprehensive representation of the structure of the individual called *chromosome*. Further, methods implementing *mutation*, *recombination* (*cross-over*) and *evaluation* of possible solution candidates are needed. Then, the GA creates an initial population which is evaluated for the considered objectives. Solutions with good results are selected and moved into a new generation. Some of them are then mutated or two promising individuals are combined and also added to the new generation. This generation is then evaluated again. This process is repeated until a certain stopping criterion is reached resulting in a set of Pareto-optimal individuals. We utilize the PISA framework [14] to implement the required functions for the GA.

1) Chromosome Structure: We modeled the previously presented decision problem (see Section III-C) of choosing communication techniques and their mapping with a single chromosome representation. The structure is shown in Figure 2. The chromosome consists of two parts. The first part describes the mapping of task input and output nodes onto a specific memory. Each available memory in the system has a unique identifier and the value of the gene determines onto which memory the node should be mapped. The second part of the chromosome defines the properties for each *communication out node* of the program graph. As described before, *communication in nodes* inherit their properties indirectly from the corresponding *communication out node*. Each *communication out node* is represented by four genes. The first gene indicates which type of FIFO should be used. The second defines onto which device or memory this FIFO should be mapped. In our case, the target platform does not provide hardware supported communication mechanisms. But those can be easily integrated in our algorithm as described in Section IV-A. The third and fourth genes describe properties of the employed communication mechanism. Here, the third gene represents the

ISBN 978-3-8007-3657-7

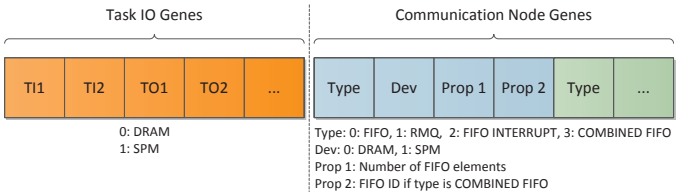


Fig. 2. Chromosome structure

capacity in terms of the number of elements this FIFO should have. In case of a combined FIFO where multiple variables are transmitted simultaneously, a set of feasible combinations of FIFOs is calculated during the setup of the whole optimization process. Thus, the last gene identifies one solution out of this solution space. If the GA creates an individual, where a node is mapped to a combined FIFO, the algorithm takes care that all nodes of the combined solution have the same properties and become a combined FIFO.

2) Chromosome Operations: Our GA-based algorithm requires a mutation and a cross-over function. The mutation function randomly changes the values of one or more genes (position in chromosome) of an individual. Increasing the FIFO capacity has a linear effect on the solution's runtime in case of blocking/stall time. In case of the FIFO size we only allow new values to vary by δ around the old capacity before the mutation process. We observed that an interval of ± 5 is a good value for δ. In addition, we limit the maximum FIFO size to 255 entries. This behavior can be disabled by the algorithm's user. If the type of *communication out node* is mutated to a combined FIFO, a default solution for this node out of the previously calculated solution space is selected. To keep the individual valid, all nodes inside this combined solution will be assigned to combined FIFO type and the FIFO capacity is also set to the same value. During the recombination (cross-over), two individuals are combined into a new individual. The algorithm takes care that the new individual is valid and repairs genes if necessary, e.g. in case of combined FIFOs.

3) Individual Evaluation: The evaluation of an individual is split up into three phases: *normalization*, *static* and *dynamic* evaluation. In the following, we describe the three phases of our evaluation algorithm:

a) Normalization: As mentioned above not every gene is required for the evaluation process, e.g. the combined FIFO identifier is not used for the other FIFO types. Therefore, we normalize the individuals which makes them easily comparable. This normalization process masks unused positions keeping the semantic of this individual.

Our algorithm uses a database where evaluation results of the normalized individuals are stored. If an individual has already been evaluated, the evaluation function loads the results from this database which drastically reduces the evaluation time. This database technique can be applied for deterministic target systems where two executions of the same application with the same input data and starting point results in the same behavior. For a nondeterministic system, each individual must be evaluated.

TABLE I
FREQUENCY-DEPENDENT HIGH-LEVEL PROCESSOR ENERGY MODEL

	Frequency		
State	100 MHz	250 MHz	500 MHz
Active Cycle	917.007 fJ	918.127 fJ	921.729 fJ
Stall Cycle	605.340 fJ	606.460 fJ	610.063 fJ
Idle Cycle	92.007 fJ	93.127 fJ	96.729 fJ

b) Static Evaluation: The next step in the evaluation process analyzes the individual's genes in a static way. Here, the required memory size for the communication implementation is calculated and validated whether this solution is feasible. If a solution can not be implemented on the actual system, the individual is invalidated and the result is stored in the database which ends the evaluation process for this individual.

c) Dynamic Evaluation: Finally, runtime behavior of an individual is evaluated by execution on the target system. Key objectives (runtime and energy consumption) are measured and collected. The performance indicators of an individual are the required memory size, runtime and energy consumption and those values are also stored into the database for faster lookup and reduced optimization time.

V. Framework

This section covers the used software infrastructure supporting our optimization algorithm. We implemented our approach in a tool called *Parallelism Implementer* and *Communication Optimizer* (PICO) utilizing the MACC [15] framework. The parallelization tool PAXES [2] passes a Pareto-front of parallelized solutions of a sequential application to PICO. By utilizing ICD-C [16], a C compiler front-end, our algorithm extracts data dependencies and implements the required data exchanges. Task creation and management is implemented using a lightweight runtime library utilizing R^2G [17]. As operating system we use RTEMS [13]. PICO operates fully parallelized such that all individuals of a generation can be evaluated in parallel to reduce the optimization time. Finally, a set of Pareto-optimal solutions is returned to the user.

The homogeneous and heterogeneous platforms composed of ARM processors are created and simulated using Synopsys Virtualizer [18]. Runtime measurements start at the beginning of the application and end after termination of the application. Thus, booting and shutting down the operating system is not considered. Energy consumption is evaluated utilizing a high-level model. For this purpose we implemented a Metrix component attached to our simulation environment which is triggered on every instruction, bus, cache and memory access and accumulates the corresponding energy values during evaluation. For the processor we use energy values depending on the current processor's frequency as shown in Table I. The energy consumption for the bus is a static value for each access. Memory values were obtained from CACTI [19].

VI. Evaluation

This section presents our evaluation environment and the results we obtained. First, we describe the test platform and

ISBN 978-3-8007-3657-7

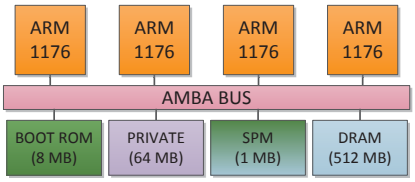


Fig. 3. Target platform (Caches and memory controllers hidden)

benchmarks used. Following, the results of the communication optimization are presented. Finally, implications from these results are discussed.

A. Evaluation Environment

To evaluate our multi-objective GA-based communication optimization we used applications from the UTDSP benchmark suite [20] and other representative applications which are often used on MPSoCs, like a JPEG encoder. We created a platform containing four ARM processors and four memories connected through a bus as depicted in Figure 3. For simplicity, memory controllers and caches are not included in the figure. In the homogeneous case, all processors are running at the same frequency of 500Mhz. For the heterogeneous case, we employ a single-ISA platform inspired by ARM's big.Little [21] paradigm where a system is composed of processors with different performance. To achieve a comparable behavior, we adjust the frequencies of the processors such that the heterogeneous platform consists of 2 processors running at 500Mhz, one at 250Mhz and one at 100Mhz. As mentioned before, our systems have four memories. The first memory (8MB) is reserved for the operating system and thus not usable for communication. For each processor a fixed partition is reserved in the second memory (64MB) where usually private data like stacks are stored. A small scratch pad memory (SPM) (1MB) is partially used by the operating system and free space can be used for communication. The last memory is a 512MB DRAM where communication channels can be mapped to.

B. Results

In this section we present results generated by our GA-based communication optimization approach. We present two applications (Spectral, JPEG2000) from our investigated benchmark set which benefit from our optimization. Both applications use a pipeline structure to process data which requires data transfer between concurrently running tasks. For demonstration purposes in this paper, we selected one parallelization solution for each application and platform from the Pareto-front generated by the automatic parallelizer. We chose a population size of 30 individuals and 15 generations. We allow mutation of already evaluated individuals to increase the search space of our algorithm by reducing the number of equal solutions. To compare our results with a naive communication implementation, we added two predefined solutions to the results. The first solution (ID 1) implements all communication with standard (polling) FIFOs of capacity one mapped to DRAM. In addition, all data transmitted between parent and children is stored in the SPM. The second solution (ID 2) is equal to solution ID 1 except for the FIFO size. Here, we use a high capacity (255 elements if possible) to reduce stalls. We limited the available SPM memory for communication to ensure that enough space on the SPM is left for the operating system. We conducted 3 runs for each benchmark with SPM size of 1kB and 2kB for both benchmarks, 512 bytes for JPEG and 3kB for Spectral. These configurations investigate the algorithm's performance for resource-restricted systems. A SPM capacity of 512 bytes is not insightful for the Spectral benchmark because the communicated data do not fit onto the SPM in this configuration. In the following, only Pareto-optimal solutions are presented.

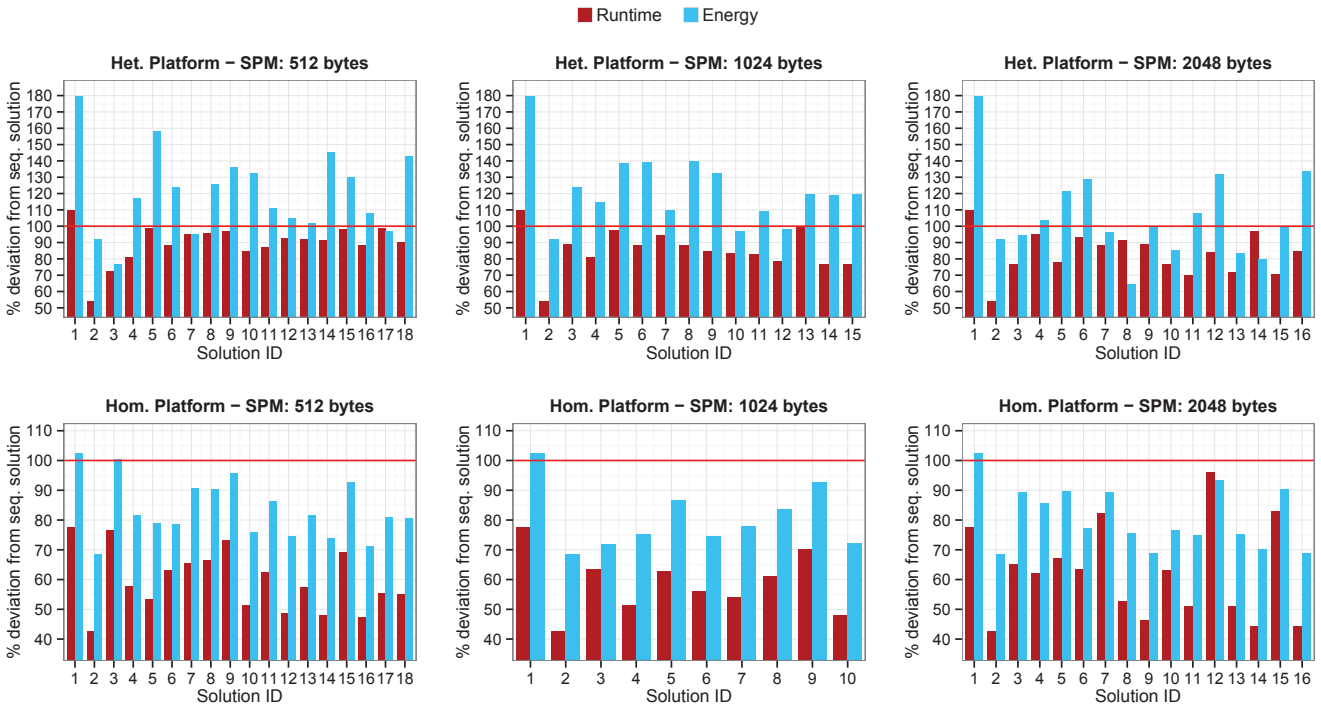
1) Performance: Figure 4(a) shows the results for simulated runtime and energy consumption for the JPEG2000 benchmark on the heterogeneous and homogeneous platform and Figure 4(b) for the Spectral benchmark. Results show the deviation from the sequential solution for each individual indicated by a red line at 100%. Thus, values under the red line mean performance increases and values above decreases compared to the sequential solution. Only one processor is active for the sequential solution and the other three are disabled and not considered in terms of energy consumption. As mentioned before, solutions 1 and 2 describe the corner cases of a naive communication implementation and give a good impression of possible speedups for the parallelizations which were generated by the PAXES parallelizer. Solution 2 usually performs best in case of runtime because it eliminates all FIFO stalls resulting from insufficient capacity, but this solution drastically increases the memory requirements (cf. Figure 6). For the JPEG benchmark, the results show that our algorithm is able to optimize multiple objectives simultaneously. For example, a reduction of energy consumption is achieved by about 25% and runtime reduction of about 28% for the heterogeneous platform with 512 bytes SPM (solution 3). If energy consumption is more important for a developer, our algorithm is able to generate a solution with reduction of about 35% for the energy consumption (SPM 2kB, solution 8).

The results of runtime and energy consumption for Spectral are depicted in Figure 4(b). For this benchmark, the parallelization does not achieve high speedups as indicated by solutions 1 and 2 but our approach was able to reduce the energy consumption compared to these solutions. Small runtime reductions were also achieved for some solutions.

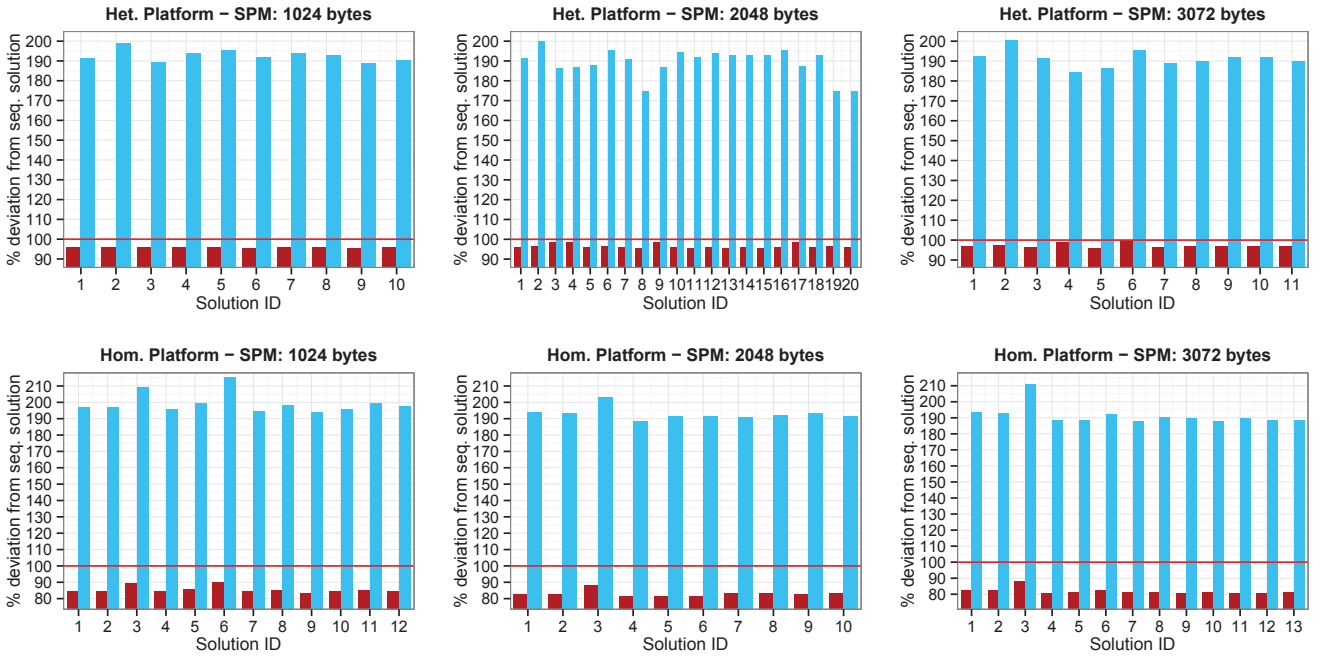
2) Communication Mapping: The GA-based communication optimization algorithm presented in this paper chooses between four different communication prototypes to transmit data between concurrently running tasks as described in Section IV-A. Figure 5(a) and Figure 5(b) show the applied prototypes for the JPEG and Spectral benchmarks. The algorithm implemented communication with standard FIFO or interrupt-based FIFO for JPEG. For solution 3 on the heterogeneous platform with 512 bytes shown in Figure 4(a), the interrupt-based FIFO prototype leads to the energy savings because a task goes to idle state if it is blocking, which also allows the processor to idle which reduces energy consumption.

For the Spectral benchmark, the algorithm created solu-

ISBN 978-3-8007-3657-7



(a) JPEG2000



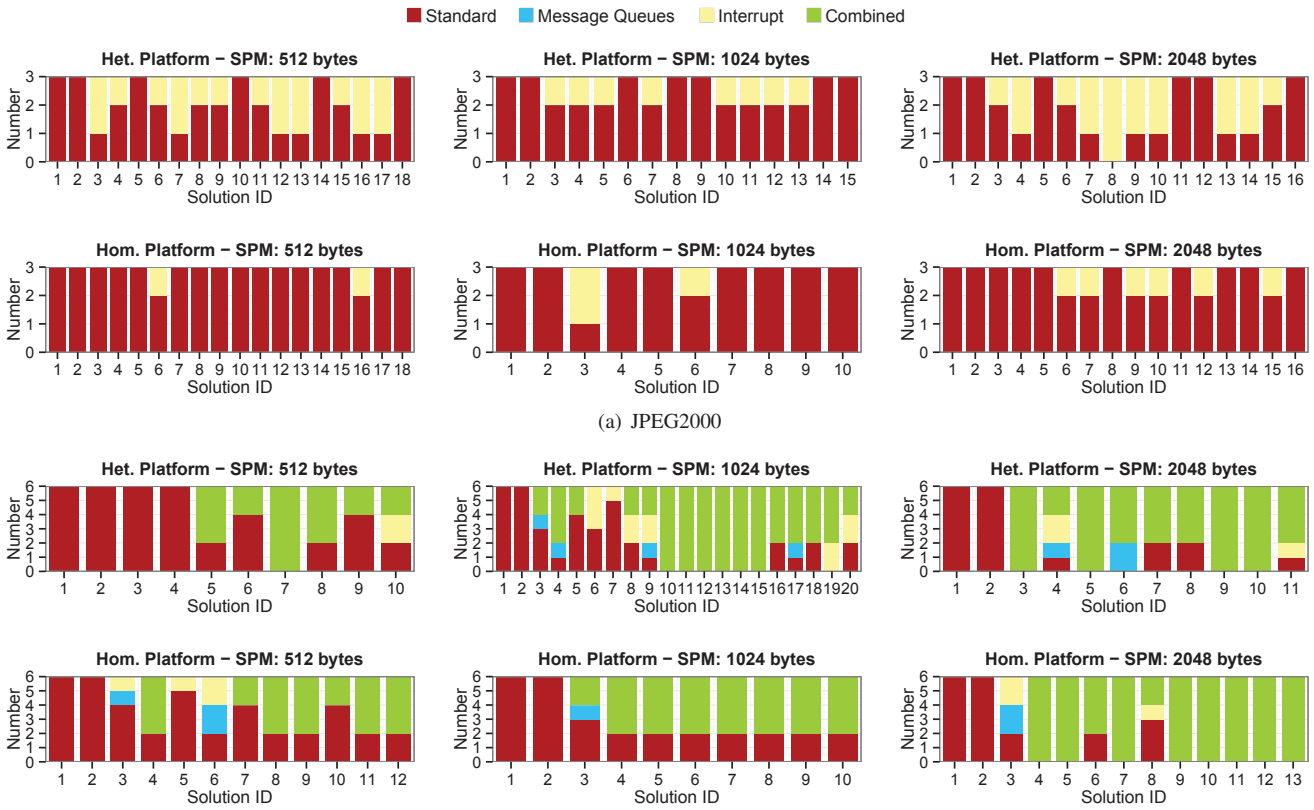
(b) Spectral

Fig. 4. Performance

tions which use all communication prototypes as depicted in Figure 5(b). Combined FIFO solutions are preferred subsequently, leading to reduced management overhead. In addition, solutions exist which use the communication mechanisms provided by the operating system. The optimization algorithm also creates a good mapping for data transmission between parent and child tasks which can be observed indirectly in the memory footprint presented in the following.

3) Memory Footprint: The memory requirements for the generated solutions of the optimization algorithm are shown in Figure 6(a) for JPEG and Figure 6(b) for Spectral respectively. As expected, solution 2 has a large memory footprint. The actual values for the maximum capacity FIFOs are not shown to increase the visual comparability of the generated solutions. The results show that our algorithm is able to utilize the restricted SPM. The presented GA-based communication

ISBN 978-3-8007-3657-7



(a) JPEG2000

(b) Spectral

Fig. 5. Communication Types

optimization algorithm uses runtime, energy consumption and SPM utilization as objectives. Solutions with large RAM requirements are valid, e.g. solution 3 of the Spectral for 3kB SPM. If communication mapped onto DRAM should also be reduced, the evaluation function of the GA can be extended easily to fulfill this requirement. The results prove that consideration of various communication techniques is necessary to further optimize applications parallelized by PAXES. Runtime reductions by roughly 55% and energy consumption reductions of about 35% were achieved. At the moment the approach introduced here does not provide feedback information to PAXES. This information could improve the results of the parallelization process as indicated by the obtained results presented before. In general, our results emphasize that utilizing communication capabilities of modern MPSoCs is crucial to execute applications efficiently.

All (cycle-accurate) simulation-based approaches have the drawback of long simulation runs for the evaluation. Static models could reduce the evaluation time. Creating accurate static models for runtime and energy consumption for modern MPSoCs is a complex task. Thus, simulation-based approaches are a good way to find solutions or optimization directions.

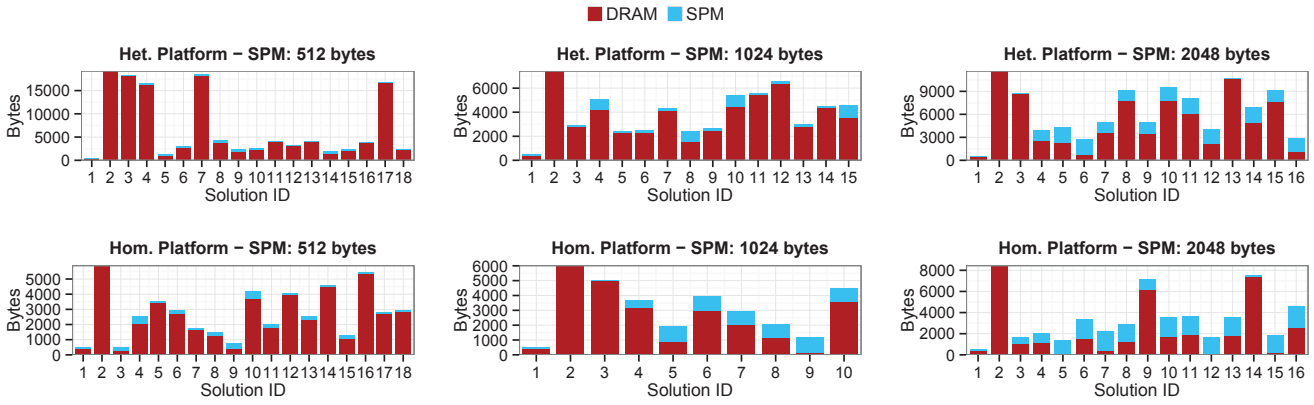
VII. Conclusion

Creating software for modern embedded heterogeneous and resource-restricted multiprocessor systems is a complex task. Parallel applications require efficient data exchange between concurrently running tasks. Today's MPSoCs provide various ways to transmit data between processing units, like memories or hardware FIFOs. To improve the performance utilizing these techniques is crucial.

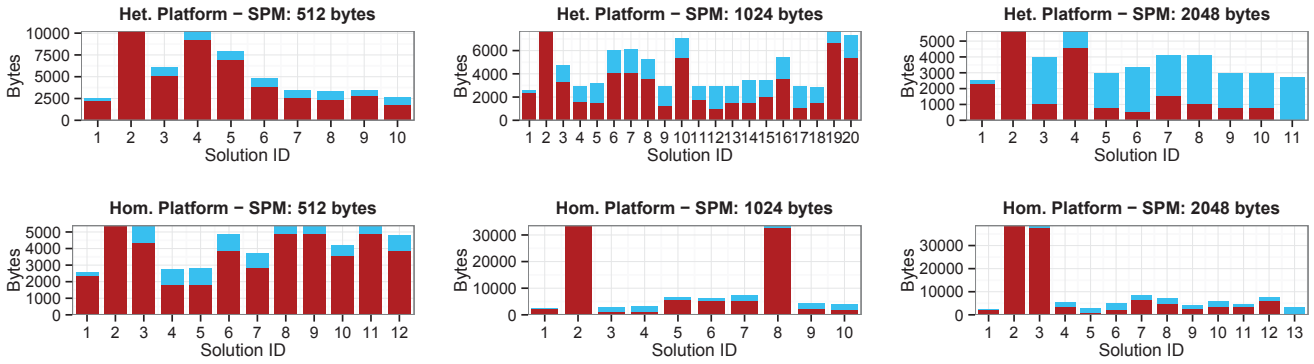
We created a multi-objective aware communication optimization of parallelized C programs which is able to exploit various communication mechanisms. Our approach is able to optimize runtime, energy and memory performance using a high-level cost model by using genetic algorithms. We combined our algorithm with PAXES, a state-of-the-art automatic parallelizer, and evaluated our approach with real-world benchmarks for two platforms. The results show that our algorithm is able to optimize parallelized applications and to provide more detailed performance numbers compared to PAXES. We were able to reduce the energy consumption by around 35% for JPEG compared to the sequential application. Furthermore, runtime reductions of roughly 55% were achieved for this benchmark on the homogeneous platform. The optimization algorithm achieves a reduction of energy consumption by about 15% compared to the naive communication implementation even for suboptimal parallelized applications like the presented Spectral benchmark.

Further investigation in providing feedback regarding communication costs to the parallelization process could enable new parallelization opportunities which might lead to better performance. Static models could be used to generate solution candidates without time consuming simulation.

ISBN 978-3-8007-3657-7



(a) JPEG2000



(b) Spectral

Fig. 6. Memory Consumption

ACKNOWLEDGMENT

The authors would like to thank DFG for supporting part of this work within the Collaborative Research Center SFB 876 “Providing Information by Resource-Constrained Data Analysis”, project B2, and Synopsys for the provision of the virtual prototyping IDE Virtualizer [18].

REFERENCES

[1] D. Cordes, M. Engel, O. Neugebauer *et al.*, “Automatic Extraction of Pipeline Parallelism for Embedded Heterogeneous Multi-Core Platforms,” in *Proc. of CASES*, 2013.

[2] D. Cordes, O. Neugebauer, M. Engel *et al.*, “Automatic Extraction of Task-Level Parallelism for Heterogeneous MPSoCs,” in *Proc. of ICPP*, 2013.

[3] H.-W. Park, H. Oh, and S. Ha, “Multiprocessor SoC design methods and tools,” *IEEE Signal Processing Magazine*, vol. 26, no. 6, 2009.

[4] J. Ceng, J. Castrillon, W. Sheng *et al.*, “MAPS: An Integrated Framework for MPSoC Application Parallelization,” in *Proc. of DAC*. ACM Press, 2008.

[5] S. Verdoolaege, H. Nikolov, and T. Stefanov, “pn: A Tool for Improved Derivation of Process Networks,” *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, 2007.

[6] E. Cheung, H. Hsieh, and F. Balarin, “Automatic buffer sizing for rate-constrained KPN applications on multiprocessor system-on-chip,” in *Proc. of HLDVT*, 2007.

[7] D. Nadezhkin, S. Meijer, T. Stefanov *et al.*, “Realizing FIFO communication when mapping kahn process networks onto the cell,” in *Proc of. SAMOS*, 2009.

[8] W. Haid, L. Schor, K. Huang *et al.*, “Efficient execution of Kahn process networks on multi-processor systems using protothreads and windowed FIFOs,” in *Proc. of ESTIMedia*, 2009.

[9] F. Ferrandi, P. L. Lanzi, C. Pilato *et al.*, “Ant Colony Heuristic for Mapping and Scheduling Tasks and Communications on Heterogeneous Embedded Systems,” *IEEE TCAD*, vol. 29, no. 6, Jun. 2010.

[10] J. Castrillon, A. Tretter, R. Leupers *et al.*, “Communication-aware mapping of KPN applications onto heterogeneous MPSoCs,” in *Proc. of DAC*, 2012.

[11] M. Odendahl, J. Castrillon, V. Volevach *et al.*, “Split-cost communication model for improved MPSoC application mapping,” in *Proc. of SoC*, 2013.

[12] C. Erbas, S. Erbas, and A. Pimentel, “A multiobjective optimization model for exploring multiprocessor mappings of process networks,” in *Proc. of CODES+ISSS*, 2003.

[13] RTEMS, “RTEMS Operating System — Real-Time and Real Free,” http://www.rtems.com/, 2014.

[14] S. Bleuler, M. Laumanns, L. Thiele *et al.*, “PISA—A Platform and Programming Language Independent Interface for Search Algorithms,” in *Proc. of EMO*, 2003.

[15] R. Pyka, F. Klein, P. Marwedel *et al.*, “Versatile System-level Memory-aware Platform Description Approach for embedded MPSoCs,” in *Proc. of LCTES*, 2010.

[16] R. Pyka and J. Eckart, “ICD-C Compiler framework,” http://www.icd.de/es/icd-c/icd-c.html, June 2014.

[17] A. Heinig, “R2G: Supporting POSIX like semantics in a distributed RTEMS system,” TU Dortmund, Faculty of Computer Science 12, Dortmund, Tech. Rep. 836, Dec. 2010.

[18] Synopsys, “Virtualizer, Virtual Prototyping Solution,” http://www.synopsys.com, 2014.

[19] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, “CACTI 6.0: a tool to model large caches,” HP Laboratories, Tech. Rep., 2009.

[20] C. G. Lee, “UTDSP Benchmark Suite,” http://www.eecg.toronto.edu/~corinna/DSP/infrastructure/, 2014.

[21] Peter Greenhalgh, ARM, “Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7,” http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf, 2013.

