2010 IEEE Annual Symposium on VLSI

Mapping embedded applications on MPSoCs: the MNEMEE approach

Christos Baloukas, Lazaros Papadopoulos, Dimitrios Soudris,

Institute of Communications and Computer Systems (ICCS), 9, Iroon Polytechniou Str., 15773, Athens, Greece

Sander Stuijk, Eindhoven University of Technology, Eindhoven, Netherlands

Olivera Jovanovic, Florian Schmoll, Design Automation for Embedded Systems Group, TU-Dortmund, Germany

Daniel Cordes, Robert Pyka, Informatik Centrum Dortmund e.V., Dortmund, Germany

Arindam Mallik, Stylianos Mamagkakis, IMEC vzw, Leuven, Belgium

François Capman, Séverin Collet, Thales Communications France, 160 boulevard de Valmy, 92704, Colombes, France

Nikolaos Mitas, Dimitrios Kritharidis, Intracom Telecom, Athens, Greece

Abstract— **As embedded systems are becoming the center of our digital life, system design becomes progressively harder. The integration of multiple features on devices with limited resources requires careful and exhaustive exploration of the design search space in order to efficiently map modern applications to an embedded multi-processor platform. The MNEMEE project addresses this challenge by offering a unique integrated tool flow that performs source-to-source transformations to automatically optimize the original source code and map it on the target platform. The optimizations aim at reducing the number of memory accesses and the required memory storage of both dynamically and statically allocated data. Furthermore, the MNEMEE tool flow performs optimal assignment of all data on the memory hierarchy of the target platform. Designers can use the whole flow or a part of it and integrate it into their own design flow. This paper gives an overview of the MNEMEE tool flow along. It also presents two industrial case studies that demonstrate who the techniques and tools developed in the MNEMEE project can be integrated into industrial design flows.**

I. INTRODUCTION

In today's embedded systems market there is a trend towards integrating more and more services on mobile devices. These systems combine applications from various domains like communications and multimedia (e.g., HD-video coders, wireless protocols, image processing and 3D games). All these examples make heavy use of data transfers and computation, which makes multi-processor platforms the perfect candidate for their implementation. However, Multiprocessor Systems-on-Chip (MPSoCs) include complex memory hierarchies and synchronization systems, which in turn makes it difficult to map an application on an MPSoC. For such complex design targets, the design choices will have a high impact on the performance of the system and the success of the device on the market.

There are several challenges that MPSoC designer faces. First, the application should be parallelized to take advantage of the multiple processor system. This is achieved by breaking the execution into several tasks that can be performed in parallel. Second, the statically and dynamically allocated data structures must be optimized to take advantage of the memory hierarchy of the target platform. Third, the final optimized parallel code should be mapped and compiled onto the MPSoC.

In current embedded system design flows, the optimization of the source code is done mainly by hand. This process is becoming very tedious and error prone as the complexity of the system and the applications are constantly increasing. This makes it necessary to develop a systematic methodology and tooling that will deal with of the aforementioned challenges automatically.

The MNEMEE project [1] comes to address the challenge of mapping an application onto an MPSoC platform. Several state-of-the-art source-to-source optimization methodologies and tools have been developed that automatically extract possible parallelization from a target source code, while they also optimize statically and dynamically allocated data structures for less memory accesses and storage. Additionally, the code is mapped onto the various processors and memories of the MPSoC platform. Each methodology and tool has a very clearly defined interface that allows the whole set of techniques developed under the MNEMEE project to be integrated into a common tool flow or used as individual optimization steps. It is this exact feature that renders the MNEMEE tools easy to use in industrial applications, where companies already work with certain tool chains. By integrating the MNEMEE tools into their tool chain, industrial designers can automate their flows. As will be demonstrated in section III, parts of the tool flow can be used to replace existing manual or less efficient steps in an industrial design flow. Two examples are provided, one from the communications domain and one from the multimedia domain.

The remainder of this paper is organized as follows. The next section gives an overview of the developed techniques of the MNEMEE project. Section III presents two industrial cases that demonstrate the applicability of the MNEMEE approach. Section IV concludes this paper.

This work was supported in part by the EC through FP7 IST project 216224, MNEMEE.

II. MNEMEE TOOL FLOW

A. *Overview*

Figure 1 shows the MNEMEE tool flow. The input of the tool flow is sequential source code written in C. The output of the flow is source code that is parallelized and optimized for the target MPSoC and its memory hierarchy. The first step targets the optimization of all dynamically allocated data structures by changing their implementation. These changes may affect the parallelization and therefore they are performed first. Step 2 identifies any potential parallelization in the source code and implements it by breaking the code into several tasks. Step 3 optimizes the statically allocated data structures. Since the parallelization of the original source code has already taken place, the tool can map the statically allocated data structures efficiently onto the memory hierarchy, as their size and behavior is known. The next step, maps the dynamically allocated data structures onto the memory hierarchy. Step 5 maps the parallelized application onto the processors and memories. The MNEMEE tool flow offers two alternatives mapping techniques. The first technique called scenario-aware mapping, tries to exploit the dynamic behavior of an application in order to save resources. The second technique, called memory-aware mapping, focuses on finding a mapping that minimizes the energy consumption of the memory subsystem. Finally, step 6 further optimizes the scratchpad allocation of each processor in the target platform.

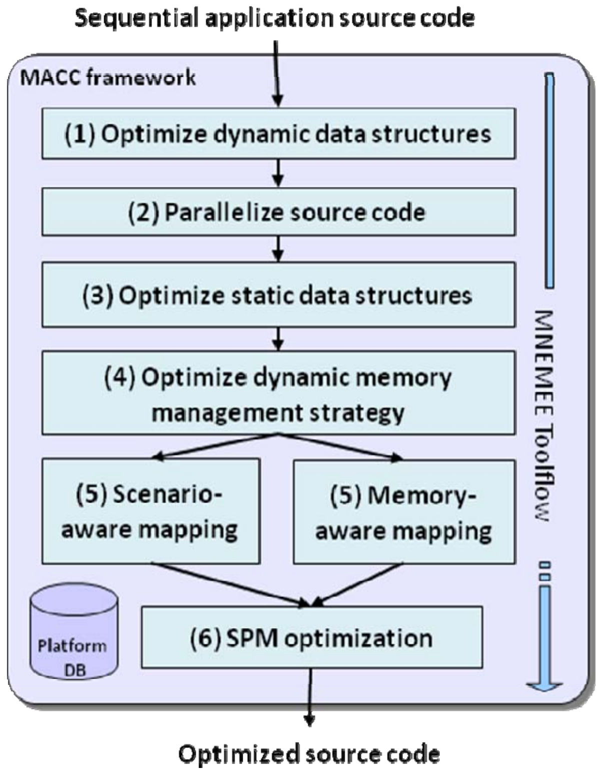
Sequential application source code
MACC framework
(1) Optimize dynamic data structures
(2) Parallelize source code
(3) Optimize static data structures
(4) Optimize dynamic memory management strategy
(5) Scenario-aware mapping
(5) Memory-aware mapping
Platform DB
(6) SPM optimization
MNEMEE Toolflow
Optimized source code

Figure 1 MNEMEE tool flow.

To combine the large number of required processing steps into a single tool flow, the MACC framework for source level optimization development has been used. This framework exploits the abstract syntax tree code representation provided by the ICD-C compiler development framework. MACC provides a common platform model along with a well defined interface for integration of analysis and optimization tools. This framework was partially developed in the MNEMEE project [9]. Furthermore, a graphical user interface is provided to enhance the usability of the tool flow.

B. *Dynamic data type optimizations*

The dynamic data type optimizations step (step 1) changes the implementation of all dynamic data structures like dynamic arrays, linked lists and trees, based on the Dynamic Data Type Refinement (DDTR) methodology [15]. All dynamic data types (DDTs) are profiled to identify their access patterns and allocation behavior. The optimized implementation is customized to fit this particular access pattern. The objective is to boost performance and restrain the memory consumption. This customization of the data structures is achieved by restructuring these data structures using components from a library of DDTs. As an example, we can consider a list that is accessed sequentially. The use of a memory pointer to store the last accessed element can greatly boost the performance without having to resort to an array solution. The linked list can be restructured adding this memory pointer component.

C. *Parallelization*

The parallelization tool is the second one in the MNEMEE tool flow. Its purpose is a fully automatic exploitation of parallelism for the given sequential application's source code, given by the MACC framework. Therefore, the application is transformed in a so called hierarchical task graph, based on the one presented by Girkar in [10]. Once this intermediate representation is extracted from the source code, an ILP based parallelization approach is used to search for parallelism on each hierarchical level of the task graph. Only by introducing hierarchy into the task graph representation it is possible to make usage of very complex ILP formulations. The whole graph is analyzed for possible parallelism by a bottom-up search strategy, which extracts a balanced, parallelized version of the application's source code. Though, the algorithm deliberates about whether it should extract new tasks on the current level of hierarchy or whether it should just use the tasks, extracted earlier deeper in the hierarchy. In addition, the parallelization tool also considers special requirements for embedded systems and is e.g., able to limit the number of extracted concurrently executed tasks to the number of available processor units.

Once the whole graph is processed by the parallelization algorithm, the taken decisions are annotated on the application's source code and a parallelization specification is generated, which complies with the input specifications of the MPMH tool [7]. This tool is then used in step 3 of the tool flow to implement the extracted parallelism.

D. Optimize Static Data Structures

The MPSoC Parallelization and Memory Hierarchy (MPMH) tool [7] generates parallel source code based on the directives specified by the previous step in the tool flow. This step has annotated the source code with parsections. These parsections specify the sequential code segments that must be parallelized. A directive has been added to each parsection that specifies the number of threads that will execute this parsection in parallel. The parallelization tool (step 2) further indicates whether the work that is done in the parsection must be divided over these threads in terms of functionality (i.e., to split the parsection based on functional parallelism), or in terms of loop iterations (i.e. to split the parsection based on data parallelism), or a combination of both depending on what is the most appropriate for a given parsection. Given the source code and the parallelization directives, the MPMH tool generates a parallel version of the code and insert FIFOs and synchronization mechanisms where needed. Hence, the designer does not need to worry about dependencies between threads. This is taken care of automatically by the tool. This ensures that the tool will always generate correct-by-construction parallel code. MPMH provides the optional mechanism of shared variables. Shared variables need explicit synchronization and communication between threads. The tool will check for possible inconsistency in the synchronization for the shared variables. However it cannot guarantee correct-by-construction parallel code for these shared variables. This option is therefore not used in the MNEMEE tool flow.

The MPMH tool is also capable of performing static data optimizations by optimizing the accesses to static data arrays. The tool uses compile-time application knowledge and profiling data to find out which data copies can be made and whether these copies are beneficial. It also determines how the data and the copies have to be mapped onto the various memory layers in a memory hierarchy such that the energy consumption is minimized and/or the performance is maximized. The result of the optimization is a transformed parallelized application with data copies and block transfers explicitly expressed in the source code, automatic handling of synchronization of data, and a mapping of the data and copies to the various memory layers.

E. Optimize Dynamic Memory Management

The dynamic memory management methodology is responsible for deciding where a data type or an individual dynamic variable should be placed on a certain memory layer in the memory hierarchy of the targeted MPSoC. The decision is based on the allocation behavior of all dynamic data in the source code. A profiling report reveals the allocation/deallocation timeline of dynamic objects. This report is analyzed to identify the most frequently used objects. These objects are placed closer to the processors in the memory hierarchy, while other data structures are placed in a higher level of the memory hierarchy.

This step does not finalize the mapping of dynamic objects to the memories. Instead it provides the remainder of the flow with hints about the preferred placement. These hints are used to guide the mapping decisions that are made in step 6.

F. Task Mapping

Two alternative techniques are available in the tool flow to map the parallelized application onto the processors and memories in the MPSoC. The first technique, called scenario-aware mapping, tries to exploit the dynamic behavior of an application in order to save resources. The second technique, called memory-aware mapping, focuses on finding a mapping that minimizes the energy consumption of the system while considering memory requirements of tasks.

1) Scenario-aware mapping

Modern streaming applications are becoming increasingly complex and dynamic. Existing mapping techniques (e.g. [4], [6]) model these applications using relatively simple and static models, such as (homogeneous) synchronous dataflow graphs [3]. These models abstract from the dynamic behavior of an application which may lead to a large overestimation of its resource requirements. The dynamic behavior of an application can be taken into account in a mapping technique by using a scenario-based design approach [2]. In this approach, the dynamic behavior of an application is viewed upon as a collection of different behaviors (scenarios) occurring in some arbitrary order, but each scenario by itself is fairly static and predictable in performance and resource usage. Therefore, resource allocation can be performed for each scenario using existing mapping techniques. However, these mapping techniques can only provide timing guarantees per scenario. They cannot guarantee the timing behavior when switching between scenarios. For many streaming applications it is however important that timing guarantees are provided when switching between scenarios. The scenario-aware mapping technique can provide such guarantees.

The input of the scenario-aware mapping step is a set of synchronous dataflow graphs (i.e., one for each scenario). These graphs are automatically derived from the parallelized application source code. The scenario-aware mapping step allocates processing, memory, and communication resources for all these graphs. The output of this mapping step is a set of mappings that provide different trade-offs between the amounts of processing, memory, and communication resources that are used from the MPSoC. A run-time mechanism (e.g. [5], [8]) can use this set of mappings to adapt the mapping of an application to the available resource in different use-cases. The development of such a run-time mechanism is also studied in the context of the MNEMEE project. At this moment, the run-time library does however not support run-time configuration. Therefore, the MNEMEE tool flow now selects one of the mappings generated by the scenario-aware mapping step (i.e., the mapping which minimizes the memory usage). This mapping is then used by the last step in the tool flow.

2) Memory-aware mapping

The memory-aware mapping tool provides a static assignment of tasks to processors. The focus of the tool lies in the integration of the memory subsystem in the mapping optimization decision. The memory hierarchy and the impact on energy, runtime and communication is often disregarded by other mapping tools. The mapping tool considers the memory requirements of tasks and maps them to available memories in the hierarchy. However, the mapping tool does not decide on the final mapping of memory objects to the memories. It only provides hints to the last step of the flow. The last step finalizes the mapping of data objects to memories.

The memory-aware mapping tool is based on the DOL framework [12]. A multiobjective optimization is implemented which balances the load on the processors and communication channels, and minimizes the overall energy of the system. An evolutionary algorithm based on EXPO [13] and PISA [14] performs this optimization. In a first step, several mapping solutions (genes) are generated. In the next step, an energy and performance evaluation is accomplished. The best solutions are stored and used for the generation of new solutions. These last steps are repeated until a maximum number of generations are reached. At the end a set of pareto optimal solutions is provided. Since the goal of the MNEMEE project is to minimize the energy consumption of the system, the solution with the minimal energy consumption is selected and provided to the next step in the MNEMEE tool flow.

G. *Scratchpad memory allocation*

The scratchpad memory allocation tool enables a system designer to implement an efficient memory allocation for statically allocated data within a very short time. It exploits scratchpad memories that are known for fast memory accesses consuming little energy. By allocating frequently accessed data objects to these memories the runtime and energy consumption of a system can be reduced significantly. Unfortunately, beside the favored properties scratchpad memories have a small size. Therefore, the tool applies a knapsack-based approach [11] to calculate the set of data objects that, once allocated to scratchpad memories, will lead to the most savings.

By using integer linear programming, the scratchpad memory allocation tool can solve this problem optimally in a very short time. Finally, the tool implements the resulting optimal non-overlaying allocation by applying source-to-source transformations. Another major advantage of the tool is, that it operates fully automated, thus a system designer can reduce runtime and energy consumption at the push of a button.

III. INDUSTRIAL APPLICATION

The MNEMEE tool flow can be used to automate the already existing too chains of embedded design industries, by replacing traditional manual techniques. This section presents two examples from companies working on different domains, namely communications and multimedia domain. Both examples demonstrate the integration of the MNEMEE tools into their design flow, emphasizing the automation achieved.

A. *Communications domain*

In the context of the MNEMEE project Intracom Telecom targets the IEEE 802.16e system for broadband wireless communications, for fixed, nomadic and mobile users IEEE 802.16e is a broadband wireless solution that enables convergence of mobile and fixed broadband networks through a common wide area broadband radio access technology and flexible network architecture. IEEE 802.16 and WiMAX are designed as a complementary technology to Wi-Fi and Bluetooth.

The target platform is the Freescale MSC8144 [16], a high-performance multicore DSP device. It includes four extended cores, each one comprising the DSP core with dedicated instruction cache, data cache, memory management unit (MMU), interrupt controller (EPIC) and timers. The complexity of the platform demands large design effort to map a streaming application like 802.16e. The pre-MNEMEE design approach from Intracom can be seen on Figure 2. The application's source code has to be manually partitioned to take advantage of the four DSP cores. Then the processes should be verified so that the partitioning does not break any timing constraints. Finally, manual memory mapping of data objects needs to be performed.

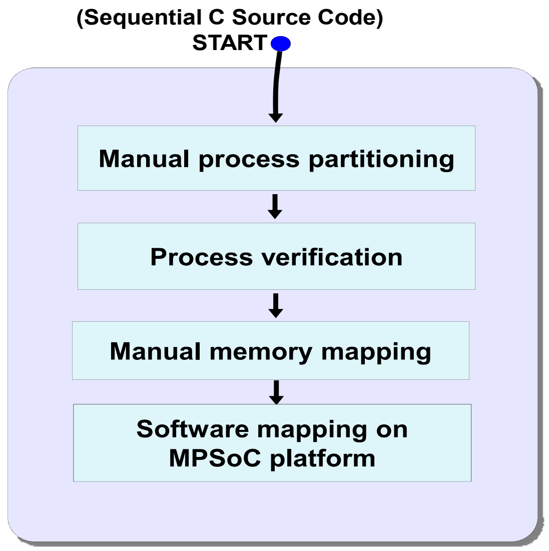
(Sequential C Source Code)
START
Manual process partitioning
Process verification
Manual memory mapping
Software mapping on MPSoC platform

Figure 2 Intracom: pre-MNEMEE design flow

With the help of the MNEMEE tool flow the design flow was changed to that of Figure 3. The automatic process partitioning that is used now to map the processes to cores, provided a boost in the design time saving, while it was possible to achieve acceleration by a factor of up to 2.2. Furthermore, the mapping of data objects is also performed automatically further reducing design effort. The manual operation took ~2 days whereas the automatic process only ~4 hours. A coarse performance difference between this optimised solution and simply mapping everything to DDR, was found to be ~30%. Therefore, usage of the specific tool was considered an efficient solution as it concerns both performance and design time.

B. *Multimedia domain*

The multimedia application is a state of the art low bit rate speech coder based on the enhanced Mixed Excitation Linear Predictive (MELPe) algorithm. This coder constitutes the new NATO standard STANAG 4591 [17][18]. The target platform is the OMAP-L137 [19], a dual core low-power application processor comprising an ARM92EJ-S and a C674x DSP core.

Both processors have their own instruction and data cache, but also share some internal and external memory.

Before integrating the MNEMEE tool flow in the design flow, the required steps were those depicted on Figure 4. The source code has to be split manually between the DSP and the ARM core and then a series of manual optimizations and code rewriting takes place. Critical functions that do not meet the necessary constraints are written in assembly code.

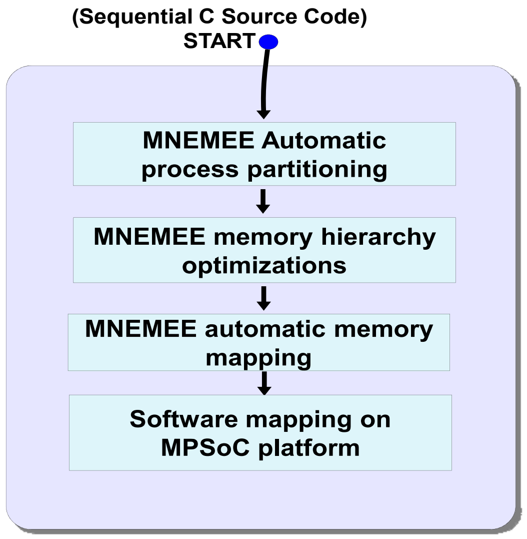
(Sequential C Source Code)
START
MNEMEE Automatic process partitioning
MNEMEE memory hierarchy optimizations
MNEMEE automatic memory mapping
Software mapping on MPSoC platform

Figure 3 Intracom: after integrating the MNEMEE design flow

C. *Multimedia domain*

The multimedia application is a state of the art low bit rate speech coder based on the enhanced Mixed Excitation Linear Predictive (MELPe) algorithm. This coder constitutes the new NATO standard STANAG 4591 [17][18]. The target platform is the OMAP-L137 [19], a dual core low-power application processor comprising an ARM92EJ-S and a C674x DSP core. Both processors have their own instruction and data cache, but also share some internal and external memory.

Before integrating the MNEMEE tool flow in the design flow, the required steps were those depicted on Figure 4. The source code has to be split manually between the DSP and the ARM core and then a series of manual optimizations and code rewriting takes place. Critical functions that do not meet the necessary constraints are written in assembly code.

Integrating the multimedia application on the OMAP-L137 platform led us to face two main challenges: (1) how to efficiently split the code between the two processors and (2) how to make the most of the four available memory levels. Manually addressing these challenges will require a lot of design and integration efforts. The different optimisation tasks depicted in Figure 4 have an impact on each other thus leading to an iterative and error prone optimisation process. Moreover the optimal solution could hardly be obtained manually. Using the MNEMEE tool flow, every step is performed and linked to the next in an automated way. This new design flow is given in Figure 5. The parallelism of the application is first exploited to split the code between the two processors. Even if the MELPe algorithm is mostly sequential, the tools are expected to provide low-grain parallelism optimization which could hardly be obtained manually. The static and dynamic data are then allocated in memories. Finally the communications between processors and data transfers are handled. Defining which memory should host which data is hard to do without tools that can provide information about data usage at run-time. The MNEMEE tools do provide such information, but they also use them to give an optimised mapping and the associated source-code. Furthermore, as most of the tools perform source-to-source optimization, the output source code can be viewed and analyzed.

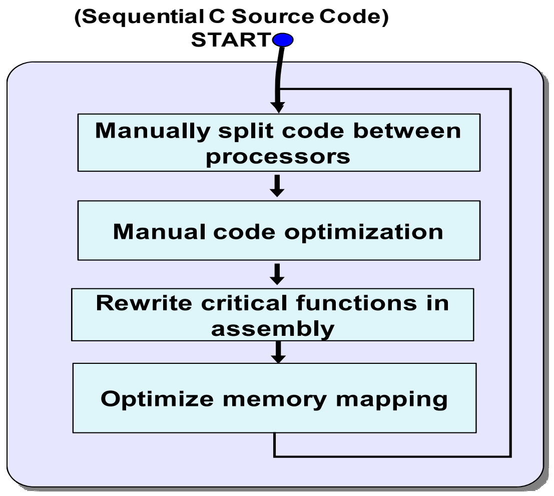
(Sequential C Source Code)
START
Manually split code between processors
Manual code optimization
Rewrite critical functions in assembly
Optimize memory mapping

Figure 4 Thales: pre-MNEMEE design flow

The main advantages of using the MNEMEE tool flow is that each step is optimized in two ways: (1) the tools can generate and evaluate various alternative solutions, which can hardly be done by hand; and (2) the tools can work simultaneously on multiple optimization targets – execution time, energy consumption or memory footprint – to find out the best solution. Of course, some parts of the source code may still require manual optimization, but it should be far less than with a full manual optimisation process.

It should also be noted that if the application needs to be modified or completed with additional functionalities, the use of an automated tool flow such as the MNEMEE tool flow will greatly facilitate the re-factoring process.

IV. Conclusions

The complexity of novel embedded systems is increasing rapidly. These systems combine many different streaming applications in a single system. To meet the processing and memory requirements of these applications, multiprocessors systems-on-chip with a memory hierarchy must be used. The complexity of these architectures and applications make the design of these systems very challenging. A decreasing time-to-market and the need to differentiate products add to this design challenge. Manual optimization and mapping of the application source code is becoming prohibitively slow. Therefore, a structured methodology and automated tools are needed to map the application source code onto the target hardware. The MNEMEE project provides a set of techniques that fills the need for automation in the respective industry's design flows. In this paper, the MNEMEE techniques have been presented, along with two real-world industrial examples, demonstrating the applicability and the gains that are possible by their exploitation.

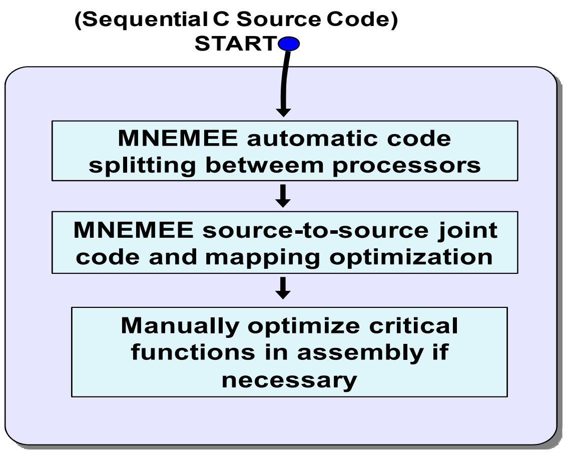
(Sequential C Source Code)
START
MNEMEE automatic code splitting betweem processors
MNEMEE source-to-source joint code and mapping optimization
Manually optimize critical functions in assembly if necessary

Figure 5 Thales: after integrating the MNEMEE design flow

REFERENCES

[1] MNEMEE Project (IST-216224) - http://www.mnemee.org/

[2] S. Gheorghita, M. Palkovic, J. Hamers, A. van de Cappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. van de Putte, and K. D. Bosschere. System-scenario-based design of dynamic embedded systems. ACM Transactions on Design Automation of Electronic Systems 14, 1 (January 2009), p. 1–45.

[3] E. Lee and D. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. IEEE Transactions on Computers 36, 1 (January 1987), p. 24–35.

[4] O. Moreira, J.-D. Mol, M. Belooij, and J. van Meerbergen. Multiprocessor resource allocation for hard-real-time streaming with a dynamic job-mix. In 11th Real Time and Embedded Technology and Applications Symposium, RTAS 05, Proceedings (March 2005), IEEE, p. 332–341.

[5] H. Shojaei, A. Ghamarian, T. Basten, M. Geilen, S. Stuijk, and R. Hoes. A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for CMP run-time management. In 46th Design Automation Conference, DAC 09, Proceedings (June 2009), ACM, p. 917–922.

[6] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In 44th Design Automation Conference, DAC 07, Proceedings (June 2007), ACM, p. 777–782.

[7] Y. Iosifidis, A. Mallik, S. Mamagkakis, E. De Greef, A. Bartzas, D. Soudris, F. Catthoor. A Framework for Automatic Parallelization, Static and Dynamic Memory Optimization in MPSoC Platforms. In the Design Automation Conference, DAC 10, Proceedings (June 2010), ACM.

[8] Ch. Ykman-Couvreur, V. Nollet, F. Catthoor, H. Corporaal. Fast Multi-Dimension Multi-Choice Knapsack Heuristic for MP-SoC Run-Time Management. In International Symposium on SoC, Proceedings (2006), IEEE, p. 1–4.

[9] R. Pyka, F. Klein, P. Marwedel and S. Mamagkakis. Versatile System-level Memory-aware Platform Description Approach for embedded MPSoCs, LCTES 2010, April 2010.

[10] M. Girkar and C. D. Polychronopoulos. The hierarchical task graph as a universal intermediate representation. International Journal of Parallel Programming, 22(5):519–551, 1994.

[11] S. Steinke, L.Wehmeyer, B. Lee, and P. Marwedel. Assigning program and data objects to scratchpad for energy reduction. In Design, Automation and Test in Europe, DATE 02, Proceedings, p. 409, IEEE, 2002.

[12] L. Thiele, I. Bacivarov, W. Haid, K. Huang. Mapping Applications to Tiled Multiprocessor Embedded Systems. In Application of Concurrency to System Design, ACSD 07, Proceedings (July 2007), IEEE, 2007.

[13] L. Thiele, S. Chakraborty, M. Gries, M., and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In 39th Annual Design Automation Conference, DAC 02, Proceedings (June 2002), ACM, p. 880-885.

[14] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA – A Platform and Programming Language Independent Interface for Search Algorithms. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and . Thiele, editors, *Evolutionary Multi-Criterion Optimization (EMO 2003)*, volume 2632/2003 of *LNCS*, pages 494–508. Springer-Verlag Heidelberg, 2003.

[15] C. Baloukas, J.L. Risco-Martin, D. Atienza, C. Poucet, L. Papadopoulos, S. Mamagkakis, D. Soudris, J. Ignacio Hidalgo, F. Catthoor, J. and Lanchares. Optimization methodology of dynamic data structures based on genetic algorithms for multimedia embedded systems. J. Syst. Softw. 82, 4 (April 2009), p. 590-602.M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[16] MSC8144 Reference Manual,
http://www.freescale.com/files/dsp/doc/ref_manual/MSC8144RM.pdf

[17] "The 600 bits/s, 1200 bits/s and 2400 bits/s NATO interoperable narrow band voice coder", NATO Standard STANAG No. 4591 Edition Y (amendment W) Ratification Draft 1, January 2006.

[18] G. Guilmin, F. Capman, B. Ravera and F. Chartier. "New NATO STANAG narrow band voice coder at 600 bits/s", Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, Toulouse, May 2006.

[19] OMAP-L137 Reference Manual,