Versatile System-level Memory-aware Platform Description Approach for embedded MPSoCs

Robert Pyka
Informatik Centrum Dortmund ICD,
Dortmund, Germany
pyka@icd.de

Felipe Klein
Computer Systems Laboratory,
University of Campinas, Brazil
klein@ic.unicamp.br

Peter Marwedel
Faculty of Computer Science
12,Technische Universität Dortmund,
Germany
peter.marwedel@tu-dortmund.de

Stylianos Mamagkakis
Interuniversity Micro-electronics Center IMEC, Leuven, Belgium
mamagka@imec.be

Abstract

In this paper, we present a novel system modeling language which targets primarily the development of source-level multiprocessor memory aware optimizations. In contrast to previous system modeling approaches this approach tries to model the whole system and especially the memory hierarchy in a structural and semantically accessible way. Previous approaches primarily support generation of simulators or retargetable code selectors and thus concentrate on pure behavioral models or describe only the processor instruction set in a semantically accessible way, A simple, database-like, interface is offered to the optimization developer, which in conjunction with the MACCv2 framework enables rapid development of source-level architecture independent optimizations.

Categories and Subject Descriptors B3.3 [*Memory Structures*]: Performance Analysis and Design Aids; D.2.2 [*Design Tools and Techniques*]: Computer-aided software engineering; C0 [*General*]: System architectures; C4 [*Performance of Systems*]: Modeling techniques

General Terms Algorithms, Languages, Design, Performance

Keywords Architecture description, component, channel, configuration, definition, energy models, framework

1. Introduction

In the past years, a huge amount of work has been done in the area of optimized program compilation. Especially lots of effort has been spent for optimizations which take particular properties of the memory subsystem into account. Nevertheless, the usual approach in that area is to use optimization-specific memory configuration descriptions, or even incorporate this information into the optimizer code. On the one side, this allows for a compact and simple memory description, on the other side combining various optimizations becomes difficult, porting them to other architectures even more. Furthermore, a consistent memory model would be beneficial for ensuring comparable results.

Among all the system modeling languages developed recently, some of them incorporate a memory model. In almost all cases, the target application is generation of simulators, transformation to hardware description languages or code generator generation, which does not fit well with source-level optimizations. The first class of languages implement a behavioral model, which can be transformed into executable code on the host platform and thus supporting simulation. Transformations to HDLs require primarily structural information. A fixed set of properties per component is required to interpret the description and translate it to the lower level HDL. Furthermore, due to the fact, that a complete and synthesizable hardware description has to be generated, a high level of detailed information is required right from the beginning of the system specification. In contrast to this, for memory aware optimizations usually a quite abstract system model is sufficient. Finally, models for code generator generators concentrate on the internal structure and instruction set of the processor.

Therefore, we propose a combined structural and semantical description at system level. On the one side, it offers a structural modeling approach for the system designer. But on the other side, a database-like interface is presented to the optimizations designer. It supports model refinement and requires only limited effort for the initial abstract system model.

The structure of this paper is as follows: It starts with an overview of requirements imposed by the intended application field. The next section provides references to related work and puts this approach in relation to these publications. Section 4 describes details of the system model and concentrates on the key feature of this system description, which allows the database-like access to the system model properties. In Section 5 the MaCCv2 framework is introduced. MaCCv2 provides the surrounding infrastructure which incorporates our system description model. Afterwards, an application example is shown in Section 6. This example demonstrates an implementation of the static scratchpad allocation technique. This optimization technique was implemented in a fully architecture independent way.

In the following results section we are going to show that the completely different method of accessing system and memory hierarchy properties does provide same quality results as the simulation based approach in the modeled MPARM SoC.

The paper closes with a conclusion. As a result, we can summarize that our approach qualifies as a viable way for rapid development of memory aware source-level optimization techniques which can adopt to different architectures without the need of developer interaction.

2. Requirements

The memory description proposed in this paper has to fulfill a set of requirements. These requirements were chosen according to the needs of current memory optimization techniques and available architectures. Furthermore, the memory description should be sufficiently generic to be applicable to future memory architectures. For example a fixed set of memory component types would be an unacceptable limitation in this context. Therefore, the proposed modeling approach provides in addition to the predefined components and channels an open API to extend the set of available building blocks. Finally, it has to permit the design of a memory description in a top down approach in terms of complexity and precision, without losing compatibility to optimization techniques which do not require such a high level of details. The refinement of memory models includes the possibility to add aspects of interest as required. The remaining model and the API should not be affected by the amount of aspects (i.e. latencies, energy values etc.) provided by the model for a particular memory device type. Referring to Section 4 this approach does not impose limitations on number or type of provided component properties.

Since the memory description should be accessible from within the optimization techniques, an API has to be provided. An object-oriented representation of the memory model – as used in this approach – fits best the requirements of current programming style. In conjunction with the previously stated requirement for extensibility, this results in a challenging task, which was successfully tackled through the integration into the MaCCv2 framework, shortly described in Section 5. Previous experiences show that beside pure simulation-oriented models a query-oriented database-like interface is also required. Especially for fully compile-time based optimizations, this is the key to a portable implementation. Current MPSoCs are designed in various configurations: Starting from simple shared memory systems consisting of a set of equal computing elements, up to heterogeneous systems with separated address spaces. This requires a model which goes beyond the description of memory devices but also incorporates an interconnection model.

Finally, a graphical user interface which allows for an easy design and modification of memory descriptions would be beneficial for a rapid development process.

3. Related Work

According to the classification of system modeling languages along the abstraction level as shown in Figure 1, the description language presented in this paper can be assigned to the PMS level. Processor-Memory-Switch models were introduced by Bell and Newell in [5]. They describe the system at an abstract level where the main building blocks are the system components (i.e. processors and memories) and the interconnection between them.

Various architecture description languages (ADLs) have been developed in recent years. One comprehensive approach to classify them is by the main application target. A first application target is the automatic transformation into a hardware description language. This helps in automating hardware development as well as in shortening the development cycle, where precise simulators can be derived in advance from the HDL representation of the system. A second main application target is the automatic generation of development tools like compilers, assemblers and linkers. Both application targets impose almost disjoint requirements on the ADL. On the one side, hardware description requires precise structural information, on the other side, automatic tool generation requires a behavioral and semantic description of the system.

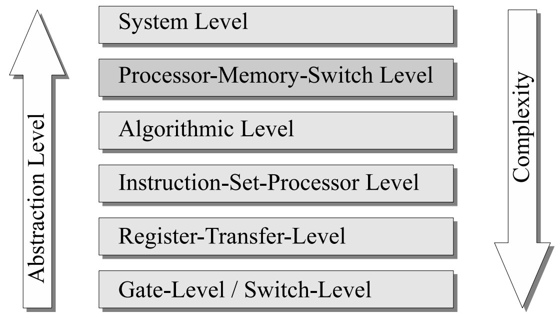

Abstraction Level
System Level
Processor-Memory-Switch Level
Algorithmic Level
Instruction-Set-Processor Level
Register-Transfer-Level
Gate-Level / Switch-Level
Complexity

Figure 1. System modeling levels

LISA [12] is an ADL which targets primarily at the automatic generation of application-specific hardware and corresponding simulators and low-level tools. Primary architectural targets are signal processing and generic irregular single processor architectures. The language has been extended later towards automatic compiler generation. To accomplish this task, an additional semantic instruction set model has been added [7]. Since the main target of LISA is the cycle-true description of the DSP, neither a sophisticated system model nor detailed memory models exist. The timing model integrated into LISA concentrates on the specification of the pipeline behavior. There is no energy model incorporated into LISA.

Another ADL developed recently is ArchC [4]. ArchC was designed to support processor architecture description. While the language has evolved also the possibility to design memory hierarchies has been added. Similar to LISA, ArchC also covers the structural and behavioral view of a system model. Since ArchC uses the SystemC language, which provides extensions to C++ for description of timing and concurrency, ArchC models are described in C++ code. On the positive side, this opens a possibility to describe a large scale of different systems, on the negative side, it is really hard to extract any semantic meaning from the model, once it comes to other tasks than simulator generation. The SystemC language offers all the expression possibilities of the C++ language. This is perfect for simulation, since this allows for compiled simulation which is the fastest simulation method available. But even the task of extracting timing information out of such a model in advance without imposing any restrictions to the modeling style is a almost intractable problem. Closely related to ArchC is PDesigner [2]. Basically, it is a graphical editor which can be used for intuitive component based development of ArchC system models.

Another group of system modeling languages are specialized system descriptions targeting mapping of applications on MPSoCs. As examples, references to the ADL used in DEADALUS [8] or the hardware platform description in the CIC based retargetable parallel programming framework for MPSoC [14] are given. In general, these system descriptions are from the structural point of view similar to the one presented in this paper. Nevertheless, significant differences can be observed when it comes to the annotation of component and channel properties. These approaches suffer from the fact that changes to the properties of one component need to be annotated in several places of the system description. An example for such an annotation would be the energy consumption of a memory component. In contrast to our approach, a change of such value will affect the per-access energy values annotated to processing elements. In general this will require user interaction, and in-depth system knowledge to precompute these values.

EXPRESSION has been developed in the late 90's [11]. EXPRESSION aims primarily at automatic generation of software development tools. The motivation for this language was faster design space exploration on a single processor of the SoC. To accomplish this task, EXPRESSION describes the system in a structural and behavioral way. In contrast to the previous languages, EXPRESSION offers an explicit memory model. There is a fixed set of parameters which can be used to describe properties of memories available to the processor. On the downside of this ADL, there is no method to convert the description into an HDL for automatic generation of processor hardware.

A slightly different application scenario is specified for the TDL [13] language. The primary goal of this language is to support retargetable postpass optimizations at assembly level. This is what comes closest to the approach presented in this paper. TDL includes a structural description of the resources present in the system. This includes memories and corresponding cache hierarchies. Furthermore, behavioral description of the instruction set is the second key part of this language. Nevertheless, there are significant differences to our ADL. The semantical information in the resource section does not allow for modeling structural dependencies between memories, furthermore only a single processor based memory hierarchies can be described.

Finally, there exists a vast variety of other ADLs. As examples, references to hardware related ADLs like nML [9], ISDL [10] and MIMOLA [17] are given. All of them do not focus on development of memory hierarchy aware source code transformations. The common goals are simulation, HDL extraction or compiler generation. None of them satisfies all the requirements imposed in the previous section. In a broader scope, the set of software related ADLs also has to be take into account. Examples in this class of ADLs are Wright [3] or Darwin [16]. In contrast to this approach, these ADLs concentrate on description of the architecture of the application instead of the platform the application is being executed on.

4. Model

The novel architecture description model presented in this paper can be classified as a structural PMS model. In contrast to previous ADLs where a detailed model has to be developed in advance, in this approach only high-level structural information has to be provided for a new architecture. Once tools require more details, these can be added to the model without losing backward compatibility. Furthermore, the main application target of this ADL is the development of code optimizations at source-level which leverage the knowledge of the memory hierarchy. This implies the accessibility of the system properties while the optimizations are being executed. In contrast to ADLs with simulation as the main target, system descriptions can contain pure behavioral parts, which are passed through the generator tool without any further semantical analysis. ADLs designed for compiler and development tool generation contain semantical information, but this is usually limited to the description of the instruction set.

The ADL proposed in this paper provides a structural model of the memory hierarchy, enriched with semantical information. This model allows for a database-like access to the system description, where the optimizations can query system properties which are required to apply them on different architectures. The approach presented in this paper has been developed as part of the MaCCv2 compiler framework. Therefore, it smoothly integrates within this framework. The memory description is part of the complete system and application description. The memory description presented here is defined as an object oriented C++ API provided to the optimization or analysis tool. The actual format the memory description is stored in, is not part of this specification. Due to the strong relation between the MaCCv2 framework and the ICD-C compiler tools [1], the current implementation uses an XML based storage method. ICD-C is a compiler development platform. It represents C-code as a source-level abstract syntax tree and offers a code selector interface and a wide range of transformation and optimization techniques.

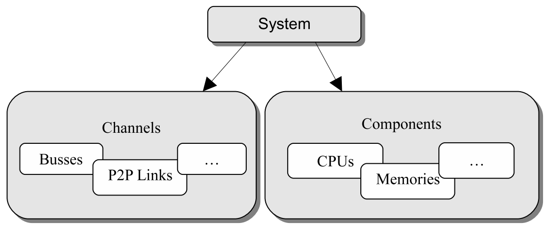


Figure 2. System Model.

Basically, the system description consists of a set of channels and components. Figure 2 depicts the top-level structure of a system description. Channels represent the interconnections between components. Examples are: Usual busses, point-to-point links or some other kind of abstract direct connections. The last one could be useful in the case the processor register file is also modeled as a memory component. Registers are part of the processor architecture, but they have similar restrictions in terms of maximum number of parallel accesses and delays as usual memories. Therefore, it is a valid model to separate the processing unit and the register file and connect them through a direct link. Components are self-contained parts of the system which have to exchange data with each other. The most common types of components are processing units and various kinds of memories. Additional components like ASIPs or DMA units may be included in the model. In most cases, components will map to particular parts of the on-chip hardware. Nevertheless, this is not required. Also, hard disks or even software-controlled data exchange between processors could be modeled as a virtual memory device.

From the structural point of view, the system models described in our modeling language are similar to the one used in languages for simulator modeling. Therefore, a translation to another system description language is possible. For example, from our system model a SystemC skeleton can be generated. Due to the high-level representation and the missing behavioral information, it is not possible to generate a fully specified SystemC model. Nevertheless, if future tools would require automatic translation to SystemC it is possible to attach behavioral information as user extended data to the system model. A particular translation tool which can handle this data could generate full fledged SystemC models.

A typical architectural example is shown in Figure 3. Multiple processing units with a local memory and dedicated caches are connected to a shared main memory. The basic blocks of this structural view are components, which are equipped with ports that are connected to channels. Components may initiate requests or may be a target for requests initiated by others. Structural memory hierarchies may be modeled as shown for the cache components. In that case, the component has multiple ports which are connected to different channels. On the one side, it is the target for requests on the other side, it issues requests to the next level.

Going deeper into the details, Figure 4 shows the concept of address space translation and mappings. Each component has a set of address spaces. They may have different properties. For

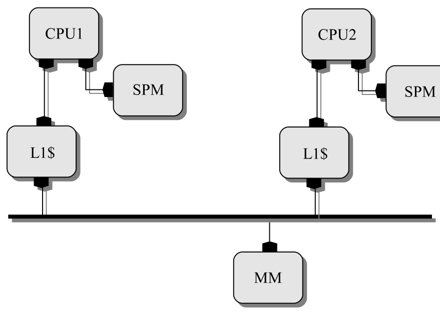


Figure 3. Interconnection Model.

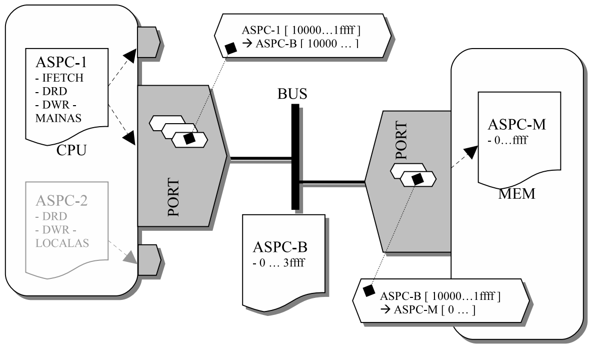


Figure 4. Mapping Model.

example, a processing unit having a Harvard architecture will have at least two address spaces; one for instructions and a second one for data. Another example could be a processing unit which has a local memory that is not part of the main address space and has to be accessed through auxiliary instructions.

Address spaces are not limited to be part of access initiating components like processing units, but also target components have at least one address space. In the case of a simple memory, this would be a plain range starting at 0 and going up to the size of this memory.

Besides components, channels have also a set of address spaces assigned to them. They represent which kinds of accesses may be transported over that particular channel. The mapping between address spaces of components and channels is performed at the connection port. Each port has a set of mapping rules which translate between component address spaces and channel address spaces and vice versa. It is possible to have mapping rules for the same address space at different ports. This allows for modeling systems which have a bus based interface to the main memory and a fast dedicated interface to a local scratchpad memory, while both are located in the same address space.

The memory description is presented to the application as an object graph with nodes representing components and channels and edges representing connections. Traversing this graph allows the application to collect structural properties of a system. Each object in this graph may have user defined attributes assigned to it. This resembles the often used approach of direct annotation of required component properties, present in other modeling languages. This is feasible for attributes which are solely related to a particular component. An example would be the number of banks a memory consist of. An example of less suitable value type would be the per-access energy consumption annotated to a processing unit. This value depends on several system properties. On the one side,

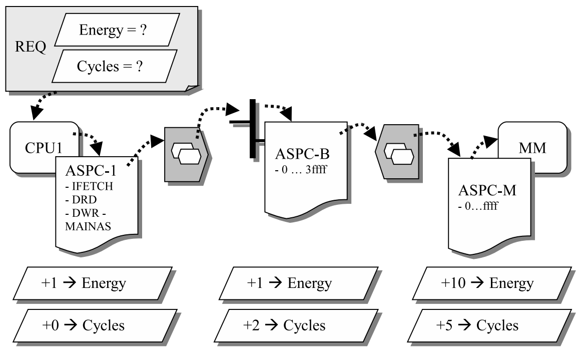


Figure 5. Routing Model.

in general there will be no single value, but a set of values which depend on the number of memories present in the system. On the other side, when performing design space exploration, the update of these values is a non-trivial task. It requires user interaction and in-depth system knowledge. When exploring the effects of changes in the energy consumption of a particular memory, in general several values across the whole system description may need an update. To avoid this error-prone overhead, the proposed model offers a generic mechanism to collect component and chanel properties. Its basic item is the access-request. It is similar to a database query. The application (i.e. the optimization) issues an access to a component object—usually a processing unit—and the implementation of the model ensures to route the query through the structural description. After the access has reached another component, the target component processes the request and fills in the information it can provide according to the request. Figure 5 shows the access processing scheme built into this memory description model. Which kind of information is being returned depends on the actual implementation of this component. Usually, energy values, latencies or data values will be returned. In the case a memory device can provide its data content for a particular access, this query based approach could resemble also simulation based approaches. But in contrast to them, not only the actual value is being returned, but also information about which kind of additional accesses were required to get this value. For example, an access directed to a cache will provide information about the accesses to the next level caused by a cache miss.

Each request has a basic set of properties which are necessary to route the request through the model:

- Address range.
- Number of bytes.
- Alignment.
- Access mode (e.g. ”IFETCH”).

These properties are required in each request to ensure that the model can resemble in the routing algorithm the paths the request would have taken in an actual hardware instance.

Further, an arbitrary set of requested aspects may be attached to the access request. In most cases this will be:

- Energy values
- Latencies / Cycle values
- Throughput
- Data values

The implementation of an optimization technique issues an access request to the component which would initiate the access in a physical environment. The access request is targeted for an address range in a particular address space and access mode (i.e. ”IFETCH”). In the first step, the port to which the access should be routed is looked up. As mentioned before, each port has a set of mapping rules. Each rule states from which address space to which one the access can be mapped, and for which address range in the ingress address space it is valid for. After an applicable rule has been found, this port is used for further routing. Basically, the access is being translated and issued to the channel the port is connected to. The next step forwards the access to the target port. The procedure is the same as described before for the component. A port with a suitable mapping rule is being searched. After mapping the access to the target component address space, the access request is being forwarded to that component for service.

Throughout that request chain, three places exist where requested properties may be updated; namely at the initiating component, the channel and the targeted component. If at one place, particular properties can not be provided by the implementation, this is visible in the request, too. Assuming the case that each component and channel on the path can provide a particular aspect value, an update operation has to be performed at each place. For cumulative values like cycle counts the update operation will reduces into an addition. Other types of aspects may need other update operations (i.e. for throughput this could be the computation of a minimum value). Since computation of aspect values is performed in reverse order, starting at the target component, even more complex update functions which need knowledge of other aspect values from partial results may be performed. A common example for this kind of complex update functions would be the computation of the energy consumption of a processing unit. A practical example for this complex update function occurs in the energy model for the MPARM processor, used in the evaluation part of this paper. This processor’s energy model provides two energy states; namely running and idling. When accessing a memory the processor consumes some energy to setup the access and than idles until the resulting value is available, still consuming energy. Therefore, the computation has to take into account the number of cycles the access will take. Since the values are computed bottom up, the amount of cycles the memory and interconnect requires to perform a particular access has already been computed and is known to the update function at the processor level.

In general, multiple target components will be addressed in an access request. For example, in a system consisting of a main memory and a scratchpad memory, an access request covering a sufficient wide address range could target both memories. Therefore, the reply contains information about every path a request has taken. While the request is posted, the user can choose which path should be the primary result. Currently between best-case or worst-case answers can be choosen. In the example, one path would target the scratchpad, the other would target the main memory. In general, both paths will result in different values for each aspect. Scratchpads require less cycles and less energy than the main memory. Therefore, aspect values of scratchpad would be the best-case answer, while the aspect values of main memory would be the worst-case answer.

Furthermore, in general an access request may exceed the maximal width per access of involved components and channels. Therefore, an access request can be split into an access sequence while being routed through the system description. The model ensures proper accumulation of values in that case. Nevertheless, each step and each transformation involved in the access processing is recorded. Therefore, besides the accumulated values, detailed analysis of the effects which lead to the result can be performed.

The detailed access processing history allows to compute precise values for components with sophisticated aspect models. An example for this kind of components are DRAM memories. In general the access times and energy consumption of DRAM memories vary depending on the sequence of previously performed accesses. At the first view, this may contradict the static approach proposed in this paper. This approach requires the computation of aspect values to be stateless across multiple queries, otherwise the resulting values would depend on the control flow of the optimization algorithm performing the queries. Obviously, resulting in completely unreliable values. To support this kind of memories, the approach presented here exploits the possibility to model explicit access sequences. Either the user (i.e. the optimization algorithm) may place queries which do not consist of a single access, but specify an access sequence. Alternatively, a sequence may be build implicitly, as described in previous paragraph. This would enable a DRAM memory to treat subsequent accesses differently computing more precise value. Furthermore, the knowledge of the purpose of the query may be taken into account to improve the result quality. If the optimization algorithm needs conservative values, it may ask for those worst case results. This case is shown in the Example–Section 6. Another option would be to place a query for the best case values.

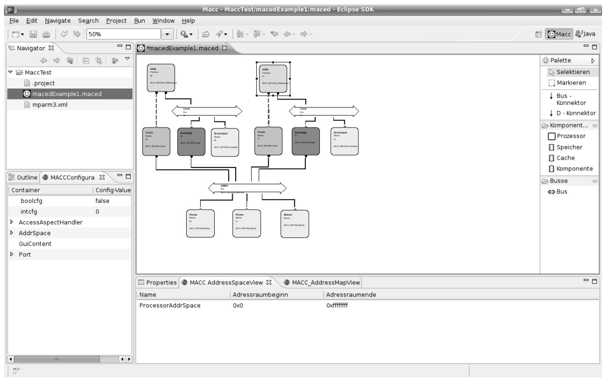
Besides DRAMs, other memory types benefit from this approach. Caches will in general take sequences into account. Furthermore, specialized memory types optimized for particular access patterns, like video memories optimized for frame-buffer accesses or memories organized in multiple banks, where accesses can be interleaved, will have the option to provide values with respect to an access sequence.

5. Framework

The system description model is tightly integrated into the MaCCv2 framework. MaCCv2 is intended as a platform for the rapid development of source-level memory aware optimization techniques. The key features are:

- The integration of a scalable structural PMS-level system model.
- Database-like interface to the system model. The properties of the system model are accessible to the optimization developer at runtime.
- Integration of the application code into system model. The ICD-C code representation is used to attach program code to processing units.
- A graphical user interface for system modeling. An Eclipse based plug-in has been developed, which allows to modify the system model from within this widely used IDE.
- A template for optimizations and analysis tool development is provided. This enables the development of interchangeable tools which can be combined and reused in further optimization techniques or upcoming architectures.
- Interface to backend tools (i.e. compilers, linkers, simulators).
- Configuration and event notification interfaces.

The first two features are an integral part of the system model and the corresponding API. They have been presented in the previous section. Furthermore, the structural model can be used to solve the problem of representing program code assignment to particular processors. Among other properties, each processing unit has one or more ICD-C based C program code representations assigned to it. This code is stored in an ICD-C based abstract syntax tree. Any transformation or optimization available for ICD-C can be performed on program code stored in a MACC system model.



Container	Config-Value
boolcfg	false
intcfg	0
AccessAspectHandler	
AddrSpace	
GuiContent	
Port	

Name	Adressraumbeginn	Adressraumende
ProcessorAddrSpace	0x0	0xffffffff

Figure 6. Graphical user interface.

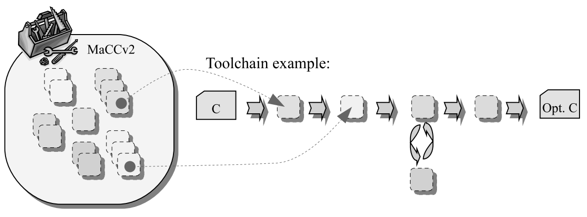


Figure 7. MaCCv2 example optimization chain.

In the context of the MACCv2 framework, a graphical editor has been integrated into the Eclipse IDE. Figure 6 shows the editor. It can be used to define new system models or alter exiting models (e.g. change the scratchpad size or configuration parameters). Furthermore, the framework provides methods for ordered execution of transformation, analysis and optimization techniques. Each implementation may specify requirements which task has to be performed first. The usual application scenario for this feature is shown in Figure 7. The blocks in this figure represent transformation, analysis and optimization tasks which have been combined to form a source-level optimizer. Finally, a rich set of methods is provided which supports rapid development of optimization techniques and provides an abstraction layer which allows for integration of those techniques into a comprehensive graphical user interface.

6. Application Example

The example shows a typical approach of implementing a static scratchpad allocation algorithm in a fully architecture independent way using the memory model and the MACCv2 framework.

To demonstrate the allocation strategy, we choose a system model which describes the MPARM [15] system. Figure 8 shows the model with the default configuration of 4 cores. According to the original setup, caches have been disabled.

The static scratchpad allocation used for demonstration purposes was presented by Steinke et al. in [18]. Further extension toward multiprocessor systems has been done by Verma et al. [19]. Basically the approach solves a knapsack problem for each processor separately using integer linear programming. Without loss of generality, our example targets energy reduction as the objective. A set of input parameters is required per processor to formulate the ILP:

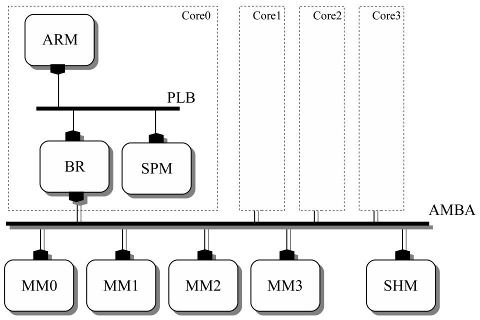


Figure 8. MPARM Model.

- A set of code or data items which can be placed independently in the main memory or the scratchpad memory. These items are identified as memory objects in literature. In our example global variables (i.e. scalars and arrays) and functions are used.
- Size of each memory object.
- Access counts to each memory object.
- Scratchpad size as the limit for the knapsack.
- Energy consumption per main memory access.
- Energy consumption per scratchpad memory access.

The set of movable items can be identified easily in the abstract syntax tree of the C program provided by ICD-C [1]. The processor nodes in the system model incorporate the ICD-C based representation of the program code which is going to be executed at this processor. Iterating over the global symbol table provides the names and references to these objects. Access counts to each of these memory objects are generated via profiling in MACCv2. From the point of view of this example optimization, the access count information is inherent to the system description. Each MACCv2 based optimization can specify its requirements, which processing and analysis steps have to be performed first. In our case, we would specify the requirement to run the access count generator and size estimator. The former in turn would employ profiling and simulation to collect this values. Once static analysis methods become more sophisticated they may replace the profiling step.

In the context of this paper, more interesting are the steps required to find the architectural properties. Determining them is as easy as looking up some predefined values, but with the confidence of always getting precise up to date values without the need for manual adaption to new platforms. Part of the MACCv2 framework is an address space analyzer, which provides a mapping list of address ranges in the address spaces of an initiator component (i.e. an ARM processor) and the final target component. The analysis is performed by default across all memory hierarchy levels, therefore in the mapping the final target component (i.e. Main memory) will be visible.

For the architecture shown in Figure 8, a mapping would look as follows for the first processor:

```
ARM0.[0x0-0xbfffff] -> MM0.[0x0-0xbfffff]
ARM0.[0x19000000-0x190fffff] -> SHM.[0x0-0xfffff]
ARM0.[0x22000000-0x22002fff] -> SPM0.[0x0-0x2fff]
```

Similar mappings exist for other processors. According to the object class in the MACCv2 representation of each target component in that list, the scratchpad memory can be identified easily (i.e. the type of that memory is a derived class of MACC_Scratchpad). In a second step, its size in the processor address space has to be retrieved. Simple arithmetic operations on the address ranges are necessary to compute a size of 12k bytes for our example architecture.

The energy values which are required to compute the gain for each object for the knapsack formulation can be retrieved in a similar way. As described in Section 4, the key feature of this model is to provide system properties on a query based interface. Two sets of queries are required for this optimization. The first one would be placed to the address space range of the main memory, the second one to the range of the scratchpad memory. Each set would collect three values; for data read, data write and instruction fetch. The difference of the corresponding values of each set is the gain factor multiplied by the access counts. All the prerequisites for the ILP formulation are present now. Solving the ILP will result in a set of decision variables indicating which memory object has to be placed on the scratchpad. The final step required in this optimization is to translate these decision variables into linker hints, which will be annotated to this particular memory objects directing them either to the scratchpad or the main memory. The optimization can delegate the actual compiling and linking process to MACCv2.

7. Results

For the application example presented in the previous section, the key value directing the ILP solution is the gain which can be achieved per memory object when moving it to the scratchpad memory. Therefore, we have compared the computed gain values of our model to the ones which are achieved by the whole system simulation in MPARM.

The simulation platform consists of the MPARM SoC simulator extended with the memory hierarchy simulator MEMSIM [20]. The system configuration defines one processing tile with a 12k Bytes scratchpad and a 12M Bytes main memory connected via an AMBA-AHB bus. The energy and latency values for these memories are configured according to the reference design:

- Scratchpad memory read: 0.0241275682 nJ / 0 WS
- Scratchpad memory write: 0.0080780647 nJ / 0 WS
- Main memory read: 10.6813104793 nJ / 10 WS
- Main memory write: 1.0667890999 nJ / 10 WS

The benchmark application chosen for this comparison was a chain of matrix operations. In the first step, a matrix $\boldsymbol{A}$ has to be transposed, and in the second step, a second matrix $\boldsymbol{B}$ is added and the result is stored in another matrix $\boldsymbol{C}$.

$$\begin{aligned}\boldsymbol{AT} &= \boldsymbol{A}^T \\ \boldsymbol{C} &= \boldsymbol{AT} + \boldsymbol{B}\end{aligned}$$

All matrix values are 32 bit unsigned integers. Two sets of experiments have been performed, one with 10×10 matrices, another with 30×30 matrices.

For our purposes, this benchmark has the advantage that the access counts to each matrix are fixed and can be determined via profiling. Table 1 shows the access counts for 10×10 matrices which have been determined in advance.

Matrix	Reads	Writes
$\boldsymbol{A}$	100	0
$\boldsymbol{B}$	100	0
$\boldsymbol{C}$	0	100
$\boldsymbol{AT}$	100	100

Table 1. Access counts for 10×10 matrices.

Corresponding values for 30×30 matrices are shown in Table 2.

Matrix	Reads	Writes
$\boldsymbol{A}$	900	0
$\boldsymbol{B}$	900	0
$\boldsymbol{C}$	0	900
$\boldsymbol{AT}$	900	900

Table 2. Access counts for 30×30 matrices.

For each matrix size, we have run the benchmark with all matrices located in the main memory in the MPARM simulator and retrieved the total energy consumption. This results in an energy value of $27130.266nJ$ for the 10×10 matrix and $231908.203nJ$ for the 30×30 matrix. Afterwards, we have moved each matrix to the scratchpad memory and got total energy reductions according to Table 3. This values are the gain factors which could be used in the ILP formulation presented in previous section.

Matrix	10×10	30×30
$\boldsymbol{A}$	$1122.741nJ$	$10104.672nJ$
$\boldsymbol{B}$	$1122.741nJ$	$10104.672nJ$
$\boldsymbol{C}$	$164.459nJ$	$1480.125nJ$
$\boldsymbol{AT}$	$1287.202nJ$	$11584.812nJ$

Table 3. Measured gain per matrix.

The next step consists of computing these values using our MACC system model without the need for full system simulation. The model has been designed according to the simulator setup. The energy and latency values for memories are setup to the same default values as used for simulation. The interconnection energy computation is based on work done by Bona et al. [6]. The subsequent values of total energy per access were retrieved with four access queries. Two queries directed to the scratchpad memory and two to the main memory, each one for a 4 bytes read and a 4 bytes write access correspond to the size of the matrix elements.

A typical query consists of only few lines of code; setup, add aspects, perform query and retrieve value:

```
MACC_Access *acc=new MACC_SingleAccess(...);

acc->addData(ASPECT_ENERGY);
acc->addData(ASPECT_CYCLES);
acc->queryAccess(AS_WORSTCASE,ASPECT_ENERGY);

val=acc->getValue(ASPECT_ENERGY);
delete acc;
```

Matrix	10×10	30×30
$\boldsymbol{A}$	$1120.457nJ$	$10084.114nJ$
$\boldsymbol{B}$	$1120.457nJ$	$10084.114nJ$
$\boldsymbol{C}$	$164.238nJ$	$1478.143nJ$
$\boldsymbol{AT}$	$1284.695nJ$	$11562.258nJ$

Table 4. MACCv2 computed gain per matrix.

Via these queries, we have retrieved total per access energy values of $0.0791284nJ$ per scratchpad read, $0.0630784nJ$ per scratchpad write, $11.2837nJ$ per main memory read and finally $1.70546nJ$ per main memory write. With these values and the access counts, we were able to compute the gain values for each matrix. The values can be found in Table 4. Comparing these values to the ones generated via full system simulation, we can conclude that the system modeling approach presented in this paper has the capability to provide very accurate information about the energy consumption. The computed results were all less than 0.21% off compared to the actual values. Furthermore, the results are stable along the increased number of accesses, even for the bigger matrices with nine times higher access counts, the divergence remains almost the same.

Matrix	Divergence
$\boldsymbol{A}$	0.203%
$\boldsymbol{B}$	0.203%
$\boldsymbol{C}$	0.134%
$\boldsymbol{AT}$	0.195%

Table 5. Relative divergence to simulation results.

8. Conclusion

We have presented a novel system modeling approach with the primary target in support for developing architecture independent source code optimizations, which can still take advantage of architectural properties to achieve higher gains. Especially the knowledge of properties of the memory subsystem, enables a wide range of code optimization opportunities. Together with the query-based interface of the system model, which is capable of providing instant results, the memory aware optimization techniques can be guided without the need for time consuming simulation. Furthermore, the implementation of these technique can be fully architecture independent. Required architecture information is retrieved from the system model, dependencies on other processing steps may be specified in a generic way, without the need to relay on a architecture dependent implementation of those steps.

As shown for the MPARM system model, even with a high level model which consist only of few building blocks, it is possible to achieve quite precise results. We have demonstrated that our model can provide energy values which are less than 0.21% off to the actual values.

The system modeling approach integrates smoothly into MACCv2 compiler framework which offers a comprehensive infrastructure for compiler development and in particular for architecture-independent source-level transformation development.

References

[1] ICD-C Compiler framework. `http://www.icd.de/es/icd-c/icd-c.html`, 2008.

[2] PDesigner: Simulator development framework. `http://www.cin.ufpe.br/~pdesigner/`, 2008.

[3] R. Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon, School of Computer Science, January 1997. Issued as CMU Technical Report CMU-CS-97-144.

[4] R. Azevedo, S. Rigo, M. Bartholomeu, G. Araujo, C. Araujo, and E. Barros. The ArchC Architecture Description Language and Tools. *International Journal of Parallel Programming*, 33(5):453–484, 2005.

[5] C. G. Bell and A. C. Newell. *Computer structures: Readings and examples (McGraw-Hill computer science series)*. McGraw-Hill Pub. Co., 1971. ISBN 0070043574.

[6] A. Bona, M. Caldari, V. Zaccaria, and R. Zafalon. High-Level Power Characterization of the AMBA Bus Interconnect. In *SNUG*, 2004.

[7] J. Ceng, M. Hohenauer, R. Leupers, G. Ascheid, H. Meyr, and G. Braun. C Compiler Retargeting Based on Instruction Semantics Models. In *Proc. Design, Automation and Test in Europe*, pages 1150–1155, 2005. doi: 10.1109/DATE.2005.88.

[8] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra. A framework for system-level modeling and simulation of embedded systems architectures. *EURASIP J. Embedded Syst.*, 2007(1):2–2, 2007. ISSN 1687-3955. doi: http://dx.doi.org/10.1155/2007/82123.

[9] A. Fauth, J. Van Praet, and M. Freericks. Describing instruction set processors using nML. In *Proc. European Design and Test Conference ED&TC 1995*, pages 503–507, 6–9 March 1995. doi: 10.1109/EDTC.1995.470354.

[10] G. Hadjiyiannis, S. Hanono, and S. Devadas. ISDL: An Instruction Set Description Language For Retargetability. In *Proc. 34th Design Automation Conference*, pages 299–302, June 9–13, 1997.

[11] A. Halambi, P. Grun, V. Ganesh, A. Khare, N. Dutt, and A. Nicolau. EXPRESSION: a language for architecture exploration through compiler/simulator retargetability. In *Proc. Design Automation and Test in Europe Conference and Exhibition 1999*, pages 485–490, 9–12 March 1999. doi: 10.1109/DATE.1999.761170.

[12] A. Hoffmann, T. Kogel, A. Nohl, G. Braun, O. Schliebusch, O. Wahlen, A. Wieferink, and H. Meyr. A novel methodology for the design of application-specific instruction-set processors (ASIPs) using a machine description language. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(11):1338–1354, Nov. 2001. doi: 10.1109/43.959863.

[13] D. Kästner. TDL: a hardware description language for retargetable postpass optimizations and analyses. In *GPCE '03: Proceedings of the 2nd international conference on Generative programming and component engineering*, pages 18–36, New York, NY, USA, 2003. Springer-Verlag New York, Inc. ISBN 3-540-20102-5.

[14] S. Kwon, Y. Kim, W.-C. Jeun, S. Ha, and Y. Paek. A retargetable parallel-programming framework for MPSoC. *ACM Trans. Des. Autom. Electron. Syst.*, 13(3):1–18, 2008. ISSN 1084-4309. doi: http://doi.acm.org/10.1145/1367045.1367048.

[15] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon. Analyzing on-chip communication in a MPSoC environment. In *Proc. Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 752–757, 16–20 Feb. 2004. doi: 10.1109/DATE.2004.1268966.

[16] J. Magee and J. Kramer. Dynamic structure in software architectures. *SIGSOFT Softw. Eng. Notes*, 21(6):3–14, 1996. ISSN 0163-5948. doi: http://doi.acm.org/10.1145/250707.239104.

[17] P. Mishra and N. Dutt, editors. *Processor description languages*, chapter MIMOLA - A Fully Synthesizable Language, pages 35–63. Morgan Kaufmann, 2008.

[18] S. Steinke, L. Wehmeyer, B.-S. Lee, and P. Marwedel. Assigning program and data objects to scratchpad for energy reduction. In *Proc. Design, Automation and Test in Europe Conference and Exhibition*, pages 409–415, 4–8 March 2002. doi: 10.1109/DATE.2002.998306.

[19] M. Verma and P. Marwedel. *Advanced Memory Optimization Techniques for Low-Power Embedded Processors*. Springer, 2007.

[20] L. Wehmeyer and P. Marwedel. *Fast, Efficient and Predictable Memory Accesses*. Springer, 2006.