

Data-Reuse-Driven Energy-Aware Cosynthesis of Scratch Pad Memory and Hierarchical Bus-Based Communication Architecture for Multiprocessor Streaming Applications

Ilya Issenin, *Member, IEEE*, Erik Brockmeyer, Bart Durinck, and Nikil D. Dutt, *Fellow, IEEE*

Abstract—As technology advances, it becomes feasible to implement a large multiprocessor systems-on-chip (MPSoCs) to satisfy the increased performance demands of embedded applications. The increased complexity of systems leads to an increased power consumption. Reducing the consumption is an important task, considering that the available power may be limited in battery-operated embedded systems. The selection of memory and communication architectures affects the power efficiency of the design. In this paper, we propose a novel approach that enables the energy-aware cosynthesis of both memory and communication architectures for streaming applications. As opposed to earlier techniques, we propose a powerful compile-time analysis of memory access behavior in multiprocessor systems, which adds flexibility in selecting scratch-pad-based memory architectures. We propose and compare three memory/communication synthesis techniques, namely, an optimal mixed integer-linear-programming (ILP)-based cosynthesis technique, a mixed ILP (MILP)-based traditional two-step synthesis approach, where memory and communication synthesis is sequentially performed, and a cosynthesis heuristic that synthesizes energy-efficient hierarchical bus-based communication architectures with guaranteed throughput. Our experimental results on a number of streaming applications show that both the traditional two-step synthesis approach and heuristic result in up to 50% worse power consumption in comparison with our proposed cosynthesis approach. However, on some of the streaming benchmarks, our cosynthesis heuristic approach was able to find optimal or near-optimal results in a much shorter time than the MILP cosynthesis approach.

Manuscript received July 17, 2007; revised January 22, 2008. This work was supported in part by the National Science Foundation under Grant CCR-0203813 and Grant CCR-0205712. This paper was recommended by Associate Editor V. Narayanan.

I. Issenin is with the Center for Embedded Computer Systems, Department of Computer Science, Donald Bren School of Information and Computer Sciences, University of California, Irvine, CA 92697-2620 USA, and also with Teradek, Irvine, CA 92618 USA (e-mail: isse@ics.uci.edu).

E. Brockmeyer is with the Interuniversity MicroElectronics Center, 3001 Leuven, Belgium (e-mail: brockmey@imec.be).

B. Durinck is with the Interuniversity MicroElectronics Center, 3001 Leuven, Belgium, and also with ARM Belgium, 3001 Leuven, Belgium (e-mail: bart.durinck@imec.be).

N. D. Dutt is with the Center for Embedded Computer Systems, Department of Computer Science, Donald Bren School of Information and Computer Sciences and with the Department of Electrical Engineering and Computer Science, The Henry Samueli School of Engineering, University of California, Irvine, CA 92697 USA (e-mail: dutt@ics.uci.edu).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TCAD.2008.925781

Index Terms—Bus synthesis, data reuse, embedded systems, hierarchical bus-based communication architecture, memory optimizations, multiprocessor scratch pad memory (SPM) hierarchy.

I. INTRODUCTION

IMAGE and video processing applications typically have a lot of parallelism and are thus perfect candidates for being parallelized and executed on multiprocessor systems-on-chip (MPSoC). In addition to easy parallelization, such applications typically have regular memory access patterns that can be statically analyzed and predicted at compile time. This property allows us to replace caches that are used in general-purpose systems by scratch pad memories (SPMs). SPMs are more power efficient than caches because of the absence of tag memory, tag comparators, and hardware that enforce cache coherence. Furthermore, SPMs allow for the easier and tighter estimation of worst case execution time bounds of real-time behaviors.

With the memory subsystem being one of the important contributors to embedded system power consumption, the communication subsystem also contributes to the total system power consumption. While, with the shrinking technology feature sizes, the energy consumption and performance of transistors (and thus computational cores and memories) are improving, the energy consumption and delay of long global wires do not follow that of the transistors and do not change or even increase with the decreased technology feature size [12], [30]. As the result, the relative energy consumption of the communication subsystem is increasing in comparison with that of the computation blocks and memory subsystem. That is why in modern designs, both the memory and communication architectures play an important role in meeting the performance and energy consumption requirements of applications executed on MPSoCs.

In this paper, we consider the problem of simultaneously designing and optimizing both the memory and communication subsystems.

In order to design SPM-based memory subsystems, the compile-time analysis of the application is required so that the necessary transfers between memories are effected and coherence is maintained if the same SPM is used by several processors. In Section III, we extend the uniprocessor data reuse analysis introduced earlier [16] by proposing a multiprocessor

data reuse analysis technique that is able to generate a number of SPM hierarchies for a given multiprocessor code. This enables the exploration of the design space of customized memory hierarchies.

Traditionally, the task of memory/communication subsystem synthesis consists of two steps. First, the memory architecture (physical memories and the mappings of data to them) is defined. In a following step, the connectivity synthesis is performed. While greatly reducing the complexity of exploration, the separation of these two steps can lead to suboptimal solutions and miss many interesting design points.

Some recent efforts have begun to simultaneously explore memory architectures together with communication synthesis. However, in all these works, the memory architecture transformations were very simplistic, e.g., merging several memories into one [24] or splitting the memories and removing the ones that are used by only one processor from the shared bus and making them private [23]. In this paper, we propose to perform memory and communication cosynthesis based on the multiprocessor data reuse analysis, which finds a set of possible buffers, which holds copies of the frequently used data in the main memory. That allows us to consider a much broader set of possible memory architecture transformations in comparison with previous approaches to communication synthesis. After obtaining the buffers, we select and map some of these buffers into physical memories and simultaneously perform communication synthesis for minimal power while meeting given time constraints.

In our approach (unlike in most of the previous works), we are targeting hierarchical time-division multiple-access (TDMA) bus-based communication system. The use of buses with TDMA arbitration (e.g., Sonics SiliconBackplane III [27]) is beneficial for streaming applications because it allows the bus to provide channel throughput guarantees, making the communication subsystem fully predictable even in the presence of multiple logical channels on the bus (when several masters are communicating with several slaves at the same time). Predictability eliminates the need for overdesigning the bus to meet the constraints in the worst case of collisions, considering that there are no collisions on the TDMA bus. Another advantage of having a communication subsystem with guaranteed throughput is that there is no need to perform time-consuming simulations to determine if the communication subsystem with selected parameters meets the timing constraints. This permits the exploration of a much broader design space in a reasonable amount of time and can even find the optimal solution, both of which are not usually possible with traditional non-TDMA buses.

In this paper, we propose an optimal mixed integer-linear-programming (MILP)-based approach for memory/communication architecture cosynthesis based on a communication (or architecture) template, which is described in Section IV. We compare this approach with a traditional two-step synthesis approach (first memory, then communication synthesis) and also with a simple cosynthesis heuristic. We show that the MILP-based optimal cosynthesis approach provides results that outperform a two-step or heuristic approach in a reasonable amount of time for the benchmarks we used. However, our heuristic, being much less computationally ex-

pensive than the MILP cosynthesis approach, achieves near-optimal results on some of the benchmarks, which makes it a good candidate for solving the problems for which the MILP cosynthesis technique is not able to produce the results in a reasonable amount of time.

II. RELATED WORK

There is a significant amount of research in the areas of SPM and bus-based communication synthesis.

A number of works have presented approaches employing SPMs for performance and power improvements in uniprocessor systems. Many of the approaches use SPMs to store the most frequently used scalars or whole arrays [21], [28], [31], [32], where the placement of parts of an array in the SPM is not possible. However, our technique is geared toward the run-time placement of currently active array elements in SPMs and thus is complementary to the works which are focused on the placement of nonarray data structures. It should be noted that typical multimedia applications employ large arrays and use computations that cannot fit all the desired data structures into local memory. Thus, we believe that it is critically important to develop strategies like ours that handle accesses to arrays residing in a large and expensive main memory. We proposed such technique targeting a uniprocessor system in [16].

Among approaches targeting multiprocessor systems, Ozturk *et al.* [20] proposed to customize on-chip multiprocessors with private and shared single-level memories which use a replacement policy that is similar to a cache. Chen *et al.* [7] proposed a strategy for mapping the data to local single-level SPMs in a multiprocessor system. In contrast to these works which have a single-level and fixed (in [7]) SPM architecture, our approach presented in this paper allows the generation of a number of possible multilevel SPM hierarchies consisting of shared and private scratch pads with different overall power, performance, and communication requirements, giving our memory/communication cosynthesis techniques additional freedom in customizing the memory and communication subsystems.

In the area of bus-based communication synthesis, a number of works presented methodologies for automated bus generation. Lyonnard *et al.* [18] and Ryu and Mooney [26] present frameworks that generate custom bus systems using predefined IP cores. Pinto *et al.* [25] proposed an algorithm for topology synthesis given a placement of components together with communication requirements. Gasteier and Glesner [9] proposed a communication synthesis approach for a system based on shared buses. Pasricha *et al.* [23] presented a heuristic for throughput-driven communication synthesis. All of the aforementioned works perform design optimizations for cost/area or performance but not for energy that is targeted in this paper.

In bus optimization research aimed at energy reduction, Aghaghiri *et al.* [1] proposed to use bus encoding. Zhang and Rabaey [35] studied low-swing bus interface implementations. Chen *et al.* [6] proposed bus segmentation to reduce power consumption. Wang *et al.* [33] proposed to perform floorplanning, placing communication-intensive modules closer to each other for segmented bus-based designs. Guo *et al.* [11] extended this approach by additionally reordering blocks connected to a

linear segmented bus to place frequently communicating blocks next to each other on the bus. All of these approaches do not modify the memory architecture of the system.

In the area of memory and communication architecture synthesis, the cosynthesis aspect was mostly ignored, and memory allocation and mapping was performed before communication synthesis [4]. Only a few techniques consider memory and communication cosynthesis. Kim *et al.* [17] performed the mapping of memories to buses along with bus topology selection. Pasricha and Dutt [24] performed bus matrix communication synthesis together with limited memory transformations, optimizing for the number of buses and memory area. Papanikolaou *et al.* [22] proposed to consider several (rather than one) memory organizations, perform energy-efficient communication synthesis for each of them, and leave Pareto optimal design points as the candidates for system implementation. A communication system consisting of several linear segmented buses was considered. However, in that work, it was not explained how different memory organizations were created, and no automated way for exploring possible memory organizations was proposed. Grun *et al.* [10] performed cosynthesis by selecting memory and connectivity types from a predefined set of modules, such as cache, SPM, streaming buffers, etc., for memories and shared AMBA bus, point-to-point buses, etc., for connectivity modules.

This paper is significantly different from previous efforts in that we simultaneously perform memory/connectivity cosynthesis for hierarchical bus architectures that employ highly customized memories while aiming at system energy reduction. Considering that we are targeting buses with TDMA arbitration, we are also able to find the optimal (for our template) energy configuration with guaranteed performance.

III. MULTIPROCESSOR DATA REUSE ANALYSIS FOR EXPLORING CUSTOMIZED MEMORY HIERARCHIES

To perform memory/communication cosynthesis, it is necessary to define a space of possible memory subsystem configurations that are considered during the cosynthesis. In this paper, we consider hierarchical SPM-based subsystems, which are constructed based on data reuse analysis. In this section, we present a technique for performing automated data reuse analysis which generates a reuse graph, which is essentially a combination of different possible scratch pad buffer hierarchies that can be implemented for a given application. This reuse graph is later used in our cosynthesis techniques described in Section IV.

A. Memory Model

In our approach, we consider a multiprocessor system consisting of several processors and shared or private SPMs. Our technique handles all data accesses with regular access pattern (i.e., vector accesses) expressible in FORAY form [14] (i.e., memory references are located inside arbitrary structure of “for” loops and have index expressions expressed by affine functions of outer loop iterators), which constitute a large portion of accesses in typical embedded multimedia applications.

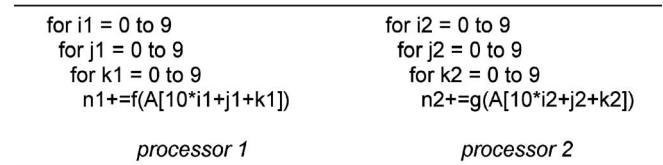


Fig. 1. Code fragment using same data.

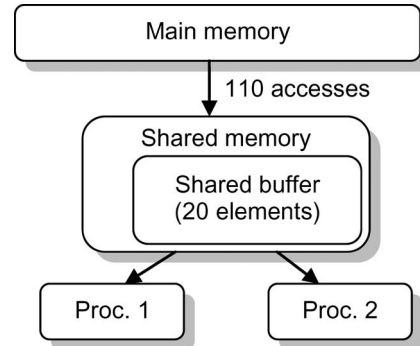


Fig. 2. Memory subsystem with shared memory.

We assume that there is a mechanism to handle all other data memory accesses. For example, all local variables and stack can be placed in the local memories of each of the processor. If there is a need to use bigger (or shared) nonlocal memory, caches supporting coherence protocol could be used by each processor to handle such irregular requests.

B. Synchronization Model

When several processors use the same data, it may be beneficial to implement a buffer in a memory that can be accessed by both processors instead of duplicating the data in local memories. We call such a buffer a shared buffer.

Fig. 1 shows an example of code where two processors are using the same array elements. While it is possible to duplicate array A and store it in local memories, this would require 218 accesses to the main memory. In this case, the working data set (elements $A[0] \dots A[108]$) are copied to the local memories of each processor. However, if we introduce a small shared SPM [which contains a circular¹ buffer of size 20 (two sets of ten elements) holding elements consumed by inner loops j and k and updating one set with each iteration of outer loop i], it is possible to reduce the number of accesses to the main memory to 110. Such a memory subsystem is shown in Fig. 2.

In this configuration, the shared buffer is updated on every iteration of the outermost loop (i). From this example, it should be clear that the presence of a shared buffer does not allow processors to run independently, which is even if there are no data dependences between processors in the original program. Now, both processors have to synchronize their execution before updating the shared buffer; otherwise, the buffer update initiated by one of the processors may expel the data from a buffer that has not been read yet by the second processor.

¹We assume that logically circular buffers are mapped into regular random access memory (SPM). The mapping is done in software, as described in [16].

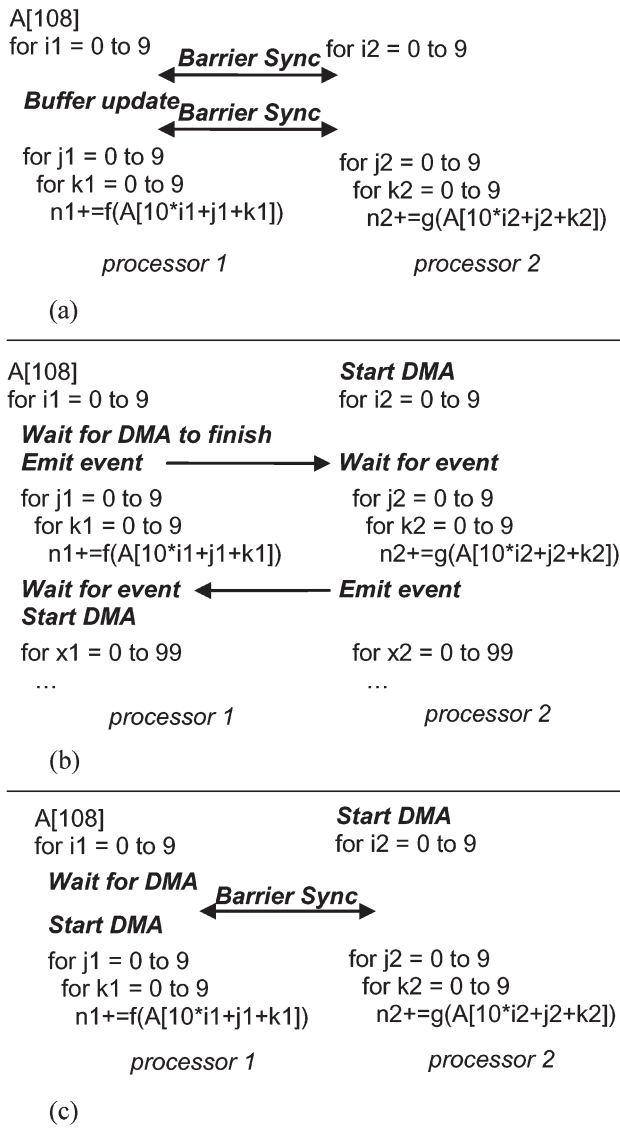


Fig. 3. Different ways of implementing buffer updates and processors synchronization. (a) Buffer update with barrier synchronization (BUBS). (b) Buffer update with execution overlapping (BUEO). (c) Buffer update using larger buffer size (BULBS).

The most straightforward way to implement such synchronization is by using two barrier synchronizations, as shown in Fig. 3(a). In this scheme, both processors are stalled when they start executing the body of loop i . Then, one of the processors issues a DMA transfer request that fills the buffer with the new data. After the DMA transfer is complete, both processors can continue their execution.

The drawback of such a scheme is that all processors are wasting time while waiting for the DMA to complete. A more efficient scheme would overlap the DMA prefetch with the program execution. If there is another piece of code in the same loop body (of loop i) that does not use the data elements from the same buffer, it is possible to overlap the buffer update with the execution, as shown in Fig. 3(b).

Even if there is not enough processing to completely hide the buffer filling latency, it is still possible to increase the size of the circular buffer by one data set so that the DMA could prefetch

Algorithm 1: Data Reuse Detection for Multiprocessors (DRDM)

1. Map the application to processors and select parts of the program to be optimized; extract loop and array reference information
 2. Select synchronization granularity. For each selection do:
 - 2.1. Convert multiprocessor reuse analysis problem into uniprocessor reuse analysis problem
 - 2.2. Apply uniprocessor data reuse analysis technique to the obtained uniprocessor problem
 3. Combine obtained reuse trees into one multiprocessor reuse graph
 4. For each of the shared buffers in the reuse graph, select synchronization scheme and adjust buffer size if necessary
-

Fig. 4. Our approach for program analysis.

data to that part of the buffer while the processors are using data from the other part. For our current example, this would require increasing the size of the buffer from 20 to 30 elements [Fig. 3(c)]. In the rest of the section, we present our approach for detecting data reuse in a multiprocessor system. We show how it can be applied for building highly customizable scratch-pad-based memory subsystems in Section IV.

C. DRDM

Algorithm 1 in Fig. 4 outlines our data reuse detection for multiprocessors (DRDM) approach. First, we obtain a code parallelized for execution on several processors and determine time-critical and memory-intensive parts (kernels) that should determine the final memory configuration. Considering that we perform compile-time analysis and that an arbitrary program cannot be statically analyzed, we assume that the memory accesses have regular access pattern that can be expressed as affine functions of the outer loop iteration (a reasonable assumption for multimedia applications). Fig. 1 shows an example of such code. Our technique takes a description of loops and array index functions for each of the processors as the input. If the code does not contain explicit index functions but has a behavior that can be described with them, our approach described in [14] can be used to convert the code into this form.

In Step 2 of Algorithm 1, we select at which loop level the processors should be synchronized (different synchronization options were discussed in Section III; in those examples, the processors were synchronized at loop i). This granularity selection trades off the overhead caused by the additional synchronizations by the buffer size (typically, synchronization at the inner loops leads to a higher number of synchronizations and smaller buffer size). Considering that a tradeoff is present, it is beneficial to keep alive more than one possible synchronization decision and do further analysis (Steps 2.1 and 2.2) on each of them, including the case when no synchronizations are present. Note that in case of more complex loop structures (other than perfect loop nests), it is possible to have more than one synchronized loop at the same time; these loops, however, should not be one inside the other to exclude the possibility of deadlocking.

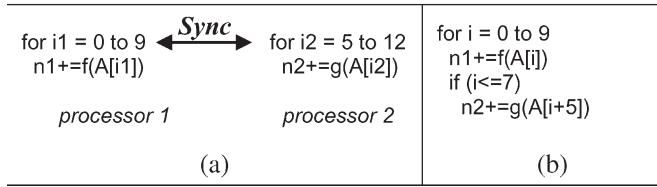


Fig. 5. Multiprocessor and uniprocessor programs with the same data reuse pattern. (a) Multiprocessor program. (b) Uniprocessor program.

There may be many possible ways to synchronize the loops. However, not all of them are equally useful, and thus, less useful ones should be pruned.

For example, for the program shown in Fig. 1, the possible synchronization loop pairs are: $i1-i2$, $i1-j2$, $i1-k2$, $j1-i2$, $j1-j2$, $j1-k2$, $k1-i2$, $k1-j2$, and $k1-k2$. However, the pair $j1-i2$ (i.e., during the first iteration of loop $i1$ on processor 1, each iteration of the loop body $j1$ is synchronized with the iteration of the loop body $i2$ on processor 2) is not useful. To understand why, note that after the first iteration of $i1$, the execution of the program on processor 2 will be finished, and processor 2 will be stalled during the rest of the nine iterations of loop $i1$. If there is no more code to execute on processor 2, this synchronization choice leads to poor load balancing, and thus, there is no need to perform any future data reuse analysis for the pair $j1-i2$.

To summarize, a set of computationally simple checks can be performed to filter out synchronization candidates that definitely cannot lead to obtaining useful scratch pad buffers after the data reuse analysis.

- 1) The number of loop iterations on all processors should not differ too much (the opposite leads to poor load balancing).
- 2) The coefficients in the index expressions of the synchronized loops and all corresponding outer loops should be the same (otherwise, there will be no regular reuse pattern, and no shared buffers will be suggested by data reuse analysis).
- 3) The intervals formed by the minimum and maximum values of index expressions on each processor should overlap (otherwise, no shared buffers will be suggested by data reuse analysis).
- 4) The number of memory accesses during one iteration of synchronized loops should be large enough so that the DMA initialization penalty and synchronization cost are less than the cost of transferring the (useful) data.

We found that on typical applications, only few options are left after applying these rules.

In Step 2.1 of Algorithm 1, we convert the multiprocessor data reuse analysis problem into a uniprocessor analysis problem, using the following observation as shown by Fig. 5.

The program shown in Fig. 5(a) has two loops executed on two processors which are synchronized as explained in the previous section. During the first iteration of the loop elements, $A[0]$ and $A[5]$ are simultaneously accessed (if the shared memory with array A has only one port, the elements are accessed in arbitrary order but during the first loop iteration). On the second iteration, elements $A[1]$ and $A[6]$ are accessed, and so on.

The functionally equivalent uniprocessor program in Fig. 5(b) accesses the same elements but in a fixed order (first $A[0]$, then $A[5]$ during the first iteration, and so on). However, from the reuse analysis point of view, both programs have identical data reuse patterns. This suggests that we can apply uniprocessor data reuse analysis to multiprocessor programs by merging programs. During such a merge, all the loops mentioned earlier, including synchronized loops, are replaced by the set of loops from the first processor, and the inner loop bodies are sequentially added. The constant term in the index expressions and the upper loop bounds may need to be adjusted (as it is done in Fig. 5). Note that the “if” condition that was added to the uniprocessor program in Fig. 5(b) to preserve the functionality is not needed because it does not change the reuse pattern.

After obtaining the uniprocessor code, it is analyzed in Step 2.2 of Algorithm 1 using a technique for uniprocessor data reuse analysis such as described in [16]. The result of the analysis is a hierarchical set of buffers, which is also called a reuse tree. Each buffer in the reuse tree can be mapped to a physical SPM or left not implemented. A more detailed description of reuse trees and the way they are obtained for uniprocessor systems can be found in [16].

In Step 3 of Algorithm 1, we combine the buffers obtained by uniprocessor data reuse analysis for different synchronization selections into one graph. Fig. 6 shows an example of a multiprocessor program with possible synchronization choices [Fig. 6(a)], reuse trees obtained by uniprocessor reuse analysis for these different synchronization selections [Fig. 6(b)–(e)], and the combined final reuse graph [Fig. 6(f)]. The merging operation is performed by taking the first reuse tree and adding nodes and edges from other reuse trees that are not already present in the graph. For example, j -loop nodes for the first processor in Fig. 6(b) and (c) are merged to the same node in the combined reuse graph; however, the i -loop nodes in Fig. 6(b) and (c) are not merged because they belong to different processors (two i -loop nodes in Fig. 6(b) belong to processors 1 and 2, and the i -loop node in Fig. 6(c) is a shared buffer that belongs to both processors at the same time). The numbers inside the boxes representing buffers are the sizes of the buffers suggested by a uniprocessor reuse analysis. Note that not all buffers in the reuse graph can be implemented simultaneously in SPMs. For example, on the i -loop level, a shared buffer (of size 16) and processor 1’s private buffer (of size 14) are exclusive alternatives and are not meant to be implemented simultaneously. The general rule is that if two buffers in the reuse graph are on the same loop level and have common descendants, they are mutually exclusive. Each node of the reuse graph (representing a buffer), in addition to its size and level in memory hierarchy, also holds information about the number of requests (traffic) to a higher level of memory hierarchy [not shown in Fig. 6(f)]. Note that the number of requests are calculated per one period of application execution, e.g., per one frame for image or video processing applications.

In Step 4 of the DRDM algorithm, for each of the shared buffers, a synchronization scheme is selected from the schemes explained earlier. The buffer update with execution overlapping (BUEO) and buffer update using larger buffer size (BULBS) schemes (Fig. 3) allow one to hide memory access latency by

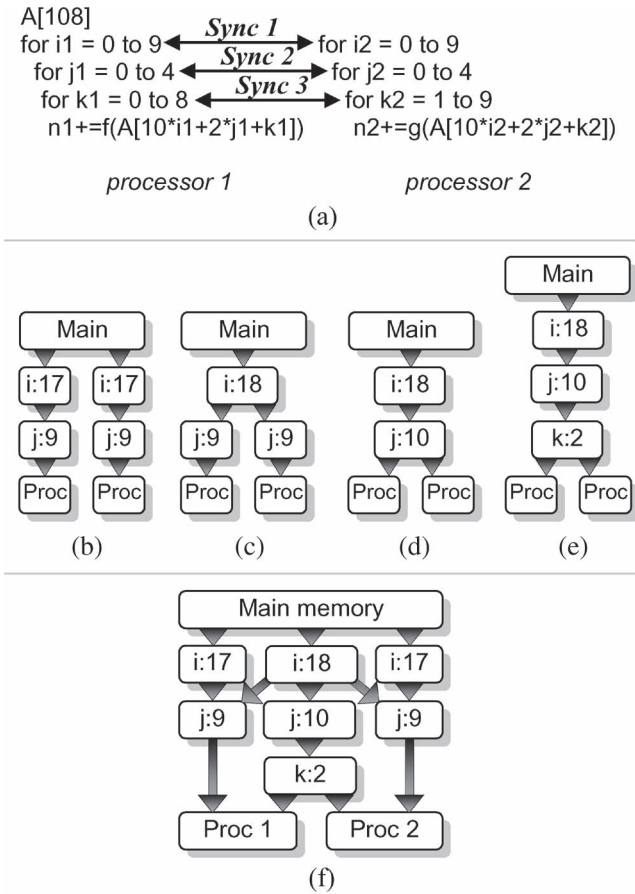


Fig. 6. Combining the buffers obtained by uniprocessor data reuse analysis. (a) Input multiprocessor program. Buffers obtained by uniprocessor data reuse analysis when (b) no synchronization between processors is present, (c) (Sync 1) synchronization is at the i -loop level, (d) (Sync 2) synchronization is at the j -loop level, and (e) (Sync 3) synchronization is at the k -loop level. (f) Final multiprocessor reuse graph.

overlapping DMA transfers with program execution. If there is enough (to hide access latency) computation in the loop that do not require the data from the shared buffer, the BUEO scheme can be selected. Otherwise the BULBS scheme should be used to hide the DMA transfer time. However, if the BULBS scheme is used, the buffer size should be increased to accommodate one extra data set in the buffer.

Finally, after applying our DRDM technique described earlier, the designer (or design space exploration algorithms, such as described in the following sections) is provided with many more options for implementing a memory subsystem than just having one memory for holding the array.

The advantage of having intermediate buffers is not only in exploiting data reuse, i.e., in reducing the number of main memory accesses (and thus saving time and energy). Typically, single accesses to fetch data are much more expensive than the block (DMA) transfer of large amounts of data at once. Our approach allows for the implementation of an efficient prefetch scheme, where data are brought as close to the processor as possible by block transfers, and the processors make single accesses only to the closest (local) memory. This scheme is efficient because the memory latency can be completely hidden (assuming that there is enough bandwidth available), and for

the processor, all the fetches of the data from the main memory will have the latency of a small local memory. In order to have the intermediate buffers that hold nonreused data present in the output of our technique, it is necessary to have them included in the reuse trees produced by the uniprocessor data reuse analysis [16]. In our cosynthesis heuristics described in the next section, we assume that such buffers with nonreused data are included in the reuse graph.

Note that in our approach, once we allocate an SPM space for a particular data reuse buffer, this allocation is retained during the entire program execution (although the mapping of array elements to the buffer changes during the program execution). Catthoor *et al.* [5] proposed in-place optimization, which maps arrays (or buffers) with nonoverlapping lifetimes to the same physical space in SPMs. Although their approach can be used with our approach presented in the next section, we did not complicate our models by incorporating this optimization step, considering that for all benchmarks we studied, all the buffers were alive all the time.

IV. DATA-REUSE-DRIVEN ENERGY-AWARE MPSoC COSYNTHESIS OF MEMORY AND COMMUNICATION ARCHITECTURE FOR STREAMING APPLICATIONS

A. Architecture Template for MPSoC Data Parallelized Applications

There are many different interconnect topologies for MPSoCs, which can be categorized into several groups, e.g., shared bus system, hierarchical bus system, bus matrix, etc.

Typically, the choice of the type of system connectivity depends on the application complexity, the way it was mapped to processing elements, the design requirements, and other factors. In this section, we perform a connectivity synthesis based on an architecture template that defines the types of connectivity allowed between MPSoC IPs.

The general hierarchical bus-based template we use in this paper is shown in Fig. 7. The template is based on buses with TDMA arbitration that provides guaranteed throughput. The template is configured by three parameters, namely, the number of processors, the number of levels of bus hierarchy, and the number of different SPM types (sizes) at each bus level. These parameters that determine the exploration space can vary depending on the application and are specified by the designer.

One of the results of the memory/connectivity synthesis is the template instance, which is a customized template with some or all components omitted, except for the main memory and processors. The position of horizontal bus connection, which can be at any bus hierarchy level (positions are shown by the dashed lines in Fig. 7) is also part of customization. The position of the horizontal bus connection determines which memories are shared and which are private; all SPMs below the horizontal bus connection are private and can be accessed only by the processors located under them in Fig. 7. Note that there are no memories allowed in the template instance above the horizontal bus connection on processors $2 \dots N$. The bridges constrain the propagation of useless traffic to higher levels of bus hierarchy and thus allow for savings in energy by reducing the number of transactions and the maximum required

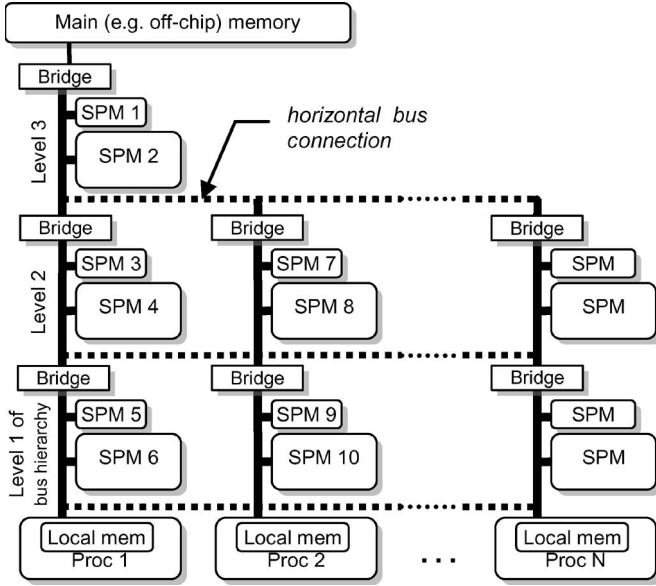


Fig. 7. Example of the template for N processors, three bus levels, and two memory types.

throughput of the buses at the higher levels of the bus hierarchy. The bridges can be implemented with internal buffers or as switches that divide the bus into segments [6].

Our proposed algorithms (described later in this section) customize the template by determining the components that are omitted and the ones that are left in customized architecture, the mapping of the data to the memories, and the position of the horizontal bus connection.

The data are moved between the SPMs, main memory, and local processor memories by DMA engines, which are not shown in Fig. 7. Note that the processors themselves can only access the data in their local memories shown inside the processor boxes. They cannot access the buses outside the processors. Only DMA controllers are allowed to prefetch the data to (or save data from) these local memories. Together with TDMA bus arbitration, this allows for the scheduling of DMA transfers so that the connectivity subsystem behavior becomes fully predictable, isolating the uncertainty or possible bus contentions introduced by unpredictable processor timing. Local processor memories also contain stack and scalar variables. Only data arrays with large footprints (or with shared data) are kept outside the local memories.

The proposed template is general enough to describe any architecture consisting of a combination of shared and private SPMs, as long as the connections are symmetric (e.g., there cannot be a shared SPM which is accessible by only some processors) and is well suited for homogeneous MPSoCs. Fig. 8 shows some of the typical custom memory subsystems that can be obtained by customizing a three-processor template. The gray components in the figure are those that were not selected for implementation. Such MPSoCs are used when an application is partitioned into several processors by using available data parallelism, with each processor processing only part of the original data set. This partitioning is often used in data streaming applications, which is when a particular task cannot be mapped into a single processor due to performance or energy-efficiency

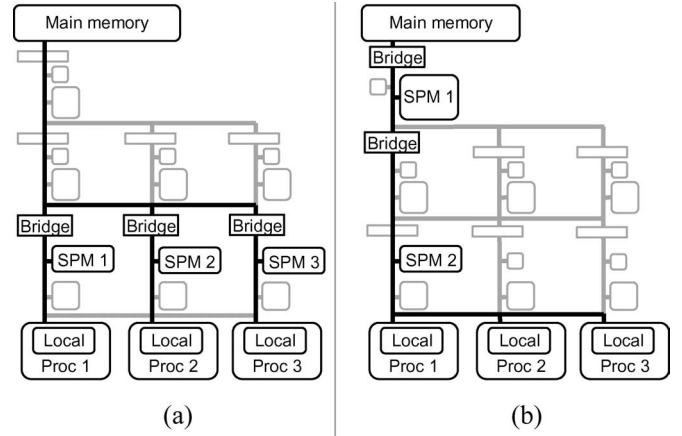


Fig. 8. Examples of customized memory subsystems for three processors. (a) Memory subsystem with private SPMs. (b) Memory subsystem with two-level shared SPMs.

constraints. However, applications partitioned into heterogeneous platforms can also use the same template (e.g., one of our benchmarks, which is the QSDPCM encoder, is parallelized into three pipelined stages, with only one of the stages using four processors for data-parallel (homogeneous) processing). The proposed template covers more complex memory architectures than, e.g., architectures considered in [7].

B. Energy Models

Recall that the goal of our cosynthesis approach is to generate a minimal energy design that guarantees the required performance.

To estimate the energy consumption, we have accounted for the following components:

$$E_{\text{total}} = E_{\text{SPM}} + E_{\text{main_mem}} + E_{\text{bus}} + E_{\text{bridge}} \quad (1)$$

where E_{SPM} is the energy consumed by the SPMs, $E_{\text{main_mem}}$ is the energy consumed by the main memory, E_{bus} is the energy consumed in the buses, and E_{bridge} is the energy consumed in the bridges. These values are defined as

$$E_{\text{SPM}} = \sum E_{\text{acc}}^* N_{\text{acc}} \quad (2)$$

$$E_{\text{main_mem}} = E_{\text{acc}}^* N_{\text{main_mem}} \quad (3)$$

where E_{acc} is the memory energy per access, N_{acc} is the number of accesses to the SPM, and $N_{\text{main_mem}}$ is the number of accesses to the main memory. Note that the total energy, number of accesses, and other values are calculated per period of application execution. Here, we ignore the leakage energy because it is a fixed value for the period of application execution, and thus, adding or removing it from optimization criteria does not change the result of the optimization

$$E_{\text{bus}} = \sum (E_{\text{bus_active}}^* N_{\text{trans}} + E_{\text{bus_static}}^* N_{\text{bus_cycles}}) \quad (4)$$

where $E_{\text{bus_active}}$ is the energy consumption of a bus (including bus interfaces) per transaction (e.g., one burst transfer of data), N_{trans} is the number of such transactions on the selected bus, $E_{\text{bus_static}}$ is the power consumption of a bus per bus clock

cycle (e.g., in the bus interfaces and in the bus clock wires), and $N_{\text{bus_cycles}}$ is the number of bus clock cycles in the application period. Considering that both $E_{\text{bus_active}}$ and $E_{\text{bus_static}}$ may depend on the number of bus interfaces present on the bus (due to power dissipation in the interfaces and the change in the bus capacitance), we have accounted for that as well

$$E_{\text{bus_active}} = E_{b_a_const} + E_{b_a_coef}^* N_{\text{bi}} \quad (5)$$

$$E_{\text{bus_static}} = E_{b_s_const} + E_{b_s_coef}^* N_{\text{bi}} \quad (6)$$

where N_{bi} is the number of bus interfaces on a particular bus and E_{a_const} , E_{b_coef} , E_{s_coef} , and $E_{b_s_coef}$ are some coefficients.

The energy spent in bridges is calculated as

$$E_{\text{bridge}} = \sum E_{\text{br}}^* N_{\text{tr_br}} \quad (7)$$

where E_{br} is the energy spent in a bridge per transaction and $N_{\text{tr_br}}$ is the number of transactions that go through the bridge. Note that because a bridge is connected to two buses, we also account for the additional energy spent in two bus interfaces.

C. MILP Synthesis Approach

We now describe an MILP approach for optimally solving the problem of communication/memory subsystem cosynthesis, which finds a configuration of the architecture template described earlier (i.e., the location of horizontal bus connection, the selection of the types of buses that satisfy throughput requirements defined by memory traffic, and the selection of bridges/SPMs to implement), together with a selection and mapping of buffers from the reuse graph to SPMs, so that the total energy consumption of the system is minimized.

The MILP problem formulation is the following:

$$\text{minimize } E \text{ under constraints } C$$

where E is the objective function, which is described in the previous section, and C represents the problem constraints, which is described later in this section.

In the following sections, we will explain MILP formulation variables, parameters, objective function, and constraints.

1) *Architecture Template Parameters*: First, we define the architecture template parameters that are specified by the designer:

$\text{proc_}n$: the number of the processors;

$\text{bus_}n$: the number of hierarchical buses in the template (three for the example in Fig. 7);

$\text{mem_}T$: the number of different types of physical memories (of different sizes) that can be used as SPMs;

$\text{mem_type}_i\text{size}$: the size of physical memory of each type;

$\text{bus_}t$: the number of different types of buses that can be used in the system;

bus_freq_i : the bus frequency (number of words per second that can be transferred through the bus) of bus type i ;

$E_{b_a_consist_i}$, $E_{b_a_coef_i}$, $E_{b_s_coef_i}$, $E_{b_s_coef_I}$, and E_{br} : the energy consumption parameters (described in the previous section) for the bus type i ;

$T_{\text{acc_}i}$: the access time of the memory of type i ;

$T_{\text{acc_main}}$: the access time of the main memory;

$E_{\text{acc_}i}$: the energy consumption per access for the memory of type i ;

$E_{\text{acc_main}}$: the energy consumption per access for the main memory.

2) *Application and Reuse Graph Information*: Here, we list the constants related to the information obtained from the application or reuse graph:

T : application period in seconds;

$\text{buf_}n$: total number of buffers in the reuse graph;

buf_req_i : number of requests to the higher level of memory hierarchy for the buffer i ;

proc_req_i : number of memory requests (in words) of processor i ;

buf_size_i : the size of the buffer i .

3) *Variables*: The following are the binary variables that describe the template configuration and reuse graph buffers' mapping. These are the values that are determined by solving the MILP problem.

ex_link_i : The horizontal bus connection is at the bus level i ($\text{ex_link}_i = 1$) or at another level ($\text{ex_link}_i = 0$); $1 \leq i \leq \text{bus_}n$. The horizontal bus connection can be at only one bus level.

$\text{ex_br}_i\text{proc}_j$: The bridge at the bus level i at processor j exists (one) or does not exist (zero).

$\text{buf}_i\text{mem}_j\text{bus}_k\text{proc}_p$: The reuse graph buffer i is mapped to the SPM of type j at the bus level k of processor p .

$\text{bus}_i\text{type}_j\text{proc}_k$: The bus i on processor k is of type j .

4) *Objective Function*: The function to be minimized by the ILP solver is the total energy consumed by the memory/communication subsystem. The energy model was described by using (1) earlier.

Note that the energy model [(1)–(7)] includes quadratic terms, which are not allowed in MILP. We use a reduction to convert such nonlinear constraints to linear form that is allowed in the MILP formulation. It is possible to express the multiplication of several variables as an MILP problem as long as only one of the variables is not binary and its bounds are known, e.g., a quadratic constraint

$$k = f * b, b \in [0, 1], 0 \leq f \leq \text{fmax}$$

where fmax is the upper bound on f , which can be rewritten as the system of linear constraints

$$\begin{cases} k \leq f \\ k \leq b * \text{fmax} \\ k \geq f - \text{fmax}(1 - b) \\ k \geq 0. \end{cases}$$

5) *MILP Constraints*: The constraints can be divided into several groups, which are briefly described here.

Constraints related to mapping of buffers to SPMs:

- 1) Each buffer from the reuse graph is assigned to zero or one physical memory.
- 2) Two buffers that belong to the same level in the reuse graph and hold data for the same processor are mutually exclusive.
- 3) Any two buffers, which are in a parent–child relation in the reuse graph, should be mapped so that the memory with the parent buffer is on the same or higher bus level in the bus hierarchy than the memory with the child buffer.
- 4) All shared buffers in the reuse tree should be mapped to the memories on the same or higher bus level as the horizontal bus connection.
- 5) All the buffers that are mapped to the memories, which are below the horizontal bus connection, should be private and be located on the proper processor bus.
- 6) There should be no memories or bridges on the buses at the levels that are higher than one of the horizontal bus connections at processors $2 \dots \text{proc}_n$.

Constraints related to the design requirements:

- 1) The buffers mapped to the same physical memory should fit into it.
- 2) The time needed to perform all accesses to physical memories should be less than the application period.
- 3) The time needed to perform all transfers on each bus should be less than the application period.

Constraints related to the bus types: Each bus has a single type assigned to it.

Symmetry constraints: To reduce the complexity of the ILP problem, we assumed that the configuration of the memories, buses, and bridges is the same (symmetric) for processors $2 \dots \text{proc}_n$. This is typically true because the code executed on the processors and the volume of the data they process are approximately the same for applications that are distributed to processors using data parallelization.

Incoming traffic for the reuse graph buffers: In order to calculate the number of accesses to the physical memories and the number of transactions on each bus, it is necessary to have equations for the number of accesses to each buffer (e.g., buffer reads) and the number of requests from the buffer to the higher level of memory hierarchy (which is the same as the number of buffer updates, i.e., writes). While the second number is fixed and known from the reuse graph, the number of buffer accesses depends on the presence of other buffers in the hierarchy.

Bus traffic: The traffic on each bus segment is calculated by summing up all the traffic that goes to/from the buffers mapped to the memories located on the bus segment and the traffic that comes from the bridges or connected segments if the bridge is not present.

The MILP problem we formulated was solved by using the commercial ILP solver CPLEX [8], as described in Section V.

D. Heuristic-Based Synthesis Approach

ILP formulations typically do not scale well with problem size. Consequently, we devised a simple greedy heuristic for memory/communication cosynthesis that has much shorter execution times and that scales better with the problem size.

Co-synthesis heuristic

input: set B of reuse graph buffers

set M of physical memories in architecture template

set BR of bridges in architecture template

output: template configuration and memory mapping (the values of $\text{buf}_b\text{mem}_m, \text{ex_br}_{br}, \text{horiz_bus_connection_level}, \text{bus}_b\text{type}_{bt}$)
Conf

1. Initialize $\text{buf}_b\text{mem}_m = 0$ for all b, m
2. Initialize $\text{ex_br}_{br} = 0$ for all br
3. for each $b \in B$ in the order of decreasing metric $(\text{buf}_b\text{in-buf}_b\text{req})/\text{buf_size}_b$ do {
4. Calculate total energy E for the current configuration
5. for each $m \in M$ do {
6. for $\text{horiz_bus_connection_level} = 1 \dots \text{bus}_n$ do {
7. $\text{buf}_b\text{mem}_m = 1; \text{buf}_b\text{mem}_{mm} = 0$ for all $mm \neq m$
8. Jump to Line 16 if current configuration is not valid
9. for each $br \in BR$ do {
10. Calculate the total energy E_1 and save the current configuration to Conf_1
11. $\text{ex_br}_{br} = \text{not ex_br}_{br}$
12. Calculate the total energy E_2
13. if $(E_1 < E_2)$ set the current configuration to the one saved in Conf_1
14. }
15. if $(E_1 < E \text{ or } E_2 < E)$ save the current config. to Conf
16. }
17. }
18. Set the current configuration to the one saved in Conf
19. }
20. Assign the slowest bus types $\text{bus}_b\text{type}_{bt}$ that still satisfies the throughput constraints
21. return Conf

Fig. 9. Heuristic cosynthesis algorithm.

Our cosynthesis heuristic is shown in Fig. 9. The basic idea is to add, one by one, reuse graph buffers to the (line 3) system and see if the mapping of the buffer to any of the (line 7) physical memories can decrease the (line 15) total energy consumption. If the insertion of the buffer decreases the total energy, the mapping is retained, and the next buffer is evaluated. All the buffers are examined on the order of their decreasing efficiency in terms of reducing traffic to the (line 3) higher level of memory hierarchy. For each buffer mapping, all the bridges are evaluated, one by one, for inclusion, and the (lines 9–14) ones that reduce the total energy are retained. We use the equations from the previous section (excluding the symmetry constraints) to calculate the bus traffic (used in line 20) and the total system energy consumption and to check if a particular configuration is valid.

The complexity of our heuristic is $\text{buf}_n * \text{proc}_n^3 * \text{bus}_n^4 * \text{mem}_t^2$, which is lower than that of the MILP approach.

E. Extensions

1) *Representing IPs Other than Processors and Memories:* The MILP approach and heuristic outlined earlier assume that the system being optimized consists of processors, memories, and interconnects. However, additional SoC IPs (e.g., peripherals) and the traffic between the processors and the IPs can easily be incorporated into the formulations by representing them as buffers/memories of a special type in the reuse graph. Custom processing elements that are used instead of processors can be represented as processors in our approach.

2) *Nonstreaming Traffic*: Sometimes, processors need to load data from memory in a nonstreaming fashion, e.g., in an unpredictable order or with the amount of transfer depending on the data. This traffic can be accounted for in our model by adding it to the reuse graph (as an array reference and an edge between the reference and the main memory or other buffer) using worst case estimation for the amount of traffic. The data should be prefetched to the local processor memory and read by the processor from there. If prefetching is not possible, e.g., if the addresses become known only during the run time immediately before the data is needed by the processor, a separate bus can be added to handle these high-priority unpredictable memory accesses. It is also possible to handle both streaming and nonstreaming traffics by the same bus using different arbitration priorities for these two kinds of traffic; however, in this case, the analytical calculation of system performance is not possible, and bus frequencies returned by our approaches may be underestimated if two or more processors generate such nonstreaming traffic on the same bus. Simulations are needed to ensure that the deadlines are met for any input data, and bus frequencies may need to be increased if violations are found.

3) *Dynamic Behavior*: Our approach uses a reuse graph as an input, which represents the reuse information for a particular sequence of memory accesses generated by the program. If the input program data change, the memory access pattern may also change, invalidating the reuse graph obtained for different input data. Luckily, for a typical streaming embedded application such as video encoders (and for others as well), the reuse graph changes only when the format (size of the frame) of the input data changes, and it remains the same when the input video sequence changes without changing its frame size.

When the embedded device has to be designed to accommodate different input formats, two approaches are possible. First, the system may be designed for a worst case, having large-enough buffers and fast-enough buses to accommodate the most demanding input sequence (video with the highest resolution in our example). However, additional power savings may be obtained by designing the system for two or more cases (or scenarios) and using a different set of buffers and the bus frequencies for each frame size. At the design time, our technique is run several times to obtain configurations for several reuse graphs corresponding to different frame sizes. At run time, the system is reconfigured every time a different frame size is selected. The number of different scenarios do not need to match the number of different frame sizes supported by the system; several different formats can be grouped together to use the settings for the same scenario.

4) *Using Our Approach With Existing Platforms*: Our techniques described in the previous sections provide the designer with the memory and communication architectures that match a given application. However, our approaches can also be used to adapt software to a preexisting fixed or reconfigurable SoC platform. To achieve this goal, the constraints should be added to the MILP formulation or our heuristic that fixes the architecture template to reflect the existing flexibility in the platform.

V. EXPERIMENTS

A. Experimental Setup

We applied our techniques to a number of typical streaming video and image processing applications, namely, Laplace, which is an image filtering algorithm; Susan, which is image recognition application; and QSDPCM, which is a video encoder [29]. The first benchmark was parallelized for 4 and 16 processors; the last two were correspondingly distributed over four and six processors.

For our energy models, we used data for 130-nm technology. The bus wires' and wire drivers' power consumptions were calculated according to [2], assuming a bus segment length of 0.5 mm. The energy for bus interfaces was taken from [34]. Memory power consumption was calculated and extrapolated based on the numbers from the MPARM simulator [19].

We modified the SimpleScalar functional simulator [3] to find the footprint and number of accesses to the data mapped to the local processors memories (all data accesses were taken into account except the ones to the arrays mapped to the main memory/SPMs). We used the approach described in Section III to obtain reuse graphs for the benchmarks. We implemented an MILP cosynthesis technique described in Section IV in a tool that automatically generates the MILP formulation, given the architecture parameters and reuse graph description. MILP problems were solved using the commercial MILP solver CPLEX 9.0 from ILOG [8].

B. Experimental Results: Comparison of Techniques

We evaluated our memory and communication cosynthesis approach using three sets of experiments. First, we applied our MILP-based memory/communication cosynthesis technique to the set of benchmarks. Then, we compared our MILP-based cosynthesis technique with a traditional approach where these two steps are separated. Finally, we used our heuristic and compared the execution time and quality of the results against the MILP approach.

Our first experiment applied our MILP-based memory/connectivity cosynthesis technique described in Section IV. We used two or three different SPM types and three bus types for each benchmark as our architecture parameters. The sizes of local processor memories were determined by the footprint of the data stored in them and were measured by profiling, using modified SimpleScalar. The results are shown in Table I. For each benchmark, the table shows the number of processors, the number of buffers in the reuse graph, the application period, the size of the picture (frame) that was processed, and the sizes of the local processor memories. The times it took the CPLEX tool to solve the problem and minimize the energy obtained are also included in the table. The energy includes the power dissipation in the main memory, SPMs, local processor memories, and connectivity subsystem.

The time that was required to solve the problems was reasonable, which is not in excess of 2 h for all designs with up to 16 processors. Note that this time is comparable with a simulation time of just one configuration when simulations are needed to check if the system meets design constraints

TABLE I
BENCHMARK PROPERTIES AND RESULTS FOR OUR
MILP-BASED COSYNTHESIS APPROACH

Benchmark	Laplace	Laplace	Susan	QSDPCM
# of processors	4	16	4	6
# of buffers in the reuse graph	10	34	9	21
Period, ms	20	25	20	20
Picture size	640x480	1920x1080	640x480	640x480
Local memory sizes	256x4	256x16	1Kx4	8K, 4Kx5
Total time to solve, min	3	47	2	120
Total energy, μ J	106	1078	370	713

TABLE II
RESULTS FOR MILP-BASED TWO-STEP SYNTHESIS APPROACH

Benchmark	Laplace-4	Laplace-16	Susan	QSDPCM
Total time to solve, min	0.4	6.5	0.2	2.0
Increase in total energy (in comparison with MILP)	25%	39%	13%	2%
Increase in communication energy	56%	71%	85%	13%

for systems designed by using buses with non-TDMA arbitration.

In our second experiment, we emulated the traditional synthesis approach done in two steps, namely, by first deciding on SPM configuration and then by performing connectivity synthesis. We used our MILP formulation for each step. Specifically, to find the energy-optimal SPM configuration, we used the same MILP problem described in Section IV but with a modified optimization criteria, specifically, we minimized the sum of the main and SPM energy consumption. To perform connectivity synthesis, we modified the MILP formulation generated by our tool by adding the constraints $ex_buf_b = 0$ for buffers that were not selected in the previous step and $ex_buf_b = 1$ for the buffers that were selected. The synthesis was performed by using the same architecture parameters as in the first approach.

Table II shows the results comparing the (row 2) obtained energy consumption of the best found solution, the (row 3) energy consumption of the communication subsystem, and the (row 1) total time to obtain the results with our MILP cosynthesis approach.

The results clearly show that while significantly reducing the synthesis time (up to 60 times), the total energy consumption is also increasing (up to 40%). If we consider only the communication subsystem energy, the increase is even more significant, which is 57% on average. This shows that the communication energy significantly depends on the selected memory hierar-

TABLE III
RESULTS FOR HEURISTIC APPROACH

Benchmark	Laplace-4	Laplace-16	Susan	QSDPCM
Total time to solve, min	0.1	19	0.1	0.5
Increase (in comparison with MILP) in total energy	15%	53%	0%	0.2%
Increase in communication energy	35%	70%	0%	2%

chy and that performing simultaneous memory/communication cosynthesis instead of separating these two steps allows for the reduction of the total energy consumption.

One of the reasons that a suboptimal configuration is selected in the two-step approach was that a shared memory appeared to be more power efficient only without considering additional power dissipated in a shared bus. Another reason was a decision to include some SPM buffers which reduce the total number of accesses to the main memory; however, the reduction in energy due to this was not enough to overcome the increased power dissipation in the bus wires and interfaces.

The last experiment evaluates our heuristic and compares it with our MILP cosynthesis approach.

Table III shows the results for the synthesis performed by the design heuristic that is also described in Section IV and contains the following: (row 1) total running time of the heuristic, (row 2) energy consumption comparison, and (row 3) communication energy comparison. Fig. 10 shows the energy results of all three techniques discussed earlier.

We note that the heuristic execution time was much smaller than the time to optimally solve the MILP problem and was on par with the time of two-step synthesis approach. However, it provided mixed results; for two of the benchmarks, it found an optimal or near-optimal solution (for Susan and QSDPCM) but the Laplace-16 solution was worse than for two-step synthesis.

This shows that if the time to optimally solve the MILP cosynthesis problem exceeds a reasonable one for a designer time limit, a combination of last two approaches (i.e., traditionally decoupled and heuristic) can be used.

C. Experimental Results: Exploration With Memory Size/Area Constraints

In this section, we explored the effect of memory area limitations on the memory and communication subsystem energy consumption. We used the Laplace benchmark and our MILP cosynthesis approach to perform the exploration. To allow for more variations in the synthesized architecture, we removed the symmetry constraints described in Section IV and added constraints that limit the total memory size. We performed experiments with several memory size limits. The obtained design points are shown in Fig. 11.

The figure shows the possibility of a tradeoff between the total memory size (area) and the memory/communication subsystem power consumption.

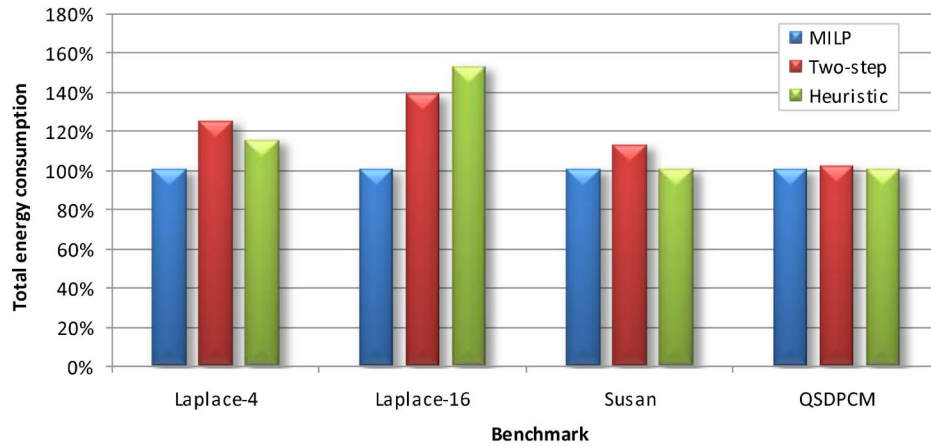


Fig. 10. Results obtained by different techniques.

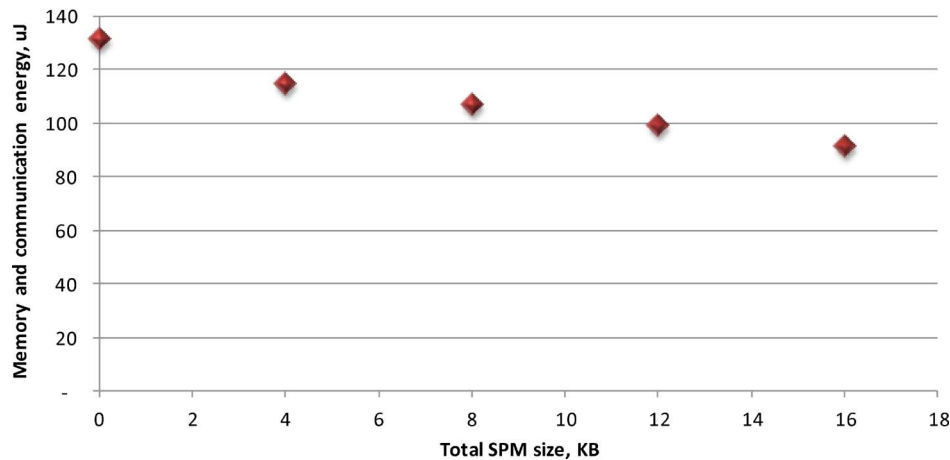


Fig. 11. Design space exploration results.

VI. CONCLUSION

The increasing use of MPSoCs places a big burden on system designers to evaluate customized memory and communication architectures that yield different power, cost, and performance tradeoffs. First, in this paper, we presented a novel multiprocessor data reuse analysis technique that opens up a large space of heretofore unexplored options for memory customization. Second, we presented several techniques aimed at memory and communication system synthesis. Traditionally, the memory and communication subsystems were designed and sequentially optimized, potentially missing out on a large number of good design points. In this paper, we proposed a novel approach for MPSoC memory/communication energy-aware cosynthesis based on data reuse information and an architecture communication template for architecture with hierarchical buses with TDMA arbitration. We proposed a template for a data-parallel partitioned application and suggested several ways to solve the cosynthesis problem—optimally by an MILP solver or sub-optimally by a heuristic. We compared these two approaches, as well as against a traditional two-step synthesis technique that first determines memory configuration and then performs communication synthesis. We showed that an optimal MILP solution takes a reasonable amount of time for systems with

up to 16 processors and provides results which are up to 50% (19% on average) better than the results from the other two approaches. Additionally, while providing 17%, on average (up to 53%), worse results than the optimal MILP technique, our heuristic achieves near-optimal results on some of the benchmarks with much smaller execution times. If the MILP solution does not terminate in an allotted amount of time, the designer can choose to use the best of the results obtained by our heuristic and the two-step technique.

ACKNOWLEDGMENT

Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending an e-mail to pubs-permissions@ieee.org. This is an expanded version of the authors' DAC 2006 and CODES-ISSS 2006 papers [13], [15]. The papers were expanded to include the discussion of different synchronization schemes for multiprocessor SPMs, discussion of how their approach can be applied to systems that have nonstreaming traffic, dynamic behavior, and predefined architecture templates; the authors also included more experimental results that explore the results of their cosynthesis approach applied to architectures with memory size constraints.

REFERENCES

- [1] Y. Aghaghi, F. Fallah, and M. Pedram, "Irredundant address bus encoding for low power," in *Proc. ISLPED*, 2001, pp. 182–187.
- [2] K. Banerjee and A. Mehrotra, "A power-optimal repeater insertion methodology for global interconnects in nanometer designs," *IEEE Trans. Electron Devices*, vol. 49, no. 11, pp. 2001–2007, Nov. 2002.
- [3] D. Burger and T. Austin, "The Simplescalar tool set," CS Dept., Univ. Wisconsin, Madison, WI, Tech. Rep. 1342, 1997, version 2.0.
- [4] L. Cai, H. Yu, and D. Gajski, "A novel memory size model for variable-mapping in system level design," in *Proc. Asia South Pacific Conf. Des. Autom.*, 2004, pp. 813–818.
- [5] F. Catthoor, K. Danckaert, K. Kulkarni, E. Brockmeyer, P. G. Kjeldsberg, T. Van Achteren, and T. Ommes, *Storage Management for Embedded Programmable Processors*. Norwell, MA: Kluwer, 2002.
- [6] J. Y. Chen, W. B. Jone, J. S. Wang, H.-I. Lu, and T. F. Chen, "Segmented bus design for low-power systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 1, pp. 25–29, Mar. 1999.
- [7] G. Chen, G. Chen, O. Ozturk, and M. Kandemir, "Exploiting inter-processor data sharing for improving behavior of multi-processor SoCs," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, May 2005, pp. 90–95.
- [8] *CPLEX ILP Solver*. [Online]. Available: www.cplex.com
- [9] M. Gasteier and M. Glesner, "Bus-based communication synthesis on system level," *ACM Trans. Design Autom. Electron. Syst.*, vol. 4, no. 1, pp. 1–11, Jan. 1999.
- [10] P. Grun, N. Dutt, and A. Nicolau, "Memory system connectivity exploration," in *Proc. Des. Autom. Test Eur. Conf.*, Paris, France, 2002, pp. 894–901.
- [11] J. Guo, A. Papanikolaou, P. Marchal, and F. Catthoor, "Physical design implementation of segmented buses to reduce communication energy," in *Proc. Asia South Pacific Conf. Des. Autom.*, 2006, pp. 42–47.
- [12] *International Technology Roadmap for Semiconductors 2005 Edition*. System Drivers, Interconnect.
- [13] I. Issenin, E. Brockmeyer, B. Durinck, and N. Dutt, "Multiprocessor system-on-chip data reuse analysis for exploring customized memory hierarchies," in *Proc. Des. Autom. Conf.*, 2006, pp. 49–52.
- [14] I. Issenin and N. Dutt, "FORAY-GEN: Automatic generation of affine functions for memory optimizations," in *Proc. DATE*, Munich, Germany, 2005, pp. 808–813.
- [15] I. Issenin and N. Dutt, "Data reuse driven energy-aware MPSoC co-synthesis of memory and communication architecture for streaming applications," in *Proc. of CODES+ISSS*, 2006, pp. 294–299.
- [16] I. Issenin and N. Dutt, "DRDU: A data reuse analysis technique for efficient scratch pad memory management," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 2, p. 15, Apr. 2007.
- [17] S. Kim, C. Im, and S. Ha, "Efficient exploration of on-chip bus architectures and memory allocation," in *Proc. CODES+ISSS*, 2004, pp. 248–253.
- [18] D. Lyonard, S. Yoo, A. Baghdadi, and A. Jerraya, "Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip," in *Proc. Des. Autom. Conf.*, 2001, pp. 518–523.
- [19] *MPARM Project*. [Online]. Available: <http://www-micrel.deis.unibo.it/sitonew/research/mparm.html>
- [20] O. Ozturk, M. Kandemir, G. Chen, M. Irwin, and M. Karakoy, "Customized on-chip memories for embedded chip multiprocessors," in *Proc. Asia South Pacific Conf. Des. Autom.*, 2005, pp. 743–748.
- [21] P. Panda, N. Dutt, and A. Nicolau, "Efficient utilization of scratch-pad memory in embedded processor applications," in *Proc. Des. Autom. Test Eur. Conf.*, Paris, France, 1997, pp. 7–11.
- [22] A. Papanikolaou, K. Koppenberger, M. Miranda, and F. Catthoor, "Memory communication network exploration for low-power distributed memory organisations," in *Proc. IEEE Workshop Signal Process. Syst.*, 2004, pp. 176–181.
- [23] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Automated throughput-driven synthesis of bus-based communication architectures," in *Proc. Asia South Pacific Conf. Des. Autom.*, 2005, pp. 495–498.
- [24] S. Pasricha and N. Dutt, "COSMECA: Application specific co-synthesis of memory and communication architectures for MPSoC," in *Proc. Des. Autom. Test Eur. Conf.*, 2006, pp. 1–6.
- [25] A. Pinto, L. Carloni, and A. Sangiovanni-Vincentelli, "Constraint-driven communication synthesis," in *Proc. Des. Autom. Conf.*, 2002, pp. 783–788.
- [26] K. Ryu and V. Mooney, "Automated bus generation for multiprocessor SoC design," in *Proc. Des. Autom. Test Eur. Conf.*, 2003, pp. 282–287.
- [27] Sonics Inc. [Online]. Available: <http://www.sonicsinc.com/sonics/products/siliconbackplaneIII/>
- [28] S. Steinke, L. Wehmeyer, B. Lee, and P. Marwedel, "Assigning program and data objects to scratchpad for energy reduction," in *Proc. Des. Autom. Test Eur. Conf.*, Paris, France, 2002, pp. 409–415.
- [29] P. Stobach, "A new technique in scene adaptive coding," in *Proc. EUSIPCO*. Grenoble, France, 1988, pp. 1141–1144.
- [30] D. Sylvester and K. Keutzer, "Impact of small process geometries on microarchitectures in systems on a chip," *Proc. IEEE*, vol. 89, no. 4, pp. 467–489, Apr. 2001.
- [31] S. Udayakumar and R. Barua, "Compiler-decided dynamic memory allocation for scratch-pad based embedded systems," in *Proc. Int. Conf. Compilers, Architecture Synth. Embed. Syst.*, 2003, pp. 276–286.
- [32] M. Verma, L. Wehmeyer, and P. Marwedel, "Dynamic overlay of scratch-pad memory for energy minimization," in *Proc. CODES*, Stockholm, Sweden, 2004, pp. 104–109.
- [33] H. Wang, A. Papanikolaou, M. Miranda, and F. Catthoor, "A global bus power optimization methodology for physical design of memory dominated systems by coupling bus segmentation and activity driven block placement," in *Proc. Asia South Pacific Des. Autom. Conf.*, 2004, pp. 759–761.
- [34] P. Wolkotte, G. Smit, and J. Becker, "Energy-efficient NoC for best-effort communication," in *Proc. Int. Conf. Field Program. Logic Appl.*, 2005, pp. 197–202.
- [35] H. Zhang and J. Rabaey, "Low-swing interconnect interface circuits," in *Proc. Int. Symp. Low-Power Electron. Des.*, 1998, pp. 161–166.



Ilya Issenin (M'97) received the degree in electrical engineering from the Moscow Engineering Physics Institute, Moscow, Russia, in 1998, and the M.S. and Ph.D. degrees in information and computer science from the University of California, Irvine, in 2001 and 2007, respectively.

He is currently with Teradek, Irvine, CA, working on low-power video designs. He is the author of a number of papers in his areas of research interest, which include memory and communication subsystems optimization and synthesis and low-power architecture optimization techniques.



Erik Brockmeyer did his master thesis on MPEG-4 in the System Exploration for Memory and Power group in 1997. This group is part of the Nomadic Embedded Systems Division, Interuniversity MicroElectronics Center (IMEC), Leuven, Belgium. He received the M.S. degree in electrical engineering from the University of Eindhoven, Eindhoven, The Netherlands, in 1998.

Since 1997, he has been with IMEC, where he worked on the automation of various steps of the Data Transfer and Storage Exploration (DTSE) script. His past work focused on efficiently mapping an application to the multiple levels of a memory hierarchy. His current research interest is focused on multiprocessor systems with a shared distributed memory subsystem.



Bart Durinck received the M.Sc. degree in computer sciences from the Univeriteit Gent, Gent, Belgium, in 1997.

He was with ICOS Vision Systems, Belgium, with the main task of developing DSP image processing and real-time system software. Since 2005, has also been with ARM Belgium, Leuven, working on low-power mobile audio codecs and 3-D graphics.



Nikil D. Dutt (S'84–M'89–SM'96–F'08) received the Ph.D. degree in Computer Science from the University of Illinois, Urbana, in 1989.

He is currently a Chancellor's Professor with the University of California, Irvine, with academic appointments in the Center for Embedded Computer Systems, Department of Computer Science Donald Bren School of Information and Computer Sciences and in the Department of Electrical Engineering and Computer Science, The Henry Samueli School of Engineering. His research interests are embedded

systems, electronic design automation, computer architecture, optimizing compilers, system specification techniques, distributed embedded systems, and formal methods.

Dr. Dutt was an ACM Special Interest Group on Design Automation (SIGDA) distinguished lecturer during 2001–2002 and an IEEE Computer Society distinguished visitor for 2003–2005. He has served on the steering, organizing, and program committees of several premier computer-aided design (CAD) and embedded system design conferences and workshops, including the Asia and South Pacific Design Automation Conference; the International Conference on Compilers Architecture, and Synthesis for Embedded Systems; CODES+ISSS; Design, Automation, and Test in Europe; International Conference on CAD; the International Symposium on Low Power Electronics and Design; and the conference on Languages, Compilers, and Tools for Embedded Systems. He serves or has served on the advisory boards of ACM Special Interest Group on Embedded Systems and ACM SIGDA, and has previously served as the vice chair of ACM SIGDA and of the International Federation for Information Processing (IFIP) workgroup 10.5. He is an ACM distinguished scientist and an IFIP Silver Core awardee. He received the Best Paper Awards at the Conference on Computer Hardware Description Languages and Their Applications (CHDL)89, CHDL91, VLSIDesign2003, CODES+ISSS 2003, IEEE Consumer Communications and Networking Conference 2006, and ASPDAC-2006. He currently serves as the Editor-in-Chief of the Association for Computing Machinery (ACM) *Transactions on Design Automation of Electronic Systems* and as the Associate Editor of ACM *Transactions on Embedded Computer Systems* and of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS.