Data Reuse Driven Energy-Aware MPSoC Co-Synthesis of Memory and Communication Architecture for Streaming Applications [*]

Ilya Issenin
University of California,
Irvine, CA 92697
isse@ics.uci.edu

Nikil Dutt
University of California,
Irvine, CA 92697
dutt@ics.uci.edu

ABSTRACT

The memory subsystem of a complex multiprocessor systems-on-chip (MPSoC) is an important contributor to the chip power consumption. The selection of memory architecture, as well as of communication architecture, both affect the power efficiency of the design. In this paper we propose a novel approach that enables energy-aware co-synthesis of both memory and communication architecture for streaming applications. As opposed to earlier techniques, we employ a powerful compile-time analysis of memory access behavior that adds flexibility in selecting memory architectures. Additionally, we target TDMA bus-based communication architectures, which not only guarantee performance, but also greatly reduce the design time and allow us to find the energy optimal system configuration. We propose and compare three techniques: an optimal mixed ILP-based co-synthesis technique, a mixed ILP-based traditional two-step synthesis approach where memory and communication synthesis is performed sequentially, and a co-synthesis 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 proposed co-synthesis approach. However, on some of the streaming benchmarks, our co-synthesis heuristic approach was able to find optimal or near-optimal results in a much shorter time than the MILP co-synthesis approach.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems.

General Terms

Algorithms, Performance, Design.

Keywords

Communication synthesis, hierarchical TDMA buses, data reuse, customized memory hierarchy, multiprocessor system-on-chip.

1. INTRODUCTION

Multiprocessor Systems-on-Chips (MPSoCs) are becoming a popular solution to meet the growing processing demands of embedded applications. In such designs, the memory and communication architecture play an important role in meeting both the performance and energy consumption requirements of applications executed on MPSoCs.

Traditionally, memory subsystem synthesis consists of two steps. First, the memory architecture (physical memories as well as 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, however, attempt to adjust memory architectures while performing communication synthesis. In all these works, the memory architecture transformations were very simplistic: e.g., merging several memories into one [12], or splitting the memories and removing the ones that are used by only one processor from the shared bus and making them private [13]. In this work, we propose to use MPSoC data reuse analysis [8] to find a set of possible buffers that 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 (which form a *reuse graph* [8]) we select and map some of these buffers into physical memories and perform communication synthesis for minimal power simultaneously while meeting given time constraints.

Another distinctive feature of our work is that unlike in most of the previous works, we are targeting hierarchical TDMA-bus based communication system. The use of buses with TDMA arbitration (e.g., Sonics SiliconBackplane III [16]) is beneficial for streaming applications since 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, since there are no collisions on 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 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 work we propose an optimal mixed ILP-based approach for memory/communication architecture co-synthesis based on a *communication (or architecture) template*, which is described in Section 3. We compare this approach with a

[*] This work was partially supported by NSF grant CCR-0203813.

traditional two-step synthesis approach (first memory, then communication synthesis) and also with a simple co-synthesis heuristic. We show that MILP-based optimal co-synthesis approach provides results that outperform two-step or heuristic approach in a reasonable amount of time for the benchmarks we used. However, our heuristic, being much less computationally expensive than MILP co-synthesis approach, achieves near-optimal results on some of the benchmarks, which makes it a good candidate for solving the problems for which the MILP co-synthesis technique is not able to produce the results in a reasonable amount of time.

2. RELATED WORK

There is a significant amount of research in the area of bus-based communication synthesis.

A number of works presented methodologies for automated bus generation. [10],[15] present frameworks that generate custom bus systems using predefined IP cores. Pinto et al. [14] proposed an algorithm for topology synthesis given a placement of components together with communication requirements. Gasteier et al. [7] proposed communication synthesis approach for a system based on shared buses. Pasricha et al. [13] presented a heuristic for throughput-driven communication synthesis. All of the abovementioned works perform design optimizations for cost/area or performance, but not for energy.

In bus optimization research aimed at energy reduction, Aghaghiri et al. [1] proposed to use bus encoding. Zhang et al. [20] studied low-swing bus interface implementations. Chen et al. [5] proposed bus segmentation to reduce power consumption. Wang et al. [18] proposed to perform floorplanning placing communication-intensive modules closer to each other.

In the area of memory and communication architecture synthesis, the co-synthesis aspect was mostly ignored and memory allocation and mapping was performed before communication synthesis [4]. Only few techniques consider memory and communication co-synthesis. Kim et al. [9] perform mapping of memories to buses along with bus topology selection. Pasricha et al. [12] perform bus matrix communication synthesis together with limited memory transformations, optimizing for number of buses and memory area.

Our work is significantly different from previous efforts in that we *simultaneously* perform memory/connectivity co-synthesis for hierarchical bus architectures that employ *highly customized* memories while aiming at system *energy* reduction. Since we are targeting buses with TDMA arbitration, we are also able to find the optimal energy configuration with guaranteed performance.

3. ARCHITECTURE TEMPLATE FOR MPSOC DATA PARALLELIZED APPLICATIONS

There are many different interconnect topologies for MPSoCs, that 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, design requirements and other factors. In this paper we perform 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 work is shown in Figure 1 and is configured by three parameters: the number of processors, number of levels of bus hierarchy and number of different scratch pad memory types (sizes) at each bus level. These parameters 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 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 Figure 1) 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 Figure 1. Note that there are no memories allowed in the template instance above the *horizontal bus connection* on processors 2..N. The bridges constrain the propagation of useless traffic to higher levels of bus hierarchy and thus allow savings in energy by reducing the number of transactions and the maximum required 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 [5].

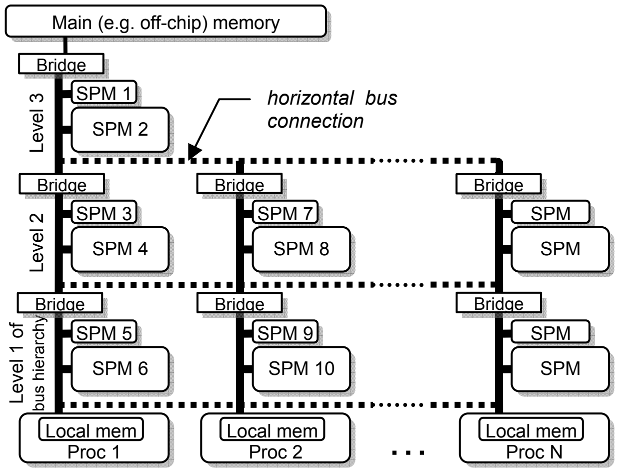

Figure 1. Example of the template for N processors, 3 bus levels and 2 memory types.

Our proposed algorithms (described in Sections 6 and 7) 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*.

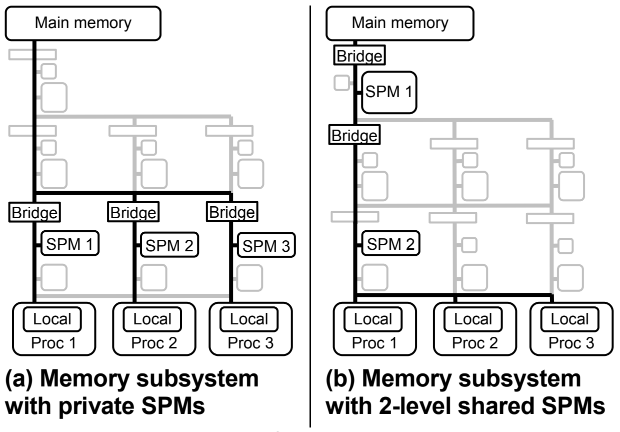

Figure 2. Examples of customized memory subsystems for 3 processors.

The data are moved between SPMs, main memory and local processor memories by DMA engines, which are not shown in the Figure 1. 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. This allows 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.

Figure 2 shows some of the typical custom memory subsystems that can be obtained by customizing a three-processor template. The gray components in the picture are those that were not selected for implementation. The figure shows that the template is general enough to describe any architecture consisting of combination of shared and private SPMs as long as connections are symmetric (e.g., there can not be shared SPM which is accessible by only some of processors) and is well suited for homogeneous MPSoCs. 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. Such partitioning is often used in data streaming applications, when a particular task cannot be mapped into a single processor due to performance or energy efficiency constraints. However, applications partitioned into heterogeneous platforms can also use the same template (e.g., one of our benchmarks, QSDPCM encoder, is parallelized into 3 pipelined stages, with only one of the stages using 4 processors for data-parallel (homogeneous) processing).

4. DATA REUSE GRAPHS

The memory/communication architecture synthesis process critically needs information about application memory access behavior. For this purpose we employ *reuse graphs* produced by our multiprocessor data reuse analysis technique [8]. Reuse graphs are built using the information about application memory request patterns. Reuse graphs are hierarchies of buffers, with each buffer holding the data that is requested several times by processor(s). Each buffer can be mapped to a physical SPM or can be omitted. The technique described in [8] also provides a way to modify the application code to add the necessary DMA transfers to update the buffer if it is selected to be placed in SPM.

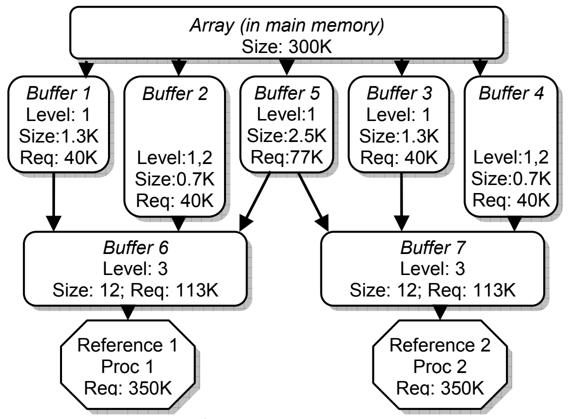

Figure 3. Example of a multiprocessor data reuse graph.

Figure 3 shows an example of *reuse graph* for one of the array references for the Laplace benchmark parallelized for two processors. It has six private buffers (1-4, 6, 7) and one shared buffer (5). Buffers 6 and 7 are private and small, so they are placed in local processor memories. The data reuse analysis tool [8] provides the following information for each buffer: its level (at which nesting loop level the updates of the buffer should take place); and size and number of requests (traffic) to higher level of memory hierarchy (a buffer if exists or the main memory). 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.

5. ENERGY MODELS

Recall that the goal of our co-synthesis 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:

$$E_{total} = E_{SPM} + E_{main\_mem} + E_{bus} + E_{bridge} \quad (1)$$

where E_{SPM} is the energy consumed by scratch pad memories; E_{main_mem} is the energy consumed by main memory; E_{bus} is the energy consumed in busses; and E_{bridge} is the energy consumed in bridges. These values are defined as:

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

$$E_{main\_mem} = E_{acc} * N_{main\_mem} \quad (3)$$

where E_{acc} is the memory energy per access; N_{acc} is the number of accesses to the SPM; and N_{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.

$$E_{bus} = \sum (E_{bus\_active} * N_{trans} + E_{bus\_static} * N_{bus\_cycles}) \quad (4)$$

where E_{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_{bus_static} is the power consumption of a bus per bus clock cycle (e.g., in the bus interfaces as well as in the bus clock wires); and N_{bus_cycles} is the number of bus clock cycles in the application period. Since both E_{bus_active} and E_{bus_static} may depend on the number of bus interfaces present on the bus (due to power dissipation in the interfaces as well as change in the bus capacitance), we have accounted for that as well:

$$E_{bus\_active} = E_{b\_a\_const} + E_{b\_a\_coef} * N_{bi} \quad (5)$$

$$E_{bus\_static} = E_{b\_s\_const} + E_{b\_s\_coef} * N_{bi} \quad (6)$$

where N_{bi} is the number of bus interfaces on particular bus; and $E_{b\_a\_const}$, $E_{b\_a\_coef}$, $E_{b\_s\_const}$, $E_{b\_s\_coef}$ are some coefficients.

The energy spent in bridges is calculated as

$$E_{bridge} = \sum E_{br} * N_{tr\_br} \quad (7)$$

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

6. MIXED ILP SYNTHESIS APPROACH

We now describe a mixed integer linear programming (MILP) approach for optimally solving the problem of communication/memory subsystem co-synthesis that finds a configuration of the architecture template described in Section 3 (i.e., location of *horizontal bus connection*, selecting types of the buses that satisfy throughput requirements defined by memory traffic, selecting which 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:

Minimize E under constraints C

where E is the objective function described in Section 6.4, and C are the problem constraints (Section 6.5).

Due to the lack of space we omit the exact formulation of some of the MILP constraints; however, we explain the meaning of the constraints that is in most cases enough to reconstruct the constraints.

6.1. Architecture template parameters

First, we define the architecture template parameters (Section 3) that are specified by the designer:

proc_n: the number of the processors;

bus_n: the number of hierarchical buses in the template (3 for the example in Figure 1);

mem_t: the number of different types of physical memories (of different sizes) that can be used as SPMs;

bus_t: the number of different types of buses that can be used in the system;

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

$E_{b\_a\_const\_i}$, $E_{b\_a\_coef\_i}$, $E_{b\_s\_const\_i}$, $E_{b\_s\_coef\_i}$, E_{br}: energy consumption parameters (described in Section 5) for the bus type *i*;

T_{acc_i}: access time of the memory of type *i*;

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

E_{acc_i}: energy consumption per access for the memory of type *i*;

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

6.2. 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;

buf_n: total number of buffers in the reuse graph that can be mapped to SPMs;

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*;

6.3. Variables

Below are the binary variables that describe the template configuration (Section 3) and reuse graph buffers mapping. These are the values that are determined by solving the MILP problem.

ex_link_i: horizontal bus connection is at bus level *i* (*$ex\_link_i = 1$*) or at other level (*$ex\_link_i = 0$*); *$1<=i<=bus\_n$*; the *horizontal bus connection* can be at only one bus level;

$ex\_br_i proc_j$: bridge at bus level *i* at processor *j* exists (1) or doesn't exist (0);

$buf_i mem_j bus_k proc_p$: reuse graph buffer *i* is mapped to SPM of type *j* at bus level *k* of processor *p*;

$bus_i type_j proc_k$: bus *i* on processor *k* is of type *j*.

6.4. 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 using Equation (1) in Section 5.

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

$k = f*b,\ b \in [0,1],\ 0<=f<=fmax$,

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

$$\begin{cases} k \le f \\ k \le b * fmax \\ k \ge f - fmax(1-b) \\ k \ge 0 \end{cases}$$

6.5. MILP Constraints

The constraints can be divided into several groups, which are briefly described here.

- ***Constraints related to mapping of buffers to SPMs:***
  - Each buffer from the reuse graph is assigned to zero or one physical memory;
  - Two buffers that belong to the same level in the reuse graph and hold data for the same processor are mutually exclusive;
  - Any two buffers, which are in parent-child relation in the reuse graph, should be mapped so that memory with the parent buffer is on the same bus level or higher in the bus hierarchy than the memory with the child buffer;
  - All shared buffers in the reuse tree should be mapped to the memories on the same bus level as the horizontal bus connection or higher;
  - All the buffers that are mapped to the memories that are below horizontal bus connection, should be private and be located on the proper processor bus;
  - There should be no memories or bridges on the buses at the levels higher than the one of horizontal bus connection at processors *2..proc_n*.
- ***Constraints related to the design requirements:***
  - Buffers mapped to the same physical memory should fit into it;
  - The time needed to perform all accesses to physical memories should be less than the application period;
  - 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 configuration of the memories, buses and bridges is the same (symmetric) for processors 2*..proc_n*. This is typically true since the code executed on processors and the volume of the data they process are approximately the same for application 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 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. For example, for the reuse graph shown in Figure 3, the number of accesses to buffer 1 is 113K if buffer 6 is present and 350K if buffer 6 is not selected for implementation.

- ***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 bridge is not present.

The Mixed ILP problem we formulated was solved using commercial ILP solver CPLEX [6] as described in Section 8.

6.6. Extensions

The MILP formulation outlined above currently represents 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 formulation by representing them as buffers/memories of a special type. If the application is implemented using custom hardware instead of processors, they can be represented as processors in our approach.

7. HEURISTIC-BASED SYNTHESIS APPROACH

ILP formulations typically do not scale well with problem size.

Consequently, we devised a simple greedy heuristic for memory/communication co-synthesis that has much shorter execution times and scales better with 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
$buf_b mem_m$, $ex\_br_{br}$, $horiz\_bus\_connection\_level$, $bus_b type_{bt}$) $Conf$

1. Initialize $buf_b mem_m = 0$ for all $b, m$
2. Initialize $ex\_br_{br} = 0$ for all $br$
3. for each $b \in B$ in the order of decreasing metric
   $(buf_b in - buf_b req)/buf\_size_b$ do {
4.    Calculate total energy $E$ for the current configuration
5.    for each $m \in M$ do {
6.       for $horiz\_bus\_connection\_level = 1..bus\_n$ do {
7.          $buf_b mem_m = 1$; $buf_b mem_{mm} = 0$ 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.            $ex\_br_{br}$ = 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 ($E_1 < E$ 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 $bus_b type_{bt}$ that still satisfies the
   throughput constraints
21. return $Conf$
```

Figure 4. Heuristic co-synthesis algorithm.

Our co-synthesis heuristic is shown in Figure 4. The basic idea is to add reuse graph buffers to the system one by one (Line 3) and see if the mapping of the buffer to any of physical memories (Line 7) can decrease the total energy consumption (Line 15). 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 in the order of their decreasing efficiency in terms of reducing traffic to the higher level of memory hierarchy (Line 3). For each buffer mapping, all the bridges are evaluated for inclusion one by one, and the ones that reduce the total energy are retained (Lines 9-14). We use the equations in Section 6 (excluding symmetry constraints) to calculate the bus traffic (used in Line 20), total system energy consumption and check if a particular configuration is valid.

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

8. EXPERIMENTS

8.1. Experimental Setup

We applied our techniques to a number of typical streaming video and image processing applications: *Laplace*, which is an image filtering algorithm, *Susan*, image recognition application, and *QSDPCM*, a video encoder [17]. The first benchmark was parallelized for 4 and 16 processors; the last two were distributed over 4 and 6 processors correspondingly.

For our energy models, we used data for 130nm technology. Bus wires and wire drivers power consumption were calculated according to [2] assuming bus segment length of 0.5mm. The energy for bus interfaces was taken from [19]. Memory power consumption was calculated and extrapolated based on the numbers from the MPARM simulator [11].

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 [8] to obtain reuse graphs for the benchmarks. Mixed ILP problems were solved using the commercial MILP solver CPLEX 9.0 from ILOG [6]. Since the ILP formulations are very large (the sizes of text files with the MILP problems are 250K-4M for our benchmarks), we created a tool that automatically generates the MILP formulation, given the architecture parameters and reuse graph description.

8.2. Experimental Results

We evaluated our memory and communication co-synthesis approach using three sets of experiments. First, we applied our mixed ILP-based memory/communication co-synthesis technique to the set of benchmarks. Then, we compared our MILP-based co-synthesis 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.

Table 1. Results for our MILP-based co-synthesis 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 mem size	256x4proc	256x16	1Kx4	8K, 4Kx5
Total time to solve, min	25	47	2	120
Total energy, µJ	106	1078	370	713

Our first experiment applied our MILP-based memory/ connectivity co-synthesis technique described in Section 6. We used 2 or 3 different scratch pad memory types and 3 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 1. For each benchmark, it shows the number of processors, the number of buffers in the reuse graph, application period, the size of the picture (frame) that was processed and the sizes of the local processor memories. The time it took the CPLEX tool to solve the problem and minimized 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: not exceeding two hours 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 for systems designed using buses with non-TDMA arbitration.

In our second experiment we emulated the traditional synthesis approach done in two steps: first deciding on scratch pad memory configuration, and then 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 6 but with a modified optimization criteria: we tried to minimize the sum of main and scratch pad memory energy consumption. To perform connectivity synthesis, we modified the MILP formulation generated by our tool by adding constraints $ex\_buf_b=0$ for buffers that were not selected in previous step and $ex\_buf_b=1$ for the buffers that were selected. The synthesis was performed using the same architecture parameters as in the first approach.

Table 2. Results for MILP-based two-step synthesis approach

Benchmark	*Lapl-4*	*Lapl-16*	*Susan*	*QSDPCM*
Total time to solve, min	0.4	6.5	0.2	2.0
Increase in total energy	25%	39%	13%	2%
Increase in communication energy	56%	71%	85%	13%

Table 2 shows the results comparing with our MILP co-synthesis approach the energy consumption of the best found solution (Row 2), the energy consumption of the communication subsystem (Row 3) and total time to obtain the results (Row 1).

The results clearly shows 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 communication subsystem energy, the increase is even more significant, 57% on average. This shows that communication energy significantly depends on the selected memory hierarchy and that performing simultaneous memory/communication co-synthesis instead of separating these two steps allows reduction of the total energy consumption.

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

The last experiment evaluates our heuristic and compares it with our MILP co-synthesis approach.

Table 3. Results for heuristic approach

Benchmark	*Lapl-4*	*Lapl-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%

Table 3 shows the results for the synthesis performed by the design heuristic described in Section 7, and contains the following: total running time of the heuristic (Row 1), energy consumption comparison (Row 2) and communication energy comparison (Row 3).

We note that the heuristic execution time was much smaller that 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 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 co-synthesis problem exceeds a reasonable for a designer time limit, a combination of last two approaches (i.e. traditionally decoupled and heuristic) can be used.

9. CONCLUSIONS

In this work proposed a novel approach for MPSoC memory/communication energy-aware co-synthesis based on data reuse information and architecture communication template for an architecture with hierarchical buses with TDMA arbitration. We proposed a template for data parallel partitioned application and suggested several ways to solve the co-synthesis problem: optimally by an MILP solver or suboptimally by a heuristic. We compared these two approaches, as well as a traditional two-step synthesis technique that determines memory configuration first, 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 and can be used (coupled with the two-step technique) for quick estimations or for the problems for which the optimal MILP technique requires unacceptable amount of time.

Future work will investigate the use of our co-synthesis approach on other architecture templates that may be useful for applications with different parallelization schemes.

REFERENCES

[1] Y. Aghaghiri et al. Irredundant address bus encoding for low power. In Proc. ISLPED 2001.

[2] K. Banerjee, A. Mehrotra. A Power-Optimal Repeater Insertion Methodology for Global Interconnects in Nanometer Designs. IEEE Trans. on Electron Devices, vol. 49, no. 11, Nov 2002.

[3] D. Burger and T. Austin. The SimpleScalar tool set, version 2.0. TR 1342, University of Wisconsin-Madison, CS Dept., 1997.

[4] L. Cai et al. A Novel Memory Size Model for Variable-Mapping In System Level Design. ASP-DAC 2004.

[5] Chen et al. Segmented Bus Design for Low-Power Systems. IEEE Trans. on VLSI Systems, vol.7, no.1, 1999.

[6] CPLEX ILP solver, www.cplex.com.

[7] M. Gasteier, M. Glesner. Bus-based communication synthesis on system level. In ACM TODAES, Jan. 1999.

[8] I. Issenin, E. Brockmeyer, B. Durinck, N. Dutt. Multiprocessor System-on-Chip Data Reuse Analysis for Exploring Customized Memory Hierarchies. In proc. of DAC, 2006.

[9] S. Kim et al. Efficient Exploration of On-Chip Bus Architectures and Memory Allocation. CODES+ISSS, 2004

[10] D. Lyonnard et al. Automatic generation of application-specificarchitectures for heterogeneous multiprocessor system-on-chip. In Proc. of DAC, 2001.

[11] MPARM project, http://www-micrel.deis.unibo.it/sitonew/research/mparm.html

[12] S. Pasricha, N. Dutt. COSMECA: Application Specific Co-Synthesis of Memory and Communication Architectures for MPSoC. In Proc. DATE 2006.

[13] S. Pasricha et al. Automated Throughput-driven Synthesis of Bus-based Communication Architectures. ASPDAC 2005.

[14] A. Pinto et al. Constraint-driven communication synthesis. In Proc. DAC 2002.

[15] K. K. Ryu, V. J. Mooney III. Automated Bus Generation for Multiprocessor SoC Design. In Proc. of DATE 2003.

[16] Sonics Inc. http://www.sonicsinc.com/sonics/products/siliconbackplaneIII/

[17] P. Stobach. A new technique in scene adaptive coding. In Proc. of EUSIPCO, Grenoble, 1988.

[18] H. Wang et al. Global Bus Power Optimization Methodology for Physical Design of Memory Dominated Systems by Coupling Bus Segmentation and Activity Driven Block Placement. In Proc. ASP-DAC 2004.

[19] P. Wolkotte et al. Energy-Efficient NoC for Best-Effort Communication. ICFPLA, 2005.

[20] H. Zhang, J. Rabaey. Low-swing interconnect interface circuits. In Proc. ISLPED, Aug. 1998.