Chapter 1

ENERGY-EFFICIENT SHARED MEMORY ARCHITECTURES FOR MULTI-PROCESSOR SYSTEMS-ON-CHIP

Kimish Patel
Alberto Macii
Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy

Massimo Poncino
Università di Verona
Dipartimento di Informatica
Verona, Italy

Abstract Most current multi-processor systems-on-chip (MPSoC) platforms do rely on a shared-memory architectural paradigm. The shared memory, typically used for storage of shared data, is a significant performance bottleneck because it requires explicit synchronization of memory accesses which can potentially occur in parallel.

Multi-port memories are a widely-used solution to this problem; they allow these potentially parallel accesses to occur simultaneously. However, they are not very energy-efficient, since their performance improvement comes at an increased energy cost per access.

We propose an energy-efficient architecture for the shared memory that can be used as an alternative to multi-port memories, and combines their performance advantage with a much smaller energy cost. The proposed scheme is based on the application-driven partitioning of the shared address space into a multi-bank architecture. This optimization can be used to quickly explore different power-performance tradeoffs, thanks to simple analytical models of performance and energy.

Experiments on a set of parallel benchmarks show energy-delay product (EDP) savings of 50% on average, measured on a set of standard parallel benchmarks.

Keywords: Multi-Processor Systems, Shared Memory, Systems-on-Chip.
1. Introduction

Modern design paradigms for MPSoCs are pushing towards architectures which are fully distributed and that work as general networks, based on a modular layered architecture, and that are able to support non-deterministic communications. Such architectures, called Networks-on-Chips (NoCs) [1], have been devised as an answer to the scaling of SoC complexity, especially in terms of the increased number of hosted processing elements, and of the decreased reliability of the communication medium.

In spite of these scalability challenges, most current SoCs are still based on a shared-medium architecture, and, consequently on a shared-memory paradigm. One reason for this slow migration to more complex architectures is cost. Shared on-chip buses represent a convenient, low-overhead interconnection, and they do not require special handling during the physical design flow. Another reason is a consequence of the limited support provided by system software for such architectures. Although current silicon technology allows to build SoCs with a large number of embedded cores, the capabilities currently offered by the embedded software (e.g., in terms of OS primitives) does not allow to fully exploit all the potential computational power; therefore, most implementations of SoC consist of few (seldom more than 16) processor cores, for which a shared interconnect is perfectly suitable.

The architecture of these MPSoC platforms is thus reminiscent of traditional multi-processor systems, where inter-processor communication and/or synchronization is provided through the exchange of data through shared memories of different types. Generally speaking, accessing the shared memories are significantly slower than accesses to local ones. First, they are placed farther away from the processors than private memories; in fact, the latter are often tightly coupled to the cores by means of dedicated local buses, while shared memories are forcedly connected to a shared bus. Moreover, accesses to the shared buses by the processors requires some form of arbitration, which may require the insertion of wait cycles in case of simultaneous accesses. As a consequence, the shared memories tend to become a major bottleneck for the bandwidth of the overall system, especially for applications in which parallelism is built around shared data.

Caching of shared data might be a solution, but it raises the well-known issue of cache coherence, i.e., the possible inconsistency between data stored in caches of different processors. Cache coherence can be solved in hardware, yet with an extra overhead that may not be affordable in small-scale, low-cost SoC as those considered in this work. Software-
based cache coherence is also a viable solution, but it essentially consists of limiting the caching of shared data to safe times [2]. For applications in which parallelism is built around shared data, this basically amounts to avoid caching of shared data. In this paper, this will be our assumption: all accesses to shared data will always imply an access to the shared memory.

Providing sufficient memory bandwidth to sustain fast program execution and data communication/transfers is mandatory for most embedded applications. Increasing memory bandwidth can be achieved by making use of different types of on-chip embedded memories, which provide shorter latencies and wider interfaces [3–5]. One typical solution used to match the computational bandwidth with that of memory is to use multi-port memories. This solution increases the sustainable bandwidth by construction, since a P-port memory allows in fact up to P accesses in parallel (i.e., in a single memory cycle). Therefore, by properly choosing the number of ports of the memory versus the number of processors, the issue of synchronization of simultaneous accesses can be easily solved.

The adoption of multi-port memories, however, comes at the price of a significant increase in area, wiring resources, and energy consumption. On the other hand, architectures based on multi-port memories seem to be the only viable option in the cases where bandwidth optimization has absolute priority.

In this work we propose an alternative architecture for the shared memory which combines the advantages, in terms of bandwidth, of the multi-port approach, with the advantages, in terms of energy consumption and access time, of partitioned memories [5]. We propose the use of small, single-port memory blocks as a way to achieve memory bandwidth increase together with low energy demand. In our scheme, the memory addressing space is mapped over single-port banks that can be simultaneously accessed by different processors, so as to mimic for a large fraction of the execution time the behavior of a dual-port memory. Energy efficiency is enforced by two facts: First, the single-port blocks have an energy access cost which is smaller than that of monolithic (either single or dual-port) memories; second, address mapping is application-driven, and cell access frequency data is thus used to determine the optimal sizes of the memory blocks.

Based on analytical expressions for performance and energy consumption that allow to explore the energy-performance tradeoff, we present experimental results showing that the new architecture guarantees energy savings as high as 69% with respect to a dual-port memory config-
uration (54% with respect to the baseline, single-ported architecture), with comparable improvement of the memory bandwidth.

The rest of the chapter is organized as follows. Section 2 provides some background material on memory energy modeling, multi-port memories, and application-driven memory partitioning. Section 3 describes how partitioned memories can be used to achieve an energy-efficient shared memory architecture. Section 4 illustrates the analytical models used to drive the energy-performance exploration engine, which is discussed in Section 5. Section 6 presents the optimization results for a set of standard parallel applications. Finally, some concluding remarks are provided in Section 7.

2. Background

Modeling Memory Energy

Unlike generic hardware modules, the energy consumption of memories is basically independent of the input activity. What matters, in fact, is whether we are reading or writing a value from or to the memory, regardless of the value. This property allows to model memory energy consumption in an very abstract way, by explicitly exposing two independent variables affecting it: the cost of an access and the total number of accesses. This translates into the following formula:

\[ e_{tot} = \sum_{i=1}^{c_{tot}} e_i \]  

(1.1)

where \(c_{tot}\) is the total number of memory accesses, and \(e_i\) is the cost of each access. For the sake of simplicity, we equally weigh all accesses (i.e., we do not distinguish the cost of a read from that of a write).

Equation 1.1 exposes the two quantities we can consider to reduce the energy consumption of a memory system and will be used throughout the paper as a reference. Techniques for reducing memory energy can be thus classified according to which variable is optimized [6].

Multi-Port Memories

A multi-port memory is simply a memory that allows multiple simultaneous accesses for reads and writes to any location in memory. Multi-port memories are typically employed as shared memories in multiprocessor designs, and are especially popular as dual-ended FIFO buffers for bus interfacing, or for video/graphics buffering.

Multiple simultaneous accesses are made possible by duplicating some of the resources required to access a cell: the address and data pins, the word-lines, and the bit-lines. Figure 1.1 shows the structure of a typical
dual-port SRAM cell, and in particular the extra word-line (with the corresponding transistors) and extra bit-line.

![Diagram of a Dual-Port SRAM Cell](image)

*Figure 1.1. Structure of a Dual-Port SRAM Cell.*

In some devices, additional overhead is also required to handle the synchronization of *multiple writes* to the same cell; this is managed through a sort of hardware semaphore which serializes the concurrent accesses.

The increase in bandwidth provided by multi-port memories comes at the price of increased area, wiring resources and power consumption. Because of this considerable overhead, multi-port memories are usually limited to a few ports (often 2, and seldom more than 4). One noticeable exception is represented by register files (although they are not strictly SRAMs), that are typically highly multi-ported (even 16 or more ports) to provide very high bandwidth in superscalar processors.

Multi-port memories can also be characterized by the *flexibility* of the ports. In some memory devices, some of the ports can be specialized, i.e., they allow only some type of access (read or write). This fact can be expressed by writing the number of ports $P = p_r + p_w + p_{rw}$, where the three terms denote the number of read, write, and read/write ports, respectively. In this work, without loss of generality, we will assume that $p_r = p_w = 0$, and $p_{rw} = P$, that is, all ports can be used for any type of access at any time.

When analyzing multi-port memories from the energy point of view, we must take into account the two following non-idealities, supported by data from several multi-port memory providers ([7],[8],[9]).

a) Energy consumption of multi-port memories *does not scale linearly with the number of ports*. For instance, the energy cost for accessing a dual-port memory is more than twice the energy required for accessing a single-port memory of the same size.

b) When a multi-port memory is used as a shared memory in a multiprocessor system, there are cases in which not all the ports are used simultaneously. It may in fact happen that the access pattern of the application does not allow to group a set of accesses
(from the processors) into a single, multi-port access. In these cases, we must consider the fact that energy consumption does not scale linearly with the number of ports that are accessed simultaneously. For instance, the energy cost for accessing a single port in a dual-port memory is larger than the one for accessing a single-port memory of the same size.

With reference to the model of Equation 1.1, the use of multi-port memories reduces $c_{tot}$, but it implies a sizable increase of the access cost $e_i$.

**Application-Driven Memory Partitioning**

Partitioning a memory block into multiple blocks, based on the memory access profile, was originally proposed by Benini et al. [10]. Their technique exploits the fact that, due to the high locality exhibited by embedded applications, the distribution of memory references is not uniform. As a consequence, some memory locations will be accessed more frequently than others. The partitioning is realized by splitting the address space (stored onto a single, monolithic memory block) into non-overlapping contiguous sub-spaces (stored onto several, smaller memory blocks).

Reduction of energy consumption is achieved because of two facts. First, each block is smaller than the monolithic one, and thus it has a smaller access cost ($e_i$). Second, and more relevant, only one of the blocks is active at a time. By properly partitioning the address space, it should be possible to access the smallest blocks most of the times, and access the largest ones only occasionally.

The original implementation of [10] employs a sophisticated recursive algorithm to determine the optimal partition with an arbitrary granularity. In this work, we will exploit their idea, yet without employing the same partitioning engine. As a matter of fact, in our case partitioning is driven by the access patterns of more than one processor.

Memory partitioning specifically targets the reduction of the access cost $e_i$, and it does not change $c_{tot}$, since it does not modify the access patterns.

3. **Partitioned Shared Memory Architecture**

The target MPSoC architecture considered in this work is depicted in Figure 1.2. Each processor core has a cache and a private memory (PM) containing private data and code, which is accessed through a local bus. Processors are also connected to another memory (SM), through a common global bus containing the data that are shared between the various
threads executing on the processors. We do not consider here other types of interconnections, such as point-to-point ones (i.e., crossbars).

In this work, starting from the assumption that the shared memory is implemented as a conventional on-chip, single-port memory, we aim at improving the performance of the accesses to the shared memory, yet in a more energy-efficient way than resorting to a multi-port memory.

The proposed shared memory architecture combines the bandwidth advantages of multi-port memories (and thus the reduction of $c_{\text{tot}}$) with the advantages, in terms of energy consumption and access time, of partitioned, single-port memories (and thus the reduction of $e_{i}$).

In our scheme, the memory address space is mapped over single-port banks that can be simultaneously accessed by the different processors, so as to mimic the behavior of a multi-port memory for a large fraction of the execution time. Each bank covers a subset of the address space, with no replication of memory words; therefore, the address subspaces are non-overlapping. The latter issue is essential to understand why the partitioned scheme can only approach the performance of the multi-port architecture. Since the memory blocks are single-ported and contain non-overlapping subsets of addresses, simultaneous accesses from the processor can be parallelized only if they fit into different memory blocks. Otherwise, the potentially parallel access must take place into two consecutive memory cycles.

Energy efficiency is enforced by two facts: First, the single-port blocks have an energy access cost which is by far smaller than that of monolithic (either single or dual-port) memories; second, address mapping is application-driven, and it accounts thus for the cell access frequency to determine the size of the memory blocks which is most suitable for memory minimization.

In the following, we will restrict our analysis to systems with two processors. Consequently, we will consider dual-port memories, and the par-
tioned architecture will also consists of two blocks at most. Although the concepts that will be discussed apply in principle to an arbitrary number of processors (with multi-port memories and multi-bank architectures), the quantitative analysis of energy and performance strictly refers to the case of two processors (with dual-port memory and two memory blocks).

Figure 1.3. Dual-Port (a) and Partitioned Single-Port (b) Architectures.

Figure 1.3 show a conceptual architecture of the dual-port and the partitioned single-port schemes. Label $A_i$ refers to addresses from processor $i$, while $D_i$ refer to data to/from processor $i$. In the dual-port scheme (Figure 1.3-(a)), the existence of two read/write ports allows to bind each processor to one port, realizing in fact a point-to-point interconnection.

In the partitioned architecture (Figure 1.3-(b)), addresses and data must be multiplexed (from processor to memory) or de-multiplexed (from memory to processor) properly, to connect the processor to the required memory block. This block diagram just shows the high-level flow of data and addresses; the actual implementation of the decoder is actually more complex, and will be discussed in the experimental section.

Related Work

The literature on energy optimization of embedded memories is quite rich (see [6] for a comprehensive survey); however, most techniques deal with the optimization of caches, scratch-pad memories, or off-chip memories, and multi-port memories are seldom addressed.

Most energy optimizations for multi-port memories are concerned with the issue of the mapping of data structures (typically, arrays) to multi-port memories, based on the access profiles of the applications.
From these profiles, these techniques evaluate simultaneous array accesses (e.g., whether two or more arrays are accessed in the same cycle), and build a so-called compatibility graph, which expresses the potential parallelization of accesses. The various approaches differ then in how this graph is used to decide the optimal allocation of array accesses to memory ports [3, 11–13].

One technique closer to the one proposed in this work has been discussed by Lewis and Brackenbury [14]. Their approach is based on the typical access patterns of DSP applications, and splits highly-multiported register files into multiple banks of predefined sizes.

4. Energy and Performance Characterization

In this section we will derive analytical expressions for the number of memory accesses and for the total energy consumption for the architectures of Figure 1.3, referred to the case of a system consisting of two processors (hereafter denoted with $P_1$ and $P_2$).

**Performance Characterization**

Let $c_1$ and $c_2$ be the number of memory accesses required by the execution of the application on processors $P_1$ and $P_2$, respectively. In the following, we will use the term memory cycle instead of memory access; we adopt this terminology in order to distinguish accesses to the shared memory that can occur in parallel. In fact, the total number of memory accesses by a processor is fixed (and determined by the memory access pattern of the application, which we do not modify); What actually changes is the time (in cycles) required to serve these accesses. Furthermore, we will denote sets with bold symbols, and their cardinalities with lowercase ones.

Our reference performance figure is the total number of memory cycles for the case where shared memory is implemented as a monolithic single-port memory. This value is $c_{spm} = c_1 + c_2$.

**Dual-Port Memory.** When the shared memory is implemented by a monolithic dual-port memory, the total number of memory accesses will be smaller than $c_{spm}$ because of the possibility of simultaneous accesses. Only a fraction of the accesses, however, will occur simultaneously.

As Figure 1.4 shows, this fraction can be represented in terms of set notation. We denote with $C_{par}$ the set of memory cycles that can access memory simultaneously; $C_{par}$ consists of the union of two subsets $C_{par} = C_{par,1} \cup C_{par,2}$, where $C_{par,1} \subseteq C_1$ and $C_{par,2} \subseteq C_2$. These two
subsets have same cardinality (i.e., \( c_{\text{par}, 1} \equiv c_{\text{par}, 2} \)) because each element of one set matches one of the other set to make a parallel access.

![Diagram of Execution Cycles](image)

*Figure 1.4. Classification of Execution Cycles.*

The number of cycles for the dual-port configuration is therefore:

\[
c_{\text{dpm}} = (c_1 - c_{\text{par}, 1}) + (c_2 - c_{\text{par}, 2}) + c_{\text{par}}/2
\]

(1.2)

where \( c_{\text{par}} = c_{\text{par}, 1} + c_{\text{par}, 2} \), denotes the total number of the parallel cycles. The division by two in the last term denotes the fact that parallel cycles are actually grouped in pairs, with each pair corresponding to a single memory access. Equation 1.2 simplifies to \( c_{\text{dpm}} = c_1 + c_2 - c_{\text{par}}/2 \), exposing the fact that the magnitude of \( c_{\text{par}} \) directly translates into a performance improvement.

**Partitioned Memory.** In the case of partitioned memory, the two memory banks now host two non-overlapping subsets of the address space. This implies that only a subset of the cycles in \( C_{\text{par}} \) can be parallelized; in particular, accesses that fall in the same subset of addresses now need to be serialized, since the two memory blocks are single-ported.

This further setting of the cycles is depicted in Figure 1.5, using the same set notation as above. We can notice that \( C_1 \) and \( C_2 \) are now both split into two subsets, where \( C_{i,j} \) denotes the cycles of processor \( i \) that fall into block \( j \).

![Diagram of Execution Cycles for the Partitioned Architecture](image)

*Figure 1.5. Classification of Execution Cycles for the Partitioned Architecture.*

This induces a partition onto \( C_{\text{par}} \), as follows. The shaded areas labeled \( A \) and \( D \) in Figure 1.5 denote parallel accesses that fall into different memory blocks: In region \( A \) (\( D \)), \( P_1 \) accesses Block 1 (Block
2), and \( P_2 \) accesses Block 2 (Block 1). Conversely, the regions labeled \( B \) and \( C \) denote accesses that fall in the same memory block (Block 2 for region \( b \), and Block 1 for region \( c \)). Cycles belonging to region \( B \) and \( C \) cause a performance penalty, because, although they can potentially occur in parallel, they must be serialized (and thus require two memory accesses). 

These subsets can be characterized by using a quantity \( \lambda \), that denotes the percentage of the cycles in \( C_{par} \) that fall in distinct memory blocks (and can thus be made parallel). \( \lambda \) will be used in the following as a compact metric to evaluate the cost of the partition. In fact, \( \lambda \) depends on where the partition has been made, that is, how many addresses fall in each block. Therefore, \( C_{par} \) consists of \( \lambda c_{par} \) cycles that can be parallelized, and \( (1 - \lambda)c_{par} \) that requires two separate accesses.

The number of cycles of the partitioned-memory architecture \( c_{spm,part} \) is therefore:

\[
c_{spm,part} = (c_1 - c_{par,1}) + (c_2 - c_{par,2}) + \lambda c_{par}/2 + (1 - \lambda)c_{par} \tag{1.3}
\]

The formula simplifies to \( c_{spm,part} = c_1 + c_2 - \lambda c_{par}/2 \), exposing the fact that \( c_{spm,part} \geq c_{dpm} \), since \( \lambda \leq 1 \). Analyzing the dependency of \( c_{spm,part} \) versus \( \lambda \), We notice that \( c_{spm,part} \) (and thus) the performance penalty of the partitioned scheme is minimized when \( \lambda \) is maximized, as expected. In particular, when \( \lambda = 1 \), all accesses in \( C_{par} \) are parallelized, and the partitioned scheme is equivalent to the dual-port memory, performance-wise. When \( \lambda = 0 \), all accesses by \( C_{par} \) overlap on the same memory block, and the partitioned scheme is equivalent to the single-port memory architecture.

**Energy Characterization**

To compute energy, we stick to the high-level model of Equation 1.1; energy is thus simply obtained by multiplying each access for its cost.

**Dual-Port Memory.** In this case we have to consider two types of access costs, depending on whether one or both ports are accessed. Total energy is obtained thus by properly weighing the terms of Equation 1.2: In formula:

\[
e_{dpm} = (c_1 - c_{par,1}) \cdot e_{dpm,1} + (c_2 - c_{par,2}) \cdot e_{dpm,1} + c_{par}/2 \cdot e_{dpm,2} \tag{1.4}
\]

The term \( e_{dpm,x} \) denotes the energy per access to the memory, in which the term \( x = \{1, 2\} \) in the subscript denotes the number of ports used in the access.
Partitioned Memory. In the case of the partitioned memory, total energy cannot be conveniently expressed by a closed formula, for two reasons. First, the energy per access depends on the size of the memory block that is accessed; the sizes of the blocks, however, are precisely the variables of the partitioning problem we are trying to solve. Second, we have two single-port memories, and each memory access from either processor will fall into one of the two memory blocks. This implies that the energy per access can only be approximated by a “average” cost (i.e., the number of accesses to Block 1 weighted by its energy cost, plus number of accesses to Block 2 weighted by its energy cost).

The accurate evaluation of energy for the partitioned architecture requires thus a simulation of the dynamic address trace of the two processors, and the application of Equation 1.1 on an access-by-access basis.

Nevertheless, we can derive an approximate expression of total energy that can be used for a rough comparison with Equation 1.4:

\[
e_{spin, part} = (c_1 - c_{par, 1}) \cdot e_{spin} + (c_2 - c_{par, 2}) \cdot e_{spin}' + (1 - \lambda) c_{par} \cdot e_{spin}'' + \lambda c_{par} / 2 \cdot (e_{spin_1} + e_{spin_2})
\] (1.5)

The first two term \((e_{spin}' \text{ and } e_{spin}'')\) are the above mentioned average access costs and represent the non-parallel memory accesses. \(e_{spin}''\) is the cost of accessing either Block 1 or Block 2 (depending on the subset of addresses), when accesses are potentially parallel but must be serialized. The last term represents the subset of potentially parallel accesses that will access Block 1 and Block 2 simultaneously \((e_{spin_1} + e_{spin_2})\).

Although approximate, Equation 1.5 allows to do some rough comparison with the dual-port scheme. First, all energy costs in Equation 1.5 are smaller than \(e_{dpm, 2}\), and, in most cases (when the sizes of the two blocks are of comparable size), also smaller than \(e_{dpm, 1}\). This implies that all four terms of Equation 1.5 are smaller than the corresponding ones in Equation 1.4, and energy is potentially smaller than the dual-port memory case, regardless of the value of \(\lambda\).

The actual dependency of \(e_{spin, part}\) on \(\lambda\) is not easily observable from Equation 1.5. A large value of \(\lambda\) increases the probability of accessing both blocks in the same cycle (this corresponds to the largest term \((e_{spin_1} + e_{spin_2})\)). Therefore, energy should be in principle reduced by choosing partitions which minimize \(\lambda\). In this case, in fact, only one of the two blocks (each one smaller than the monolithic memory) will be accessed in each cycle, thus using less energy; a small value of \(\lambda\), however, tends to increase the number of cycles, as already observed.
5. Exploration Framework

The models described in Section 4 show that there exists a trade-off between energy and performance in partitioning the shared memory. Although we are searching for energy-efficient memory architectures, we cannot ignore performance implications; therefore, in order to search for the best energy/performance tradeoff, we use energy/delay product (EDP) as a metric, and choose to minimize EDP during the space exploration.

Thanks to the simple models of Section 4, the optimization space is relatively small, since \( \lambda \) is the only parameter of the models. \( \lambda \) is a function of the access pattern of the application, but it also depends on how the address space is partitioned. Partitions can be characterized by the boundary address \( B \) that splits the address space \([0, \ldots, N-1]\) into two sub-spaces \([0, \ldots, B-1]\) and \([B, \ldots, N-1]\). Therefore, \( \lambda \) is also a function of \( B \). As an example, Figure 1.6 shows the behavior of \( \lambda \) versus \( B \) for a parallel FFT kernel; we can observe that the curve is not monotonic, showing the sensitivity of \( \lambda \) to the access pattern.

![Figure 1.6. Behavior of \( \lambda(B) \) vs. \( B \).](image)

These observations lead us to the following exploration procedure, for a shared memory of \( N \) words:

1. Compute \( e_{pm}(\lambda) \) and \( c_{pm}(\lambda) \) as in Section 4;

2. For all possible values of \( B = 0, \ldots, N - 1 \), Compute \( EDP_{pm}(\lambda) \) as \( e_{pm} \cdot c_{pm} \). \( EDP_{pm}(\lambda(B)) \) is not a function, since there may be more values of \( B \) (and thus of \( EDP \)) for a given value of \( \lambda \). An example of such curve is shown in Figure 1.7, for the parallel FFT benchmark.

3. Compute the function \( EDP^{pareto}_{pm}(\lambda) \), obtained by selecting, for each value of \( \lambda \), the smallest value of \( EDP_{pm}(\lambda) \). \( EDP^{pareto}_{pm} \) contains the Pareto points of \( EDP_{pm}(\lambda) \), and can possibly contain
some discontinuities. Figure 1.8 shows the resulting curve for the FFT benchmark.

![Graph](image)

**Figure 1.7.** Behavior of $EDP(\lambda(B))$ vs. $\lambda$.

![Graph](image)

**Figure 1.8.** Pareto Points of $EDP(\lambda(B))$.

4 Compute the minimum $EDP_{min}$ of this function, and let $\lambda_{min}$ the corresponding value of $\lambda$;

5 On the $\lambda$ vs. $B$ plot, identify the corresponding value $B_{min}$ of $B$. In case of multiple values of $B$, choose the one that makes the partitions as equal (in size) as possible.

6. **Experimental Results**

   **Experimental Setup**

   We have implemented our partitioned memory scheme in ABSS [15]. ABSS is an execution-driven architectural simulator for multiprocessor systems developed at Stanford University, that extends the ideas implemented in the AUGMINT simulator. ABSS is based on the idea of *augmentation*, that is, the instrumentation of the assembly code with
various hooks that allow to make context switches to the simulator; augmentation translates the program into a functionally equivalent program that runs on the simulated version of the processor.

The memory architecture provided by ABSS includes both private and shared memory. All the memories are connected through a single shared bus. Yet, ABSS does not provide any specific predefined cache or shared bus model; rather, it defines a specific interface to which user-defined cache and bus models can be easily hooked.

We have integrated Dinero [16] into ABSS, in order to provide accurate cache simulation data, and we have derived performance and energy models for the shared memory (both single- and dual-port) by interpolation of the results obtained from an industrial memory generator by ST Microelectronics. The target technology for all the models is 0.18μ.

Concerning the benchmarks, we have used Stanford’s SPLASH suite [17] which includes a set of kernels and parallel applications widely used in the parallel computing community.

**Energy/Performance Tradeoff Analysis**

Table 1.1 shows energy-delay product (EDP) results for the above benchmarks, for the monolithic, single-port architecture ($EDP_{mm}$) and the partitioned one ($EDP_{pm}$), obtained using the exploration procedure of Section 5. The EDP reduction (Column Δ) ranges from 40.5% to 62.3% (50.2% on average).

<table>
<thead>
<tr>
<th>Application</th>
<th>$EDP_{mm}$</th>
<th>$EDP_{pm}$</th>
<th>Δ [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barnes</td>
<td>24987.8</td>
<td>11357.5</td>
<td>54.6</td>
</tr>
<tr>
<td>FFT</td>
<td>6.4</td>
<td>3.7</td>
<td>41.2</td>
</tr>
<tr>
<td>FMM</td>
<td>853.4</td>
<td>389.6</td>
<td>54.4</td>
</tr>
<tr>
<td>LU</td>
<td>3931.3</td>
<td>2339.2</td>
<td>40.5</td>
</tr>
<tr>
<td>LU-CONT</td>
<td>3734.4</td>
<td>2073.1</td>
<td>44.3</td>
</tr>
<tr>
<td>Radix</td>
<td>59512.5</td>
<td>23180.5</td>
<td>61.0</td>
</tr>
<tr>
<td>Volrend</td>
<td>869794.2</td>
<td>453283.8</td>
<td>47.9</td>
</tr>
<tr>
<td>Water-N2</td>
<td>150460.7</td>
<td>56710.0</td>
<td>62.3</td>
</tr>
<tr>
<td>Water-S</td>
<td>10581.2</td>
<td>5770.5</td>
<td>45.5</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td><strong>50.2</strong></td>
</tr>
</tbody>
</table>

The exploration procedure also allows to compute the best performance and energy points; these are summarized in Table 1.2, where performance improvements (number of cycles) and energy saving with respect to the monolithic, single port architecture are reported (Columns $Best\ Performance$ and $Best\ Energy$).
Table 1.2. Optimal Performance and Energy Points.

<table>
<thead>
<tr>
<th>Application</th>
<th>Best Performance [%]</th>
<th>Best Energy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barnes</td>
<td>1.5</td>
<td>54.4</td>
</tr>
<tr>
<td>FFT</td>
<td>34.0</td>
<td>37.2</td>
</tr>
<tr>
<td>FMM</td>
<td>2.1</td>
<td>54.4</td>
</tr>
<tr>
<td>LU</td>
<td>10.9</td>
<td>40.3</td>
</tr>
<tr>
<td>LU-CONT</td>
<td>19.8</td>
<td>40.5</td>
</tr>
<tr>
<td>Radix</td>
<td>25.4</td>
<td>60.9</td>
</tr>
<tr>
<td>Voilrend</td>
<td>0.3</td>
<td>50.3</td>
</tr>
<tr>
<td>Water-M2</td>
<td>13.9</td>
<td>62.3</td>
</tr>
<tr>
<td>Water-S</td>
<td>8.7</td>
<td>45.9</td>
</tr>
<tr>
<td>Average</td>
<td>13.0</td>
<td>49.6</td>
</tr>
</tbody>
</table>

The comparison of Tables 1.1 and 1.2, shows that savings in the EDP is mostly due to energy savings than to performance savings. Minimum EDP points are in fact very close to minimum energy points, for most of the benchmarks, while performance improvements are less significant. Notice also that only benchmarks that exhibit a sizable amount of parallel cycles (e.g., FFT, LU-CONT, Radix) results in a sizable performance improvement. Conversely, energy does not seem to be that sensitive to the amount of parallel cycles.

Figure 1.9 shows the energy savings of the the partitioned architecture with respect to the dual-port case. Numbers refer to best-performance points, since we want to reduce the performance penalty as much as possible. The savings do not include the cost of the decoding logic. The partitioned architecture results in an average energy saving of 56% (maximum 70%). This energy saving is achieved at an increase of the total number of memory cycles of 2.4% on average (10.1% maximum).

**Decoder Implementation**

The partitioned architecture requires an ad-hoc encoder which implements the conceptual scheme of Figure 1.3. The encoder must provide two main functionalities. First, it must drive the selectors that decide to which block a given memory access is directed; to do this, it must contain the information about the boundary of the partition of the address space. Second, and more important, it must handle the connection between processors and memory blocks; this requires a sort of arbitration mechanism that allows to serialize accesses that are potentially parallel, but fall in the same subset of addresses (i.e., memory block).
Figure 1.9. Energy Savings of the Partitioned Architecture vs. the Multi-Port One.

Figure 1.10 shows a more detailed block diagram of the encoder. It takes as inputs the addresses $A_1$ and $A_2$ from the two processors, the corresponding request signals $Req_i$, and the value $B$ of the address corresponding to the partition. It then generates the addresses to be sent to each memory block $A_{B_1}$ and $A_{B_2}$, and the signals used to allow the processors to access memory $Grant_i$. The latter are both active but in the cases where potentially parallel accesses must be serialized.

Figure 1.10. Block Diagram of the Decoder.

The decoder contains two main blocks. The first block ($RH$, Request Handler) checks the two addresses $A_1$ and $A_2$, and generates the $Busy_i$
outputs as well as a signal that determines whether the accesses can be parallelized or not (S/N/S). The other block (SEL), uses three inputs to decide to what memory block to send what address: the S/N/S input, and the outputs A_1 and A_2 of two comparators (the boxes labeled with “=” which determine in which block A_1 and A_2 are falling, respectively. By using the value of B as an external input, it is possible to make the decoder application-independent, and therefore to have one single encoder for any application. We have implemented the decoder in VHDL, and synthesized it on a 0.18μm technology library by ST Microelectronics, using Synopsys Design Compiler. When applying the memory access trace of the FFT benchmark, the dissipation of the decoder is 0.35 μJ, about 1.7% of total memory energy consumption (19.8 μJ).

Concerning delay, although the decoder is on the critical path (its delay adds up to the memory access time), this is not really an issue in the partitioned architecture. In fact, the memory cycle time in this case is smaller than that of the dual-port case, since we are accessing smaller memory blocks. Quantitatively, the partitioned architecture results in a slack equal to \( d_{spm} - \max(d_{spm,1}, d_{spm,2}) \), where \( d_i \) denotes access time to the corresponding memory block. The delay of the decoder obtained from synthesis is 310ps, well within this slack.

7. Conclusions

We have proposed an energy-efficient alternative to multi-port memories suitable for the implementation of the shared memory of multi-processor systems-on-chip. The architecture is based on application-driven partitioning of the address space into multiple banks.

The target of the architecture is to achieve little or no performance penalty with respect to multi-port memories; therefore, we pursue maximum performance partitioning solutions, corresponding the case where the chance of parallelizing the accesses is maximized. The architecture can be enhanced so that zero performance penalty is achieved, thank to the use of an extra memory buffer.

Experiments on a set of parallel benchmarks has shown average energy-delay product (EDP) reductions of 50% on average, with respect to the baseline case of a single-port memory, and energy savings of 56%, with respect to the case of a multi-port memory, with an average 2% performance penalty.
References


