Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2010, Article ID 871510, 16 pages
doi:10.1155/2010/871510

Research Article

Algorithms for Optimally Arranging Multicore Memory Structures

Wei-Che Tseng, Jingtong Hu, Qingfeng Zhuge, Yi He, and Edwin H.-M. Sha

Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080, USA

Correspondence should be addressed to Wei-Che Tseng, wxt043000@utdallas.edu

Received 31 December 2009; Accepted 6 May 2010

Academic Editor: Chun Jason Xue

As more processing cores are added to embedded systems processors, the relationships between cores and memories have more influence on the energy consumption of the processor. In this paper, we conduct fundamental research to explore the effects of memory sharing on energy in a multicore processor. We study the Memory Arrangement (MA) Problem. We prove that the general case of MA is NP-complete. We present an optimal algorithm for solving linear MA and optimal and heuristic algorithms for solving rectangular MA. On average, we can produce arrangements that consume 49% less energy than an all shared memory arrangement and 14% less energy than an all private memory arrangement for randomly generated instances. For DSP benchmarks, we can produce arrangements that, on average, consume 20% less energy than an all shared memory arrangement and 27% less energy than an all private memory arrangement.

1. Introduction

When designing embedded systems, the application of the system may be known and fixed at the time of the design. This grants the designer a wealth of information and the complex task of utilizing the information to meet stringent requirements, including power consumption and timing constraints. To meet timing constraints, designers are forced to increase the number of cores, memory, or both. However, adding more cores and memory increases the energy consumption. As more processing cores are added to a processor, the relationships between cores and memories have more influence on the energy consumption of the processor.

In this paper, we conduct fundamental research to explore the effects of memory sharing on energy in a multi-core processor. We consider a multi-core system where each core may either have a private memory or share a memory with other cores. The Memory Arrangement Problem (MA) decides whether cores will have a private memory or share a memory with adjacent cores to minimize the energy consumption while meeting the timing constraint. Some examples of memory arrangements are shown in Figure 1.

The main contributions of this paper are as follows.

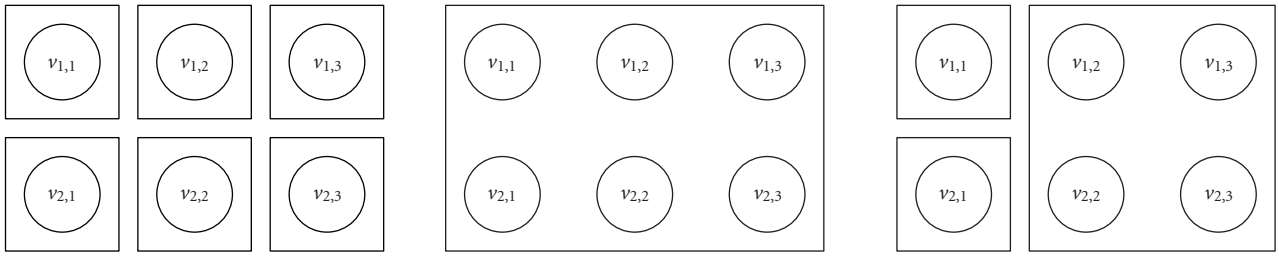
(i) We prove that MA without sharing constraints is NP-complete.

(ii) We propose an efficient optimal algorithm for solving linear cases of MA and extend it into an efficient heuristic for solving rectangular cases of MA.

(iii) We propose both an optimal algorithm and an efficient heuristic for solving rectangular cases of MA where only rectangular blocks of cores share memories.

Our experiments show that, on average, we can produce arrangements that consume 49% less energy than an all shared memory arrangement and 14% less energy than an all private memory arrangement for randomly generated instances. For benchmarks from DSPStone [1], we can produce arrangements that, on average, consume 20% less energy than an all shared memory arrangement and 27% less energy than an all private memory arrangement.

The rest of the paper is organized as follows. Related works are presented in Section 2. Section 3 provides a motivational example to demonstrate the importance of MA. Section 4 formally defines MA and presents two properties of MA. Section 5 presents an optimal algorithm



(a) All Private (b) All Shared (c) Mixed

Figure 1: Memory arrangements. Each circle represents a core, and each rectangle represents a memory.

for linear instances of MA. Section 6 proves that MA with arbitrary memory sharing is NP-complete. Section 7 presents algorithms to solve rectangular instances of MA including an optimal algorithm where only rectangular sets of cores can share a memory and an efficient heuristic to find a good memory arrangement in a reasonable amount of time. Section 8 presents our experiments and the results. We conclude our paper in Section 9.

2. Related Works

Many researchers in different areas have already begun lowering the energy consumption of memories. On a VLIW architecture, Zhao et al. [2] study the effect of register file repartitioning on energy consumption. Wang et al. [3] develop a leakage-aware modulo scheduling algorithm to achieve leakage energy savings for DSP applications with loops. For multiprocessor embedded systems, Qiu et al. [4] take advantage of Dynamic Voltage Scaling to optimally minimize the expected total energy consumption while satisfying a timing constraint with a guaranteed confidence probability. On a multi-core architecture, Hua et al. [5] use Adaptive Body Biasing as well as Dynamic Voltage Scaling to minimize both dynamic and leakage energy consumption for applications with loops. Saha et al. [6] attack the synchronization problems of concurrent memory accesses by proposing a new software transactional memory system that makes it both easy and efficient for multiprocess programs to share memory. Kumar et al. [7] focus on the interconnects of a multi-core processor. They show that interconnects play a bigger role in a multi-core processor than in a single core processor. We attack the problem from a different angle, exploring how memory sharing in a multi-core processor can affect the energy consumption.

Other researchers have worked on problems more specific to the memory subsystem of multi-core systems including data partitioning and task scheduling. In a timing focused work, Xue et al. [8] present a loop scheduling with memory management technique to completely hide memory latencies for applications with multidimensional loops. Suhendra et al. [9] present an ILP formulation that performs data partitioning and task scheduling simultaneously. Zhang et al. [10] present two heuristics to solve larger problems efficiently. The memory architectural model used is a virtually shared scratch pad memory (VS-SPM) [11], where each core has its own private memory and treats all the memories of the other cores as one big shared memory. Other researchers also start with a given multi-core memory architecture and use the memory architecture to partition data [12–16]. We approach the problem by designing the memory architecture around the application.

A few others have taken a similar approach. Meftali et al. [17] provide a general model for distributing data between private memories and a global shared memory. They assume that each processor has a local memory, and all processors share a remote memory. This is similar to an architecture with private L1 memories and a shared L2 memory. This architecture does not provide the possibility of only a few processors sharing a memory. The integer linear programming-(ILP-) based algorithm presented decides on the size of the private memories. Ozturk et al. [18] also combine both memory hierarchy design and data partitioning with an ILP approach to minimize the energy spent on data access. The weaknesses of this approach are that ILP takes an unreasonable amount of time for large instances, and timing is not considered. The generated architecture might be energy efficient but takes a long time to complete the tasks. In another publication, Ozturk et al. [19] aim to lower power consumption by providing a method for partitioning the available memory to the processing units or groups of processing units based on the number of accesses on each data element. The proposed method does not consider any issues related to time such as the time it takes to access the data or the duration of the tasks on each processing unit. Our proposed algorithms will consider these time constraints to ensure that the task lengths do not grow out of hand.

3. Motivational Example

In this section, we present an example that illustrates the memory arrangement problem. We informally explain the problem while we present the example.

The cores in a multi-core processor can be arranged either as a line or as a rectangle. For our example, we have 6 cores arranged in a 2×3 rectangle as shown in Figure 2.

Each core has a number of operations that it must complete. We can divide these operations into those that require memory accesses and those that do not. The computational time and energy required by operations that do not require memory accesses are independent of the memory

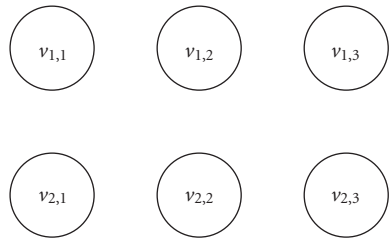


Figure 2: Motivational example. Each circle denotes a core.

Table 1: Data accesses.

	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$
$v_{1,1}$	5	0	0	3	0	0
$v_{1,2}$	0	0	0	0	0	5
$v_{1,3}$	0	0	2	0	0	0
$v_{2,1}$	4	0	0	0	0	0
$v_{2,2}$	0	0	0	0	2	0
$v_{2,3}$	0	5	0	0	0	0

arrangement. We do not consider the energy required by these operations since they are all constants, but we do consider the time required since it may affect the ability of a core to meet its timing constraint. Each core then has a constant time for the operations that do not require memory accesses. For our example, each core requires ten units of time for these operations.

For the operations that do require memory accesses, we count the number of these operations for each pair of cores. This number is the number of times a core needs to access the memory of another core. These counts for our example are shown in Table 1. In Table 1, the left column shows which core requires the memory accesses. The top row shows which core the memory accessed belongs to. For instance, $v_{1,1}$ has five operations that access its own memory and three operations that access the memory of $v_{2,1}$.

The computational time and energy required by each of these memory-access operations dependent on the memory arrangement. The least amount of time and energy required is when a core with private memory accesses its own memory. For our example, each of these accesses takes one unit of time and one unit of energy. The most amount of time and energy required is when a core accesses a remote memory. For our example, each of these accesses takes three units of time and three units of energy. In between, the amount of time and energy required when a core accesses a memory that it shares with another core is two units of time and two units of energy.

To make sure that the computations do not take too long, we restrict the time that each core is allowed to take. If, for a memory arrangement, any core takes more time than the timing constraint allows, we say that the memory arrangement does not meet the timing constraint. Sometimes it is impossible to find a memory arrangement that meets the timing constraint. For our example, the timing constraint is 25 units of time.

Two simple memory arrangements are the all private memory arrangement and the all shared memory arrangement. These are shown in Figure 1. Figure 1(a) shows the all private memory arrangement where each core has its own memory. Figure 1(b) shows the all shared memory arrangement where all cores share one memory.

Let us calculate the time and energy used by these two memory arrangements. First, let us consider the cores $v_{1,1}$ and $v_{2,1}$. In the all private memory arrangement, $v_{1,1}$ uses 5 units of time and energy to access its own memory and 9 units of time and energy to access the memory of $v_{2,1}$. Including the operations that do not need memory accesses, $v_{1,1}$ uses a total of 24 units of time and 14 units of energy. $v_{2,1}$ uses 12 units of time and energy to access the memory of $v_{1,1}$. Including the non-memory-access operations, $v_{2,1}$ uses a total of 22 units of time and 12 units of energy. Together, these two cores use 26 units of energy.

In the all shared memory arrangement, $v_{2,1}$ uses 8 units of time and energy to access the memory of $v_{1,1}$. Including the non-memory-access operations, $v_{2,1}$ uses a total of 18 units of time and 8 units of energy. $v_{1,1}$ uses 10 units of time and energy to access its own memory and 6 units of time and energy to access the memory of $v_{2,1}$. Including the non-memory-access operations, $v_{1,1}$ uses a total of 26 units of time and 16 units of energy. Together, these two cores use 24 units of energy, which is less than the 26 units of energy that the all private memory arrangement uses. However, $v_{1,1}$ takes 26 units of time, thus the all shared memory arrangement does not meet the timing constraint. We should use the all private memory arrangement even though it uses more energy.

Let us now consider the cores $v_{1,2}$, $v_{1,3}$, $v_{2,2}$, and $v_{2,3}$. In the all private memory arrangement, cores $v_{1,2}$ and $v_{2,3}$ each use 15 units of time and energy to access each other's memory. Including the non-memory-access operations, $v_{1,2}$ and $v_{2,3}$ each use 25 units of time and 15 units of energy. $v_{1,3}$ and $v_{2,2}$ each use 2 units of time and energy to access its own memory. Including the non-memory-access operations, $v_{1,3}$ and $v_{2,2}$ each use 12 units of time and 2 units of energy. Together, these four cores use 34 units of energy.

In the all shared memory arrangement, cores $v_{1,2}$ and $v_{2,3}$ each use 10 units of time and energy to access each other's memory. Including the non-memory-access operations, $v_{1,2}$ and $v_{2,3}$ each use 20 units of time and 10 units of energy. $v_{1,3}$ and $v_{2,2}$ each use 4 units of time and energy to access its own memory. Including the non-memory-access operations, $v_{1,3}$ and $v_{2,2}$ each use 14 units of time and 4 units of energy. Together, these four cores use 28 units of energy, which is less than the 34 units of energy that the all private memory arrangement uses, but the all shared memory arrangement does not meet the timing constraint for $v_{1,1}$. Hence, the best we can do with either an all shared or all private memory arrangement is to use 60 units of energy.

Instead of an all private or all shared memory arrangement, it would be better to have a mixed memory arrangement where $v_{1,1}$ and $v_{2,1}$ each use a private memory while the rest of the cores share one memory as shown in Figure 1(c). This memory arrangement uses only 54 units of energy and meets the timing constraint. All of our algorithms are able to achieve this arrangement, but it is possible to do better.

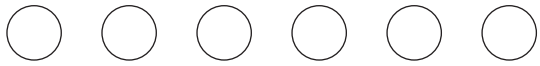


Figure 3: Linear array of cores. Each circle denotes a core.

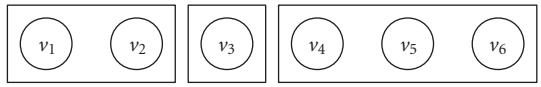


Figure 4: Memory sharing example. Each circle represents a single core. All cores in the same rectangle share a memory.

If we have an arrangement such that $v_{1,2}$ and $v_{2,3}$ share a memory but all the other cores have private memories, then we can meet the timing constraint and use only 50 units of energy. This arrangement, however, is difficult to implement since $v_{1,2}$ and $v_{2,3}$ are not adjacent to each other. In a larger chip, it is not advantageous from an implementation point of view to have two cores on opposite sides of the chip share a memory. Moreover, we prove that this version of the problem is NP-complete in Section 6.

4. Problem Definition

We now formally define our problem. Let us consider the problem of memory sharing to minimize energy while meeting a timing constraint assuming that all operations and data have already been assigned to cores. We call this problem the Memory Arrangement Problem (MA). We first explain the memory architecture then MA.

We are given a sequence $V = \langle v_1, v_2, v_3, \ldots, v_n \rangle$ of processor cores. The cores are arranged either in a line or a rectangle. For example, the cores in Section 5 are arranged in a line. An example is shown in Figure 3. Each core has operations and data assigned to it. We can divide the operations into memory-access-operations and non-memory-access operations. For a core $u \in V$, $b(u)$ is the time it takes for u to complete all its non-memory access operations. For cores $u, v \in V$, $w(u, v)$ is the number of times core u accesses a data that belongs to v. The time and energy it takes for u to access a data that belongs to v depends on how the memories of u and v are related. If u and v share the same private memory, that is, $u = v$, and u does not share a memory with any other cores, then the time and energy each memory-access operation takes are t_0 and e_0, respectively. If u and v share a memory, but $u \neq v$, then the time and energy each memory-access operation takes are t_1 and e_1, respectively. If u and v do not share a memory, then the time and energy each memory-access operation takes are t_2 and e_2, respectively. For convenience, let us denote the time and energy each memory-access operation takes as $C_t(u, v)$ and $C_e(u, v)$, respectively. For example, if v_3 and v_5 share the same memory, then $C_t(v_3, v_5) = t_1$ and $C_e(v_3, v_5) = e_1$.

We can represent the memory sharing of the cores with a partition of the cores such that two cores are in the same block if they share a memory. Let us consider the example in Figure 4. The memory sharing can be captured by the partition $\{\{v_1, v_2, v_3\}, \{v_4\}, \{v_5, v_6\}\}$.

We wish to find a partition of the cores to minimize the total energy used by memory-access operations:

$$\sum_{u \in V} \sum_{v \in V} C_e(u, v) w(u, v). \tag{1}$$

Energy is not our only concern. We also want to make sure that all operations finish within the timing constraint. Aside from memory-access operations, non-memory-access operations also take time. Since the memory sharing does not effect the time taken by non-memory access operations, for each $u \in V$ we describe all the time taken by non-memory-access operations by a single number $b(u)$. To meet a timing constraint q,

$$b(u) + \sum_{v \in V} C_t(u, v) w(u, v) \leq q \quad \forall u \in V. \tag{2}$$

MA then asks, given a sequence V, $w(u, v) \in \mathbb{Z}^*$ for each $u, v \in V$, $b(u) \in \mathbb{Z}^*$ for each $u \in V$, and nonnegative integers $t_0, e_0, t_1, e_1, t_2, e_2, q$, "what is a partition P such that the total energy used by memory-access operations is minimized and the timing constraint is met?"

Now that we have formally defined MA, we look at two of its properties. We use these properties in the later sections.

4.1. Optimal Substructure Property. Suppose that P is an optimal partition of V for an instance $I = \langle V, w, b, t_0, e_0, t_1, e_1, t_2, e_2, q \rangle$. Let B_1 be the block that contains v_1. Suppose that P' is an optimal partition for the subinstance $I' = \langle V', w, b', t_0, e_0, t_1, e_1, t_2, e_2, q \rangle$, where V' and b' are defined as follows:

$$\begin{aligned} V' &= V - B_1 \\ b'(u) &= b(u) + t_2 \sum_{v \in B_1} w(u, v) \quad \forall u \in V'. \end{aligned} \tag{3}$$

Lemma 1. *$P' = P - \{B_1\}$ is an optimal partition for I'.*

Proof. Let us prove Lemma 1 by contradiction. Suppose for the purpose of contradiction that P' is not an optimal partition for I'. Then there is a partition Q' for I' such that Q' is a better partition than P'. Since Q' is a partition that meets the timing requirements in I', $Q = Q' \cup \{B_1\}$ is also a partition that meets the timing requirements in I. Furthermore, Q is a better partition than P, a contradiction. □

4.2. Conglomerate Property. Suppose a partition P contains two different blocks of size at least 2, that is, $B_i, B_j \in P$, where $i \neq j$, $|B_i| > 1$, and $|B_j| > 1$. Let $P' = P - \{B_i, B_j\} \cup \{B_i \cup B_j\}$. If $t_1 \leq t_2$ and $e_1 \leq e_2$, then P' would be a partition that is as good as or better than P.

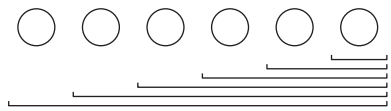


Figure 5: Subinstances. There are 6 sets cores. Each set has one more core than the previous set.

Proof. Let $V' = V - B_1 - B_2$ and $B' = B_1 \cup B_2$. The total energy used by the cores in B_1 and B_2 is

$$\begin{aligned}
&\sum_{u\in B_1}\sum_{v\in B_1} e_1 w(u,v) + \sum_{u\in B_1}\sum_{v\in V-B_1} e_2 w(u,v) \\
&\quad + \sum_{u\in B_2}\sum_{v\in B_2} e_1 w(u,v) + \sum_{u\in B_2}\sum_{v\in V-B_2} e_2 w(u,v) \\
&= \sum_{u\in B_1}\sum_{v\in B_1} e_1 w(u,v) + \sum_{u\in B_1}\sum_{v\in B_2} e_2 w(u,v) \\
&\quad + \sum_{u\in B_1}\sum_{v\in V-B'} e_2 w(u,v) + \sum_{u\in B_2}\sum_{v\in B_1} e_2 w(u,v) \\
&\quad + \sum_{u\in B_2}\sum_{v\in B_2} e_1 w(u,v) + \sum_{u\in B_2}\sum_{v\in V-B'} e_2 w(u,v) \qquad (4) \\
&\geq \sum_{u\in B_1}\sum_{v\in B_1} \mathrm{e}_1 w(u,v) + \sum_{u\in B_1}\sum_{v\in B_2} e_1 w(u,v) \\
&\quad + \sum_{u\in B_1}\sum_{v\in V-B'} e_2 w(u,v) + \sum_{u\in B_2}\sum_{v\in B_1} e_1 w(u,v) \\
&\quad + \sum_{u\in B_2}\sum_{v\in B_2} e_1 w(u,v) + \sum_{u\in B_2}\sum_{v\in V-B'} e_2 w(u,v) \\
&= \sum_{u\in B'}\sum_{v\in B'} e_1 w(u,v) + \sum_{u\in B'}\sum_{v\in V-B'} e_2 w(u,v).
\end{aligned}$$

□

5. Linear Instances

In this section, we consider the linear instances of MA. Linear instances are where the cores are arranged in a line. An example is shown in Figure 3. Let us make the assumption that only cores next to each other can share a memory. In other words, shared memories must only contain continuous blocks of cores, that is, if $u_i, u_j \in V$ are in the same block $B_x \in P$, then $u_k \in B_x$ for all $i \leq k \leq j$. This is the case in real applications since it is difficult to share memory between cores that are not adjacent. We consider what happens when we allow arbitrary cores to share a memory in Section 6.

Using the optimal substructure property of MA, we can solve the problem recursively. Unfortunately, in Section 4.1 we assumed that we already know the first block of an optimal partition. Since we do not know any optimal partitions, we will try all the possible first blocks and then find the best block. Figure 5 shows an example of the sub-instances of a problem. Notice that because of our assumption, all the sub-instances include v_n.

Let the largest sub-instance that contains the core v_i be $I_i = \langle V_i, w, b_i, t_0, e_0, t_1, e_1, t_2, e_2, q\rangle$, where V_i and b_i are defined as follows:

$$\begin{aligned}
V_i &= \{v_i, v_{i+1}, v_{i+2}, \ldots, v_n\}, \\
b_i(u) &= b(u) + t_2 \sum_{v\in V-V_i} w(u,v) \quad \forall u \in V_i.
\end{aligned} \qquad (5)$$

Note that $I_1 = I$, and there are, including I_1, only n sub-instances.

For each sub-instance I_i, let P_i be an optimal partition that satisfies the timing constraints. Let d_i be the energy consumption of P_i or ∞ if no partition can meet the timing constraint for I_i. Let V_i^ℓ be the first ℓ cores in V_i, that is, $V_i^\ell = \{v_i, v_{i+1}, v_{i+2}, \ldots, v_{i+\ell-1}\}$. Let c_i^ℓ be the minimum energy necessary for I_i if V_i^ℓ is a block in P_i. Let d_i^ℓ be ∞ if no partition of V_i that contains V_i^ℓ as a block satisfies the timing constraints. Otherwise, let d_i^ℓ be c_i^ℓ. We can define c_i^ℓ, d_i^ℓ, and d_i recursively as (6), (7), and (8), respectively.

During the computation of d_i, we record the optimal value of ℓ by recording the corresponding partition in P_i. Let $P_{n+1} = \{\}$. For all $1 \leq i \leq n$, let k be an optimal value of ℓ used to compute d_i. Then $P_i = \{V_i^k\} \cup P_{i+k}$. If $d_i = \infty$, then there is no partition for I_i that satisfies the timing requirement, and P_i is undefined. P_1 is an optimal partition for I, and d_1 is the energy necessary. If $d_1 = \infty$, then there does not exist a partition for I that satisfies the timing requirement.

Optimal Linear Memory Arrangement (OLMA), shown in Algorithm 1, is an algorithm to compute P_i and d_i. It starts by setting the sentinels for P_{n+1} and d_{n+1} in lines 1-2. The body of the algorithm is the for loop on lines 3–15. Notice that it computes P and d from n to 1. For each value of i, OLMA computes d_i^ℓ starting from $\ell = 1$. c_i^ℓ and d_i^ℓ are computed according to equations (6) and (7) on line 9. Lines 10–13 record the optimal P_i whenever a better d_i^ℓ is found. At the end of the algorithm, P_1 holds an optimal partition for

Input: An instance I of Linear MA.
Output: An optimal partition P_1 and its energy consumption d_1.

```
(1) d_{n+1} ← 0
(2) P_{n+1} ← {}
(3) for i ← n to 1 do
(4)       V_i ← {v_i, v_{i+1}, v_{i+2}, ..., v_n}
(5)       d_i ← ∞
(6)       P_i ← {}
(7)       for ℓ ← 1 to n − i + 1 do
(8)           V_i^ℓ ← {v_i, v_{i+1}, v_{i+2}, ..., v_{i+ℓ−1}}
(9)           Compute c_i^ℓ and d_i^ℓ.
(10)          if d_i^ℓ < d_i then
(11)              d_i ← d_i^ℓ
(12)              P_i ← {V_i^ℓ} ∪ P_{i+ℓ}
(13)          end if
(14)      end for
(15) end for
```

Algorithm 1: Optimal linear memory arrangement (OLMA).

Figure 6: Example for OLMA. Each circle is a core.

Table 2: Data accesses.

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	5	0	0	0	0	3
v_2	0	0	0	5	0	0
v_3	0	0	2	0	0	0
v_4	0	5	0	0	0	0
v_5	0	0	0	0	2	0
v_6	4	0	0	0	0	0

I, and d_1 holds the energy consumption of P_1. The running time of OLMA is $O(n^4)$ where n is the number of cores.

Let us illustrate OLMA with an example. We unroll the example from Section 3 to create a linear example of 6 cores as shown in Figure 6. In other words, $V = \langle v_1, v_2, v_3, \ldots, v_6 \rangle$. The memory access operations are shown in Table 2. For each core $v \in V$, $b(u) = 10$. $t_0 = e_0 = 1$, $t_1 = e_1 = 2$, and $t_2 = e_2 = 3$. The timing constraint $q = 25$.

The computed values of d_i^ℓ are shown in Table 3, and the computed values of d_i and P_i are shown in Table 4. From these values, we see that if v_1 is not in a block by itself, then it is unable to meet the timing constraint. Thus, $d_1^\ell = \infty$ for $\ell > 1$. The optimal partition for this example is $P_1 = \{\{v_1\}, \{v_2, v_3, v_4\}, \{v_5\}, \{v_6\}\}$, and its energy consumption is $d_1 = 52$. Consider the following:

$$c_i^\ell = \begin{cases} d_{i+\ell} + \displaystyle\sum_{u \in V_i^\ell} \sum_{v \in V_i^\ell} e_0 w(u,v) + \sum_{u \in V_i^\ell} \sum_{v \in V - V_i^\ell} e_2 w(u,v) & \text{if } \left| V_i^\ell \right| = 1, \\ d_{i+\ell} + \displaystyle\sum_{u \in V_i^\ell} \sum_{v \in V_i^\ell} e_1 w(u,v) + \sum_{u \in V_i^\ell} \sum_{v \in V - V_i^\ell} e_2 w(u,v) & \text{if } \left| V_i^\ell \right| > 1, \end{cases} \tag{6}$$

$$d_i^\ell = \begin{cases} \infty \text{ if } \left| V_i^\ell \right| = 1 \text{ and } b_i(u) + t_0 w(u,v) + \displaystyle\sum_{v \in V_i - V_i^\ell} t_2 w(u,v) > q \text{ for any } u \in V_i^\ell, \\ \infty \text{ if } \left| V_i^\ell \right| > 1 \text{ and } b_i(u) + t_1 w(u,v) + \displaystyle\sum_{v \in V_i - V_i^\ell} t_2 w(u,v) > q \text{ for any } u \in V_i^\ell, \\ c_i^\ell \quad \text{otherwise}, \end{cases} \tag{7}$$

$$d_i = \begin{cases} 0 & \text{if } i = n+1, \\ \displaystyle\min_{1 \le \ell \le n-i+1} \left\{ d_i^\ell \right\} & \text{otherwise.} \end{cases} \tag{8}$$

6. NP-Completeness

Let us consider MA if we do not assume that only cores next to each other may share a memory. Since any cores can share a memory, the shape that the cores are arranged in does not affect the solution. We first define the decision version of MA and then show that it is NP-complete.

An instance of MA consists of a set V, functions $w : V \times V \to \mathbb{N}$ and $b : V \to \mathbb{N}$, nonnegative integers t_0, e_0, t_1, e_1, t_2, e_2, q, and k. The question is as follows. Is there a partition P of V such that the timing requirement q is met and the energy consumption is less than k?

Let us apply the conglomerate property. For any partition P, there is a partition P' such that P' is at least as good as P and P' contains only one block that has a cardinality greater than 1. We can specify P' with a subset $V' \subseteq V$ where V' contains the cores that do not share a memory with another core. Conversely, for any subset $V' \subseteq V$, there exists a corresponding partition $P = \{V - V'\} \cup \{\{v\} | v \in V'\}$. Thus, we can restate the decision question as follows. Is there a subset $V' \subseteq V$ such that its corresponding partition meets the timing and energy requirements?

Theorem 1. *MA is NP-complete.*

Proof. It is easy to see that MA $\in$ NP since a nondeterministic algorithm needs only to guess a partition of V and check in polynomial time whether that partition meets the timing and energy requirements.

We transform the well-known NP-complete problem KNAPSACK to MA. First, let us define KNAPSACK. An instance of KNAPSACK consists of a set U, a size $s(u) \in \mathbb{Z}^+$ and a value $v(u) \in \mathbb{Z}^+$ for each $u \in U$, and positive integers B and K. The question is as follows. Is there a subset $U' \subseteq U$ such that $\sum_{u \in U'} s(u) \le B$ and $\sum_{u \in U'} v(u) \le K$?

Let $U = u_1, u_2, u_3, \ldots, u_n$, $s(u)$, $v(u)$, B, and K be any instances of KNAPSACK. We must construct set V, a functions $w : V \times V \to \mathbb{N}$ and $b : V \to \mathbb{N}$, and nonnegative integers t_0, e_0, t_1, e_1, t_2, e_2, q, and k such that there is a subset $U' \subseteq U$ such that $\sum_{u \in U'} s(u) \le B$ and $\sum_{u \in U'} v(u) \le K$ if and

only if there is a subset $V' \subseteq V$ such that its corresponding partition meets both the timing and energy requirements.

We construct a special case of MA such that the resulting problem is the same as KNAPSACK. We start by setting $V = U \cup \{u_0\}$. Then, for all $v_1, v_2 \in V$,

$$w(v_1, v_2) = \begin{cases} s(v_2) & \text{if } v_1 = u_0 \text{ and } v_2 \in U, \\ s(v_2) + v(v_2) & \text{if } v_1 = v_2 \text{ and } v_1 \in U, \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

For all $v \in V$,

$$b(v) = \begin{cases} 0 & \text{if } v \in U, \\ \sum_{u \in U} v(u) & \text{if } v = u_0. \end{cases} \quad (10)$$

We complete the construction of our instance of MA by setting $t_0 = 0$, $e_0 = 1$, $t_1 = 1$, $e_1 = 2$, $t_2 = 2$, $e_2 = 3$, $q = \sum_{u \in U}[s(u) + v(u)] + B$, and $k = \sum_{u \in U}[4s(u) + 2v(u)] - K$.

It is easy to see how the construction can be accomplished in polynomial time. All that remains to be shown is that the answer to KNAPSACK is yes if and only if the answer to MA is yes.

Since $w(u_0, u_0) = 0$, it is of no advantage for u_0 to be in a block by itself. Therefore, $u_0 \notin V'$ unless $V \subseteq V'$. The time that u_0 needs to finish its tasks is

$$\begin{aligned} b(u_0) + \sum_{v \in V} C_t(u_0, v) w(u_0, v) & \\ = \sum_{u \in U} v(u) + \sum_{u \in U - S} s(u) + \sum_{u \in S - \{u_0\}} 2s(u) & \\ = \sum_{u \in U} [s(u) + v(u)] + \sum_{u \in S - \{u_0\}} s(u). & \end{aligned} \quad (11)$$

Notice that the time required by u_0 is greater than any $u \in U$. Hence, the timing constraint is met if and only if

$$\begin{aligned} \sum_{u \in U} [s(u) + v(u)] + \sum_{u \in S - \{u_0\}} s(u) & \\ \leq q = \sum_{u \in U} [s(u) + v(u)] + B. & \end{aligned} \quad (12)$$

Thus,

$$\sum_{u \in V' - \{u_0\}} s(u) \leq B. \quad (13)$$

Table 3: d_i^ℓ.

		ℓ					
		1	2	3	4	5	6
i	1	52	∞	∞	∞	∞	∞
	2	46	48	38	40	40	
	3	31	33	35	35		
	4	29	31	31			
	5	14	16				
	6	12					

Table 4: d_i and P_i.

i	d_i	P_i
1	52	$\{\{v_1\}, \{v_2, v_3, v_4\}, \{v_5\}, \{v_6\}\}$
2	38	$\{\{v_2, v_3, v_4\}, \{v_5\}, \{v_6\}\}$
3	31	$\{\{v_3\}, \{v_4\}, \{v_5\}, \{v_6\}\}$
4	29	$\{\{v_4\}, \{v_5\}, \{v_6\}\}$
5	14	$\{\{v_5\}, \{v_6\}\}$
6	12	$\{\{v_6\}\}$
7	0	$\{\}$

The total energy consumed is

$$\begin{aligned} & \sum_{u \in V} \sum_{v \in V} C_e(u, v) w(u, v) \\ & = \sum_{v \in U} C_e(u_0, v) w(u_0, v) + \sum_{u \in U} C_e(u, u) w(u, u) \\ & = \sum_{u \in U - V'} 2s(u) + \sum_{u \in V' - \{u_0\}} 3s(u) \\ & \quad + \sum_{u \in U - V'} 2[s(u) + v(u)] + \sum_{u \in V' - \{u_0\}} [s(u) + v(u)] \\ & = \sum_{u \in U} [4s(u) + 2v(u)] - \sum_{u \in V' - \{u_0\}} v(u). \end{aligned} \quad (14)$$

The energy consumption constraint is met if and only if

$$\begin{aligned} \sum_{u \in U} [4s(u) + 2v(u)] - \sum_{u \in V' - \{u_0\}} v(u) & \\ \leq \sum_{u \in U} [4s(u) + 2v(u)] - K. & \end{aligned} \quad (15)$$

Thus,

$$\sum_{u \in V' - \{u_0\}} v(u) \leq K. \quad (16)$$

Hence, there is a subset $V' \subseteq V$ that meets both the timing and energy requirements if and only if there is a subset $U' \subseteq U$ such that $\sum_{u \in U'} s(u) \leq B$ and $\sum_{u \in U'} v(u) \leq K$. Thus, MA is NP-complete. □

7. Rectangular Instances

Since general MA is NP-complete and linear MA is in P, let us consider the case when the cores are arranged as a rectangle.

An example of such an arrangement is our motivational example shown in Figure 2. We extend OLMA to solve the rectangular case in Section 7.1. In Section 7.2, we define what staircase-shaped sets are. Then we use staircase-shaped sets to optimally solve rectangular MA in Section 7.3. We finally present a good heuristic to solve rectangular MA in Section 7.4.

7.1. Zigzag Rectangular Partitions. We propose an algorithm Zigzag Rectangular Memory Arrangement (ZiRMA) to solve this problem. ZiRMA transforms rectangular instances into linear instances before applying OLMA. It runs in polynomial time but cannot guarantee optimality.

Let us use OLMA to handle this case by treating the rectangle as a zigzag line as shown in Figure 7(b). To transform an $m \times n$ rectangle into a line, we can simply relabel each core $v_{i,j}$ of an $m \times n$ rectangle as $v_{n \cdot i+j}$. An example of a resulting line is shown in Figure 7(a). Notice how $v_{1,5}$ and $v_{2,1}$ are not adjacent in the rectangle, but they are adjacent in the line. Instead, let us relabel the cores with a continuous zigzag line so that each core $v_{i,j}$ of an $m \times n$ rectangle becomes

$$v_{j(-1)^{i+1}+(n+1)[(i+1) \bmod 2]+n(i-1).} \tag{17}$$

The resulting line on the same rectangle is shown in Figure 7(b). Notice how adjacent cores in the line are also adjacent in the rectangle. Now we can use OLMA to solve the linear problem.

Unfortunately, not all cores adjacent in the rectangle are adjacent in the line. For example, $v_{1,2}$ and $v_{2,1}$ are adjacent in the rectangle, but they are separated by 6 other cores in the line. To mitigate this problem, we run OLMA twice—once on the horizontal zigzag line shown in Figure 7(b) and once on the vertical zigzag line shown in Figure 7(c). This time, let us relabel the cores in a vertical zigzag manner so that each core $v_{i,j}$ of an $m \times n$ rectangle becomes

$$v_{i(-1)^{j+1}+(m+1)[(j+1) \bmod 2]+m(j-1).} \tag{18}$$

After both iterations are complete, we have two partitions P_h and P_v of the same set of cores. We construct a new partition such that two cores share a memory if they share a memory in either P_h or P_v. To create the final partition, we merge a block from P_h and a block from P_v if they share a core. An example merge is shown in Figure 8.

ZiRMA is summarized in Algorithm 2. Its running time is $O(m^4n^4)$ for an $m \times n$ rectangle. We illustrate ZiRMA with our motivational example. We transform the cores according to Table 5. The accesses for the horizontal zigzag transformation are shown in Table 2, and the accesses for the vertical zigzag transformation are shown in Table 6. The resulting partitions are shown in Figure 9. In this case, the reverse transformations of P_h and P_v are the same, so merging does not have an effect.

As we can see from Figure 8, the shapes created by this algorithm may be long and winding, unsuitable for real implementations. Next, we make the restriction that the cores sharing a memory must be of a rectangular shape. To optimally solve this problem, we introduce the concept *staircase-shaped* set of cores.

Table 5: Core transformations.

Rectangular	Horizontal zigzag	Vertical zigzag
$v_{1,1}$	v_1	v_1
$v_{1,2}$	v_2	v_4
$v_{1,3}$	v_3	v_5
$v_{2,1}$	v_6	v_2
$v_{2,2}$	v_5	v_3
$v_{2,3}$	v_4	v_6

Table 6: Accesses for vertical transformation.

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	5	3	0	0	0	0
v_2	4	0	0	0	0	0
v_3	0	0	2	0	0	0
v_4	0	0	0	0	0	5
v_5	0	0	0	0	2	0
v_6	0	0	0	5	0	0

7.2. Staircase-Shaped Sets. Let us call a set of cores V_s *staircase shaped* if V_s satisfies the following requirements.

(1) All cores are right-aligned, that is, for each $1 \le i \le m$, there is an integer s_i such that $v_{i,j} \notin V_s$ for all $1 \le j \le s_i$ and $v_{i,j} \in V_s$ for all $s_i < j \le n$.

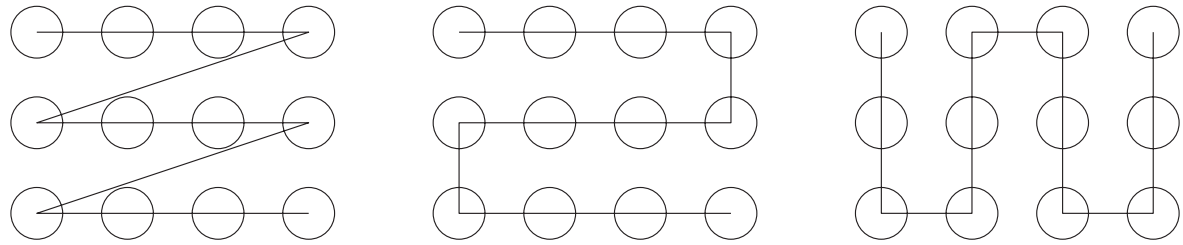
(2) Each row has at least as many cores in V_s as the previous row, that is, $s_1 \ge s_2 \ge s_3 \ge \cdots \ge s_m$.

Some examples of staircase-shaped sets are shown in Figure 10.

We can uniquely identify any staircase-shaped subset V_s of a rectangular set V by an m-tuple $s = (s[1], s[2], s[3], \ldots, s[m])$ such that $s[i]$ is the number of cores from row i of V that are not in V_s. For example, the tuples corresponding to the sets in Figures 10(a), 10(b), 10(c), and 10(d) are $(2, 1, 0)$, $(2, 2, 0)$, $(4, 2, 1)$, and $(4, 4, 2)$, respectively.

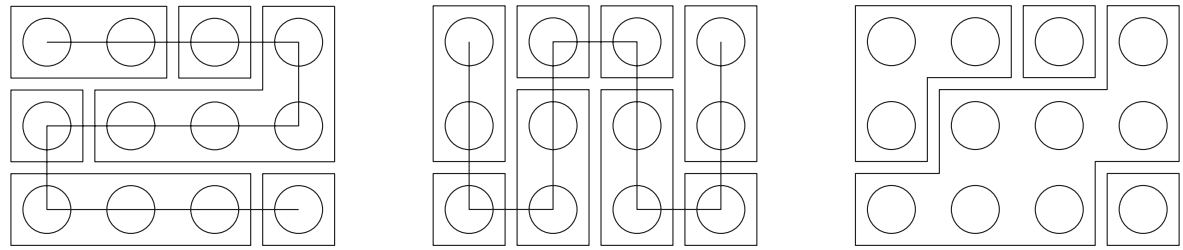
Let us consider all rectangular subsets $V_s^{i,j}$ of any staircase-shaped set V_s such that $V_s - V_s^{i,j}$ is a staircase-shaped set. Let $V_s^{i,j} = \{v_{i',j'} \mid i' \le i, j' \le j, \text{ and } v_{i',j'} \in V_s\}$. It is easy to see that $V_{s^{i,j}} = V_s - V_s^{i,j}$ is a staircase-shaped subset of V_s if V_s is a staircase-shaped set, $0 \le i \le m$, and $0 \le j \le n$. We see that $s^{i,j}$ is an m-tuple where $s^{i,j}[k] = \max(s[k], j)$ if $k \le i$ and $s^{i,j}[k] = s[k]$ if $k > i$.

Unfortunately, $V_s^{i,j}$ as defined does not necessarily have to be rectangular. To restrict $V_s^{i,j}$ to be rectangular, we define an m-tuple k_s such that for all $1 \le i \le m$, $k_s[i]$ is the largest integer such that $k_s[i] < i$ and $s[k_s[i]] \ne s[i]$. As a sentinel, let $s[0] = n+1$ so that $s[0] \ne s[i]$ for all $1 \le i \le m$. In words, row $k_s[i]$ is the closest row before row i that is different from row i. For example, the k_s's corresponding to Figures 10(a), 10(b), 10(c), and 10(d) are $(0, 1, 2)$, $(0, 0, 2)$, $(0, 1, 2)$, and $(0, 0, 2)$, respectively. Then, for all i, j such that $1 \le i \le m$, $j \le n$, and $s[i] < j \le \min(s[k_s[i]], n)$, $V_s^{i,j}$ is rectangular.



(a) Discontinuous (b) Horizontal (c) Vertical

Figure 7: Zigzag lines. We transform a rectangular problem into a linear problem by following one of these zigzag lines.



(a) P_h (b) P_v (c) P

Figure 8: Merging P_h and P_v. P is the partition resulting from merging P_h and P_v.

Input: An instance I of rectangular MA.
Output: A partition P and its energy consumption d.
(1) Create a linear instance I_h from I by transforming each core $v_{i,j}$ according to (17).
(2) Find the optimal partition P_h of I_h with OLMA.
(3) Reverse the transformation of each core in P_h by applying (17) in reverse.
(4) Create a linear instance I_v from I by transforming each core $v_{i,j}$ according to equation (18).
(5) Find the optimal partition P_v of I_v with OLMA.
(6) Reverse the transformation of each core in P_v by applying (18) in reverse.
(7) Create P by merging P_h and P_v.
(8) Compute the energy consumption d of P.

Algorithm 2: Zigzag rectangular memory arrangement (ZiRMA).

Lemma 2. *If a partition P of a nonempty staircase-shaped set V is composed of only rectangular blocks, there exists a block $B \in P$ such that $V - B$ is a staircase-shaped set.*

Proof. Let us suppose that V is m high and n wide. V then has at most m top left corners. For example, in Figure 10(a), the 3 top left corners are (3, 1), (2, 2), and (1, 3). Since all blocks of P are rectangular, none of the top left corners are in the same block. One of the blocks containing these corners is a block B' such that $V - B'$ is a staircase-shaped set. Let $B_1, B_2, B_3, \ldots, B_j$, where $j \leq m$, be the sequence of these blocks ordered by the row index of the top left corner that it contains. Let us consider all these blocks in this order.

If B_1 does not extend to the right underneath B_2, then it is a block such that the remaining blocks compose a staircase-shaped set, and the lemma is correct. If it does not, then it is not B', and one of the remaining blocks must be B'.

Let us consider B_i, where $i \leq j$. Since we are considering B_i, B_{i-1} must not be B', thus B_{i-1} extends underneath B_i, and B_i cannot extend down next to B_{i-1}. Thus, if B_i is not B', then it must extend to the right. If B_i does not extend to the right underneath B_{i+1}, then it is B', and the lemma is correct. Otherwise, it is not B', and we consider B_{i+1}. We continue this until we come to B_j.

By the same argument, B_j does not extend down next to B_{j-1}. Since this is the topmost top left corner, there is nothing above this block. Thus, B_j is B'. Thus, we have found a block such that the remaining blocks compose a staircase-shaped set. □

Lemma 3. *If a partition of a rectangular set is composed of only k rectangular blocks, there exists a sequence of the block $\langle B_1, B_2, B_3, \ldots, B_k \rangle$ such that for any integer $1 \leq i \leq k$, $\bigcup_{j=i}^{k} B_j$ is staircaseshaped.*

Proof. Since a rectangular set is staircase-shaped, we can repeatedly apply Lemma 2 to find such a sequence. □

7.3. Staircase Rectangular Partitions. We use staircase-shaped sets to find the optimal partition of a rectangular set of cores that only has rectangular blocks. For an MA instance

Table 7: d_s and P_s.

s	Shape	d_s	P_s
$(4,3,3)$		0	$\{\}$
$(4,3,2)$		15	$\{\{v_{2,3}\}\}$
$(4,3,1)$		17	$\{\{v_{2,2}\},\{v_{2,3}\}\}$
$(4,3,0)$		29	$\{\{v_{2,1}\},\{v_{2,2}\},\{v_{2,3}\}\}$
$(4,2,2)$		17	$\{\{v_{1,3}\},\{v_{2,3}\}\}$
$(4,2,1)$		19	$\{\{v_{1,3}\},\{v_{2,2}\},\{v_{2,3}\}\}$
$(4,2,0)$		31	$\{\{v_{1,3}\},\{v_{2,1}\},\{v_{2,2}\},\{v_{2,3}\}\}$
$(4,1,1)$		28	$\{\{v_{1,2},v_{1,3},v_{2,2},v_{2,3}\}\}$
$(4,1,0)$		40	$\{\{v_{2,1}\},\{v_{1,2},v_{1,3},v_{2,2},v_{2,3}\}\}$
$(4,0,0)$		54	$\{\{v_{1,1}\},\{v_{2,1}\},\{v_{1,2},v_{1,3},v_{2,2},v_{2,3}\}\}$

$I = \langle V, w, t_0, e_0, t_1, e_1, t_2, e_2, b, q\rangle$, let I_s be the sub-instance that contains a staircase-shaped set $V_s \subseteq V$, where s is an $m+1$-tuple such that $s[0] = n+1$ and for all $1 \le i \le m$, $0 \le s[i] \le n$ and $s[1] \ge s[2] \ge s[3] \ge \cdots \ge s[m]$. $I_s = \langle V_s, w, t_0, e_0, t_1, e_1, t_2, e_2, b_s, q\rangle$, where V_s and b_s are defined as follows:

$$V_s = \left\{v_{i,j} \mid 1 \le i \le m \text{ and } s[i] < j \le n\right\},$$
$$b_s(u) = b(u) + t_2 \sum_{v \in V - V_s} w(u,v) \quad \forall u \in V_s. \tag{19}$$

Let s^0 be the $m+1$-tuple that consists of all 0's except $s^0[0] = n+1$, and s^n be the $m+1$-tuple that consists of all n's except $s^n[0] = n+1$, i.e. $s^0 = (n+1, 0, 0, 0, \ldots, 0)$ and $s^n = (n+1, n, n, n, \ldots, n)$. Note that $I_{s^0} = I$. For each sub-instance I_s, let P_s be an optimal partition that satisfies the timing constraint. Let d_s be the energy consumption of P_s or ∞ if no partition for I_s can meet the timing constraint. Let $V_s^{i,j} = \{v_{i',j'} \mid i' \le i, j' \le j, \text{and } v_{i',j'} \in V_s\}$. Let $c_s^{i,j}$ be the minimum energy necessary for V_s if $V_s^{i,j}$ is a block in P_s. Let $d_s^{i,j}$ be ∞ if no partition that has $V_s^{i,j}$ as a block satisfies the timing constraints. Otherwise, let $d_s^{i,j}$ be $c_s^{i,j}$. And d_s, $c_s^{i,j}$, $d_s^{i,j}$, and P_s can be defined recursively as shown in equations (20), (21), (22), and (23), respectively.

P_{s^0} is an optimal partition, and d_{s^0} is the minimum energy necessary to meet the timing constraint. If $d_{s^0} = \infty$, then there is no partition for I that consists of only rectangular blocks that will satisfy the timing constraint.

An algorithm to compute P_s and d_s, Staircase Rectangular Memory Arrangement (StaRMA), is shown in Algorithm 3. We illustrate the algorithm on the motivational example. d_s and P_s for all s's that correspond to staircase-shaped sets are shown in Table 7. The second column of Table 7 shows the shape of the corresponding staircase-shaped set. To illustrate equation (20), $d_{(4,1,1)} = \min\{15 + d_{(4,2,1)}, 19 + d_{(4,3,1)}, 19 + d_{(4,2,2)}, 28 + d_{(4,3,3)}\} = 28$. The output partition is $P_{(4,0,0)} = \{\{v_{1,1}\},\{v_{2,1}\},\{v_{1,2},v_{1,3},v_{2,2},v_{2,3}\}\}$. Its energy consumption is $d_{(4,0,0)} = 54$.

By Lemma 3, if we search through all possible staircase-shaped sets, we search through all the partitions composed of only rectangular blocks. Since StaRMA loops through all the staircase-shaped subsets, it is able to find an optimal partition composed of only rectangular blocks.

$$d_s = \begin{cases} 0 & \text{if } s = s^n, \\ \min\limits_{1 \le i \le m} \left\{ \min\limits_{s[i] < j \le \min(s[k_s[i]], n)} \left\{ d_s^{i,j} \right\} \right\} & \text{otherwise,} \end{cases} \tag{20}$$

$$c_s^{i,j} = \begin{cases} d_{s^{i,j}} + e_0 \sum\limits_{u \in V_s^{i,j}} \sum\limits_{v \in V_s^{i,j}} w(u,v) + e_2 \sum\limits_{u \in V_s^{i,j}} \sum\limits_{v \in V - V_s^{i,j}} w(u,v) & \text{if } \left| V_s^{i,j} \right| = 1, \\ d_{s^{i,j}} + e_1 \sum\limits_{u \in V_s^{i,j}} \sum\limits_{v \in V_s^{i,j}} w(u,v) + e_2 \sum\limits_{u \in V_s^{i,j}} \sum\limits_{v \in V - V_s^{i,j}} w(u,v) & \text{if } \left| V_s^{i,j} \right| > 1, \end{cases} \tag{21}$$

$$d_s^{i,j} = \begin{cases} \infty & \text{if } \left| V_s^{i,j} \right| = 1 \text{ and } b_s(u) + t_0 w(u,u) + t_2 \sum\limits_{v \in V_s - V_s^{i,j}} w(u,v) > q \text{ for any } u \in V_s^{i,j}, \\ \infty & \text{if } \left| V_s^{i,j} \right| > 1 \text{ and } b_s(u) + t_1 w(u,u) + t_2 \sum\limits_{v \in V_s - V_s^{i,j}} w(u,v) > q \text{ for any } u \in V_s^{i,j}, \\ c_s^{i,j} & \text{otherwise,} \end{cases} \tag{22}$$

$$P_s = \begin{cases} V_s^{i,j} \cup P_{s^{i,j}} & \text{for any } i, j \text{ such that } d_i = d_{s^{i,j}} \text{ if } s \ne s^n, \\ \{\} & \text{if } s = s^n. \end{cases} \tag{23}$$

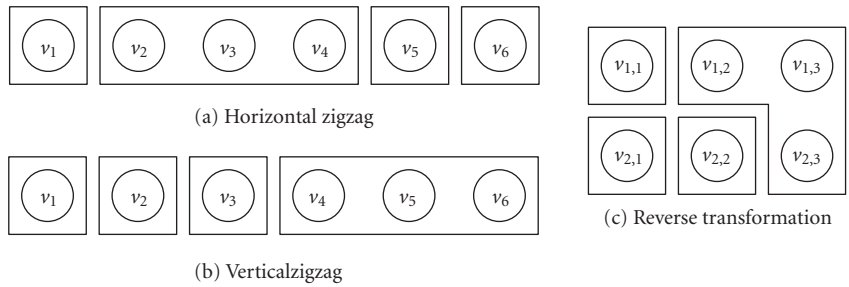


Figure 9: Partitions. The two linear partitions are transformed back and then merged together.

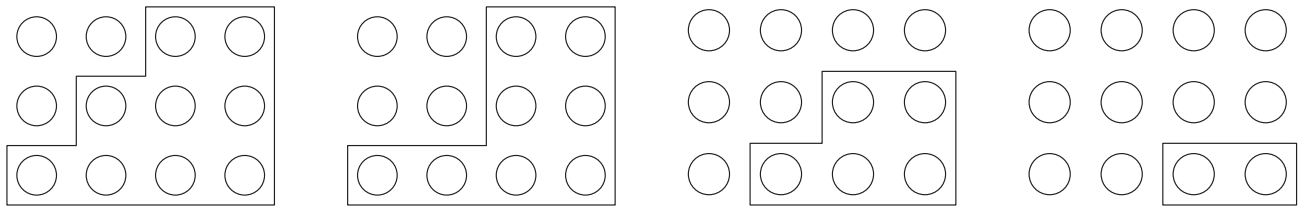


Figure 10: Examples of staircased-shapes sets. The enclosed cores make up a staircase-shaped set.

Unfortunately, the running time of StaRMA is $O(nm(n+m)!/n!m!)$ for an $m \times n$ rectangle. This is still acceptable when the number of cores is small, about 100 cores. If we also restrict the sub-instances to be rectangular, then we can have an algorithm that finds the best partition in polynomial time.

7.4. Carving Rectangular Partitions. In this section, we restrict all sub-instances as well as blocks to be rectangular. We lose in terms of optimality, but we gain much more in terms of the size of the problems we can solve in a reasonable amount of time. From our experiments, we see that we do not sacrifice much in terms of optimality either.

Since rectangles can be uniquely identified by two points, we will label our sub-instances by two points. For an instance $I = \langle V, w, t_0, e_0, t_1, e_1, t_2, e_2, b, q\rangle$ of rectangular MA, let $I_{x,y}$ be the sub-instance that contains a staircase-shaped set $V_{x,y} \subseteq V$, where $x = (x_i, x_j)$ and $y = (y_i, y_j)$ are two pairs such that $x_i \leq y_i$ and $x_j \leq y_j$. We can define $I_{x,y}$ as $\langle V_{x,y}, w, t_0, e_0, t_1, e_1, t_2, e_2, b_{x,y}, q\rangle$, where $V_{x,y}$ and $b_{x,y}$ are defined as follows:

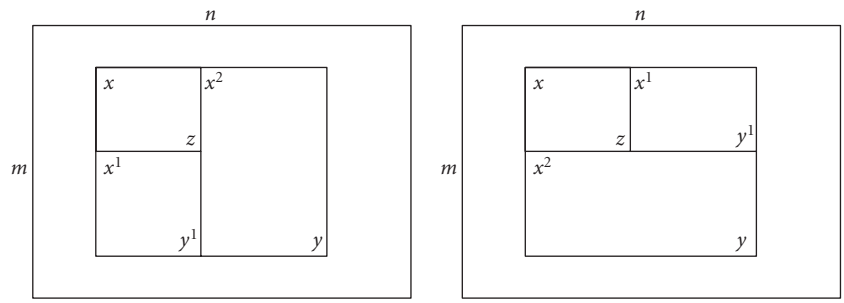
$$V_{x,y} = \left\{v_{i,j} \mid x_i \leq i \leq y_i,\ x_j \leq j \leq y_j\right\},$$
$$b_{x,y}(u) = b(u) + t_2 \sum_{v \in V - V_{x,y}} w(u,v) \quad \forall u \in V_{x,y}. \tag{24}$$

For each sub-instance $I_{x,y}$, let $P_{x,y}$ be an optimal partition that satisfies the timing constraint. Let $d_{x,y}$ be the energy consumption of $P_{x,y}$ or ∞ if we are unable to find a partition for $I_{x,y}$ that can meet the timing constraint. Let $z = (z_i, z_j)$ be a pair such that $x_i \leq z_i \leq y_i$ and $x_j \leq z_j \leq y_j$, and $V^z_{x,y} = \{v_{i,j} | x_i \leq i \leq z_i \text{and} x_j \leq j \leq z_j\}$. Suppose that $V^z_{x,y}$ is a block in $P_{x,y}$, then there are two configurations of sub-instances with two sub-instances each. In configuration 1, shown in Figure 11(a), sub-instance 1 is to the left of sub-instance 2. The two sub-instances are I_{x^1,y^1} and $I_{x^2,y}$, where $x^1 = (z_i + 1, x_j)$, $y^1 = (y_i, z_j)$, and $x^2 = (x_i, z_j + 1)$. In configuration 2, shown in Figure 11(b), sub-instance 1 is above sub-instance 2. The two sub-instances in this configuration are I_{x^1,y^1} and $I_{x^2,y}$, where $x^1 = (x_i, z_j+1)$, $y^1 = (z_i, y_j)$, and $x^2 = (z_i+1, x_j)$.

Let $c^{z,1}_{x,y}$ be the minimum energy necessary for $V_{x,y}$ if $V^z_{x,y}$ is a block in $P_{x,y}$ and we use configuration 1. Conversely, let $c^{z,2}_{x,y}$ be the minimum energy necessary for $V_{x,y}$ if $V^z_{x,y}$ is a block in $P_{x,y}$ and we use configuration 2. Similarly, Let $d^{z,1}_{x,y}(d^{z,2}_{x,y})$ be ∞ if no partition in configuration 1(7) that has $V^z_{x,y}$ as a block satisfies the timing constraints. Otherwise, let $d^{z,1}_{x,y}(d^{z,2}_{x,y})$ be $c^{z,1}_{x,y}(c^{z,2}_{x,y})$. $d_{x,y}$, $c^{z,1}_{x,y}$, $c^{z,2}_{x,y}$, $d^{z,1}_{x,y}$, and $d^{z,2}_{x,y}$ can be defined recursively as shown in (25), (26), (27), (28), and (29), respectively.

During the computation of $d_{x,y}$, we record the optimal value of z and configuration by recording the corresponding partitions in $P_{x,y}$. Let $P_{x,y} = \{\}$ for any x, y such that $x_i > y_i$ or $x_j > y_j$. For all x, y where $x_i \leq y_i$ and $x_j \leq y_j$, let z' be the optimal value of z used to compute $d_{x,y}$. If configuration 1 is used, then $P_{x,y} = \{V^{z'}_{x,y}\} \cup P_{(z'_i+1,x_j),(y_i,z'_j)} \cup P_{(x_i,z'_j+1),y}$. If configuration 2 is used, then $P_{x,y} = \{V^{z'}_{x,y}\} \cup P_{(x_i,z'_j+1),(z'_i,y_j)} \cup P_{(z'_i+1,x_j),y}$. If $d_{x,y} = \infty$, then we are unable to find a partition for $I_{x,y}$ that satisfies the timing requirement, and $P_{x,y}$ is undefined.

Note that $I_{(1,1),(m,n)} = I$, $P_{(1,1),(m,n)}$ is an optimal partition, and $d_{(1,1),(m,n)}$ is the minimum energy necessary to meet the timing constraint corresponding to $P_{(1,1),(m,n)}$. If $d_{(1,1),(m,n)} = \infty$, then we are unable to find a partition for I that consists of only rectangular blocks that will satisfy the timing constraint.



(a) Configuration 1 (b) Configuration 2

Figure 11: Sub-instance configurations. The rectangles cover the areas covered by the sub-instances.

$$d_{x,y} = \begin{cases} 0 & \text{if } x_i > y_i \text{ or } x_j > y_j, \\ \min\limits_{x_i \le z_i \le y_i} \left\{ \min\limits_{x_j \le z_j \le y_j} \left\{ \min\limits_{1 \le i \le 2} \left\{ d^{z,i}_{x,y} \right\} \right\} \right\} & \text{otherwise,} \end{cases} \tag{25}$$

$$c^{z,1}_{x,y} = \begin{cases} d_{(z_i+1,x_j),(y_i,z_j)} + d_{(x_i,z_j+1),y} \\ \quad + e_0 \sum\limits_{u \in V^z_{x,y}} \sum\limits_{v \in V^z_{x,y}} w(u,v) + e_2 \sum\limits_{u \in V^z_{x,y}} \sum\limits_{v \in V - V^z_{x,y}} w(u,v) & \text{if } \left| V^z_{x,y} \right| = 1, \\ d_{(z_i+1,x_j),(y_i,z_j)} + d_{(x_i,z_j+1),y} \\ \quad + e_1 \sum\limits_{u \in V^z_{x,y}} \sum\limits_{v \in V^z_{x,y}} w(u,v) + e_2 \sum\limits_{u \in V^z_{x,y}} \sum\limits_{v \in V - V^z_{x,y}} w(u,v) & \text{if } \left| V^z_{x,y} \right| > 1, \end{cases} \tag{26}$$

$$c^{z,2}_{x,y} = \begin{cases} d_{(x_i,z_j+1),(z_i,y_j)} + d_{(z_i+1,x_j),y} \\ \quad + e_0 \sum\limits_{u \in V^z_{x,y}} \sum\limits_{v \in V^z_{x,y}} w(u,v) + e_2 \sum\limits_{u \in V^z_{x,y}} \sum\limits_{v \in V - V^z_{x,y}} w(u,v) & \text{if } \left| V^z_{x,y} \right| = 1, \\ d_{(x_i,z_j+1),(z_i,y_j)} + d_{(z_i+1,x_j),y} \\ \quad + e_1 \sum\limits_{u \in V^z_{x,y}} \sum\limits_{v \in V^z_{x,y}} w(u,v) + e_2 \sum\limits_{u \in V^z_{x,y}} \sum\limits_{v \in V - V^z_{x,y}} w(u,v) & \text{if } \left| V^z_{x,y} \right| > 1, \end{cases} \tag{27}$$

$$d^{z,1}_{x,y} = \begin{cases} \infty & \text{if } \left| V^z_{x,y} \right| = 1 \text{ and } b_{x,y}(u) + t_0 w(u,u) + t_2 \sum\limits_{v \in V_{x,y} - V^z_{x,y}} w(u,v) > q \text{ for any } u \in V^z_{x,y}, \\ \infty & \text{if } \left| V^z_{x,y} \right| > 1 \text{ and } b_{x,y}(u) + t_1 w(u,u) + t_2 \sum\limits_{v \in V_{x,y} - V^z_{x,y}} w(u,v) > q \text{ for any } u \in V^z_{x,y}, \\ c^{z,1}_{x,y} & \text{otherwise,} \end{cases} \tag{28}$$

$$d^{z,2}_{x,y} = \begin{cases} \infty & \text{if } \left| V^z_{x,y} \right| = 1 \text{ and } b_{x,y}(u) + t_0 w(u,u) + t_2 \sum\limits_{v \in V_{x,y} - V^z_{x,y}} w(u,v) > q \text{ for any } u \in V^z_{x,y}, \\ \infty & \text{if } \left| V^z_{x,y} \right| > 1 \text{ and } b_{x,y}(u) + t_1 w(u,u) + t_2 \sum\limits_{v \in V_{x,y} - V^z_{x,y}} w(u,v) > q \text{ for any } u \in V^z_{x,y}, \\ c^{z,2}_{x,y} & \text{otherwise.} \end{cases} \tag{29}$$

Input: An instance I of Rectangular MA.
Output: P_s and d_s.
(1) $s \leftarrow$ an $(m+1)$-tuple
(2) $s[0] \leftarrow n+1$
(3) **for** $i \leftarrow 1$ to m **do**
(4) $\quad s[i] \leftarrow n$
(5) **end for**
(6) $P_s \leftarrow \{\}$
(7) $d_s \leftarrow 0$
(8) **while** $s[1] > 0$ **do**
(9) $\quad i \leftarrow m$
(10) $\quad s[i] \leftarrow s[i] - 1$
(11) $\quad$ **while** $s[i] = -1$ **do**
(12) $\quad\quad i \leftarrow i - 1$
(13) $\quad\quad s[i] \leftarrow s[i] - 1$
(14) $\quad$ **end while**
(15) $\quad$ **for** $j \leftarrow i+1$ to m **do**
(16) $\quad\quad s[j] \leftarrow s[i]$
(17) $\quad$ **end for**
(18) $\quad d_s \leftarrow \infty$
(19) $\quad k_s \leftarrow$ an m-tuple
(20) $\quad$ **for** $i \leftarrow 1$ to m **do**
(21) $\quad\quad k_s[i] \leftarrow i - 1$
(22) $\quad\quad$ **while** $s[k_s[i]] = s[i]$ **do**
(23) $\quad\quad\quad k_s[i] \leftarrow k_s[i] - 1$
(24) $\quad\quad$ **end while**
(25) $\quad$ **end for**
(26) $\quad$ **for** $i \leftarrow 1$ to m **do**
(27) $\quad\quad$ **for** $j \leftarrow s[i]+1$ to $\min(s[k_s[i]], n)$ **do**
(28) $\quad\quad\quad$ Compute $c_s^{i,j}$ and $d_s^{i,j}$.
(29) $\quad\quad\quad$ **if** $d_s^{i,j} < d_s$ **then**
(30) $\quad\quad\quad\quad d_s \leftarrow d_s^{i,j}$
(31) $\quad\quad\quad\quad P_s \leftarrow \{V_s^{i,j}\} \cup P_{s^{i,j}}$
(32) $\quad\quad\quad$ **end if**
(33) $\quad\quad$ **end for**
(34) $\quad$ **end for**
(35) **end while**

Algorithm 3: Staircase rectangular memory arrangement (StaRMA).

Input: An instance I of Rectangular MA
Output: A near optimal partition $P_{(1,1),(m,n)}$ and its energy consumption $d_{(1,1),(m,n)}$
(1) **for** $\ell_i \leftarrow 1$ to m **do**
(2) $\quad$ **for** $\ell_j \leftarrow 1$ to n **do**
(3) $\quad\quad$ **for** $y_i \leftarrow \ell_i$ to m **do**
(4) $\quad\quad\quad$ **for** $y_j \leftarrow \ell_j$ to n **do**
(5) $\quad\quad\quad\quad x \leftarrow (y_i - \ell_i + 1)$
(6) $\quad\quad\quad\quad V_{x,y} \leftarrow \{v_{i,j} \mid x_i \le i \le y_i \text{ and } x_j \le j \le y_j\}$
(7) $\quad\quad\quad\quad d_{x,y} \leftarrow \infty$
(8) $\quad\quad\quad\quad P_{x,y} \leftarrow \{\}$
(9) $\quad\quad\quad\quad$ **for** $z_i \leftarrow x_i$ to y_i **do**
(10) $\quad\quad\quad\quad\quad$ **for** $z_j \leftarrow x_j$ to y_j **do**
(11) $\quad\quad\quad\quad\quad\quad V_{x,y}^z \leftarrow \{v_{i,j} \mid x_i \le i \le z_i$ and $x_j \le j \le z_j\}$
(12) $\quad\quad\quad\quad\quad\quad$ Compute $c_{x,y}^{z,1}$, $c_{x,y}^{z,2}$, $d_{x,y}^{z,1}$, and $d_{x,y}^{z,2}$.
(13) $\quad\quad\quad\quad\quad\quad$ **if** $d_{x,y}^{z,1} < d_{x,y}$ **then**
(14) $\quad\quad\quad\quad\quad\quad d_{x,y} \leftarrow d_{x,y}^{z^1}$
(15) $\quad\quad\quad\quad\quad\quad P_{x,y} \leftarrow \{V_{x,y}^{z'}\} \cup P_{(x_i,z'_{j+1}),y} \cup P_{(z'_{i+1},x_j),(y_i,z'_j)}$
(16) $\quad\quad\quad\quad\quad\quad$ **end if**
(17) $\quad\quad\quad\quad\quad\quad$ **if** $d_{x,y}^{z,2} < d_{x,y}$ **then**
(18) $\quad\quad\quad\quad\quad\quad\quad d_{x,y} \leftarrow d_{x,y}^{z^2}$
(19) $\quad\quad\quad\quad\quad\quad\quad P_{x,y} \leftarrow \{V_{x,y}^{z'}\} \cup P_{(z'_{i+1},x_j),y} \cup P_{(x_i,z'_{j+1}),(z'_i,y_j)}$
(20) $\quad\quad\quad\quad\quad\quad$ **end if**
(21) $\quad\quad\quad\quad\quad$ **end for**
(22) $\quad\quad\quad\quad$ **end for**
(23) $\quad\quad\quad$ **end for**
(24) $\quad\quad$ **end for**
(25) $\quad$ **end for**
(26) **end for**

Algorithm 4: Carving rectangular memory arrangement (CaRMA).

A polynomial time algorithm to compute $P_{x,y}$ and $d_{x,y}$, Carving Rectangular Memory Arrangement (CaRMA), is shown in Algorithm 4. Its running time is $O(m^5n^5)$ for an $m \times n$ rectangle. It starts with small sub-instances and loops through progressively larger sub-instances. Since each sub-instance only references sub-instances smaller than the current sub-instance, all needed sub-instances have already been solved. Lines 3-4 loops through all the different y's. Lines 9-10 loops through all the possible z's. For each $V_{x,y}^z$, we compute the energy consumption on line 12. If configuration 1 uses less energy, lines 13–16 will record the corresponding $P_{x,y}$. If configuration 2 uses less energy, lines 17–20 will record the corresponding $P_{x,y}$.

8. Experiments

We evaluate ZiRMA, CaRMA, and StaRMA by comparing the memory arrangements generated to both an all shared memory arrangement and an all private memory arrangement. We do not explicitly evaluate OLMA since it is used in ZiRMA. We run experiments on two sets of instances. The instances in the first set are randomly generated, while the second set are extracted from digital signal processing (DSP) benchmarks from DSPStone [1]. For these experiments, we only consider the energy consumption of memory access operations.

8.1. Random Instances. We generate 800 random rectangular instances with varying degrees of memory access locality and penalty. The locality describes the memory accesses among cores. Clumpy means that most memory accesses are within groups of cores, between which there is little interaction. Diffuse means that memory accesses are distributed evenly among the cores, and it is difficult to divide them into groups. The penalty of remote accesses with respect to local accesses may either be mild or severe. Mild penalty means that the energy cost for accessing remote data is only two times the energy cost for accessing local data. Conversely, severe penalty means that the energy cost for accessing data

Table 8: Improvements for randomly generated instances.

Locality	Penalty	ZiRMA		CaRMA		StaRMA		CaRMA / ZiRMA
		All shared	All private	All shared	All private	All shared	All private	
Clumpy	mild	38%	4%	42%	10%	42%	10%	6%
	severe	51%	7%	56%	17%	56%	17%	10%
Diffuse	mild	40%	9%	42%	11%	42%	11%	3%
	severe	54%	14%	56%	19%	56%	19%	5%

Table 9: Improvements for DSP benchmarks.

Instance	Penalty	ZiRMA		CaRMA		StaRMA	
		All shared	All private	All shared	All private	All shared	All private
allpole	mild	6%	6%	17%	18%	17%	18%
	severe	7%	32%	22%	43%	22%	43%
deq	mild	5%	10%	17%	21%	17%	21%
	severe	7%	35%	21%	44%	21%	44%
elliptic	mild	21%	8%	21%	8%	21%	8%
	severe	8%	13%	23%	27%	23%	27%
iir	mild	5%	13%	17%	23%	17%	23%
	severe	7%	36%	20%	45%	20%	45%
lattice	mild	18%	11%	18%	11%	18%	11%
	severe	7%	20%	23%	33%	23%	33%
Average	mild	11%	10%	18%	16%	18%	16%
	severe	7%	27%	22%	38%	22%	38%

in a remote memory is several times greater than the energy cost for accessing data in a local memory.

The results from this set of random experiments are shown in Table 8. We generated 200 instances for each combination of memory access locality and penalty. The third, fifth, and seventh columns show how much better ZiRMA, CaRMA, and StaRMA perform than an all shared memory arrangement, respectively. The fourth, sixth, and eighth columns show how much better ZiRMA, CaRMA, and StaRMA perform than an all private memory arrangement, respectively. The ninth column show how much better CaRMA performs than ZiRMA.

8.2. DSP Instances. In addition to randomly generated instances, we perform experiments on instances extracted from DSP benchmarks. The benchmarks we use are an all pole filter (allpole), a differential equation solver (deq), an elliptic filter (elliptic), an infinite impulse response filter (iir), and a 4-stage lattice filter (lattice). For these instances, we unfold the benchmarks and perform the experiments on 2×4 rectangular instances with varying memory access penalty.

The results from this set of random experiments are shown in Table 9. The third, fifth, and seventh columns show how much better ZiRMA, CaRMA, and StaRMA perform than an all shared memory arrangement, respectively, while the fourth, sixth, and eighth columns show how much better ZiRMA, CaRMA, and StaRMA perform than an all private memory arrangement, respectively. The last two rows show the average improvement for both mild and severe penalties.

In summary, on instances extracted from DSP benchmarks, CaRMA and StaRMA perform an average of 18% better than an all shared memory arrangement for cases with mild memory-access penalty and an average of 38% better than an all private memory arrangement for cases with severe memory access penalty.

8.3. Computation Times. From previous sections, we know that the running times of ZiRMA, CaRMA, and StaRMA are $O(m^4n^4)$, $O(m^5n^5)$, and $O(nm(n+m)!/n!m!)$, respectively. We compare the time it takes these algorithms to process an instance. Figure 12 shows the computation times for these algorithms for instances of differing sizes. From the graph, we can see that ZiRMA and CaRMA have similar computation times, and StaRMA's computation times grow much faster.

8.4. Analysis. From these experiments, we can see that all algorithms perform the same for instances with only a few cores, and ZiRMA performs the worst for instances with many cores. For larger instances, the linear arrays that ZiRMA considers deviate more from the rectanguar mesh. Many of the small sharings in the middle of the mesh are not possible in ZiRMA since the zigzag segment that ZiRMA considers is quite long. Thus, ZiRMA struggles with large rectangular meshes, especially square meshes. We also see that CaRMA performs as well as StaRMA in most cases. As for the computation time, CaRMA takes only a little more time than ZiRMA. Thus, CaRMA produces the best results in a reasonable amount of time.

Table 10: Summary of experimental results.

Instance Type	All shared	All private
Random	49%	14%
DSP	20%	27%

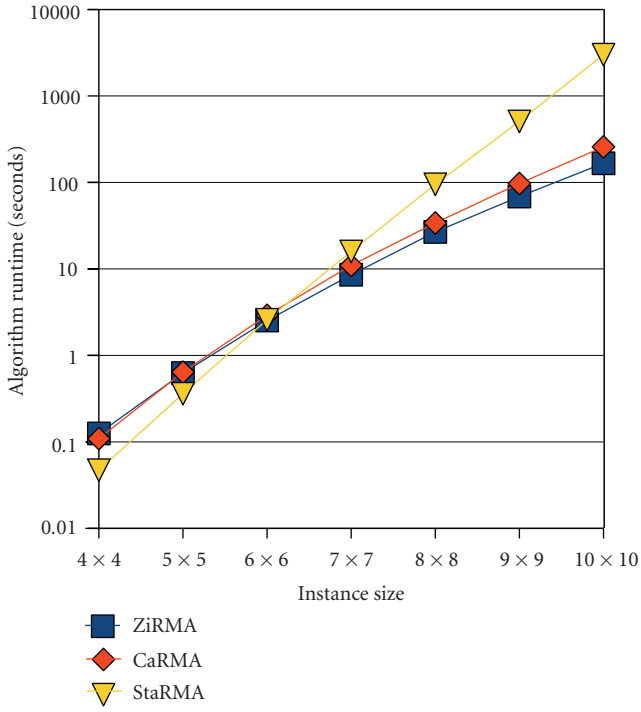


Figure 12: Runtimes for ZiRMA, CaRMA, and StaRMA.

A summary of the experimental results for CaRMA is shown in Table 10. The results from the random instances are all averaged together. On average, CaRMA produces arrangements that consume 49% less energy than an all shared memory arrangement and 14% less energy than an all private memory arrangement for randomly generated instances. For DSP benchmarks, CaRMA produces arrangements that, on average, consume 20% less energy than an all shared memory arrangement and 27% less energy than an all private memory arrangement.

9. Conclusion

We study the Memory Arrangement Problem (MA). We prove that if arbitrary cores can share a memory, then MA is NP-complete. We present an efficient optimal algorithm for solving linear instances of MA and extend the algorithm to solve rectangular cases of MA. We present an optimal algorithm for solving rectangular cases of MA where only rectangular blocks of cores share memories and an efficient heuristic to obtain good memory arrangements in a reasonable amount of time. On average, we can produce arrangements that consume 49% less energy than an all shared memory arrangement and 14% less energy than an all private memory arrangement for randomly generated instances. For DSP benchmarks, we can produce arrangements that, on average, consume 20% less energy than an all shared memory arrangement and 27% less energy than an all private memory arrangement.

Acknowledgments

This work is partially supported by NSF IIS-0513669, HK CERG 526007, HK GRF 123609, NSFC 60728206, and Changjiang Honorary Chair Professor Scholarship.

References

[1] V. Živojnović, J. M. Velarde, C. Schläger, and H. Meyr, "DSPSTONE: a DSP-oriented benchmarking methodology," in *Proceedings of the International Conference on Signal Processing and Technology (ICSPAT '94)*, Dallas, Tex, USA, 1994.

[2] Y. Zhao, C. J. Xue, M. Li, and B. Hu, "Energy-aware register file re-partitioning for clustered VLIW architectures," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '09)*, pp. 805–810, Yokohama , Japan, January 2009.

[3] M. Wang, Z. Shao, H. Liu, and C. J. Xue, "Minimizing leakage energy with modulo scheduling for VLIW DSP processors," in *Proceedings of the Distributed Embedded Systems: Design, Middleware and Resourcess (DIPES '08)*, B. Kleinjohann, L. Kleinjohann, and W. Wolf, Eds., vol. 271 of *IFIP International Federation for Information Processing*, pp. 111–120, Springer, Milano, Italy, 2008.

[4] M. Qiu, Z. Jia, C. Xue, Z. Shao, and E. H.-M. Sha, "Voltage assignment with guaranteed probability satisfying timing constraint for real-time multiproceesor DSP," *Journal of VLSI Signal Processing Systems*, vol. 46, no. 1, pp. 55–73, 2007.

[5] G. Hua, M. Wang, Z. Shao, H. Liu, and C. Xue, "Real-time loop scheduling with energy optimization via dvs and abb for multi-core embedded system," in *Proceedings of Embedded and Ubiquitous Computing (EUC '07)*, T.-W Kuo, E. H.-M. Sha, M. Guo, L. T. Yang, and Z. Shao, Eds., vol. 4808 of *Lecture Notes in Computer Science*, pp. 1–2, Springer, Taipei, Taiwan, 2007.

[6] B. Saha, A.-R. Adl-Tabatabai, R. L. Hudson, C. M. Chi, and B. Hertzberg, "McRT-STM: a high performance software transactional memory system for a multi-core runtime," in *Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '06)*, vol. 2006, pp. 187–197, ACM, New York, NY, USA, 2006.

[7] R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in multi-core architectures: understanding mechanisms, overheads and scaling," in *Proceedings of the 32nd annual International Symposium on Computer Architecture (ISCA '05)*, pp. 408–419, IEEE Computer Society, Washington, DC, USA, 2005.

[8] C. Xue, Z. Shao, M. Liu, M. Qiu, and E. H.-M. Sha, "Loop scheduling with complete memory latency hiding on multi-core architecture," in *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS '06)*, vol. 1, pp. 375–382, 2006.

[9] V. Suhendra, C. Raghavan, and T. Mitra, "Integrated scratch-pad memory optimization and task scheduling for MPSoC architectures," in *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '06)*, pp. 401–410, ACM, Seoul, South Korea, 2006.

[10] L. Zhang, M. Qiu, W.-C. Tseng, and E. H.-M. Sha, "Variable partitioning and scheduling for MPSoC with virtually shared scratch pad memory," *Journal of Signal Processing Systems*, pp. 1–19, 2009.

[11] M. Kandemir, J. Ramanujam, and A. Choudhary, "Exploiting shared scratch pad memory space in embedded multiprocessor systems," in *Proceedings of the 39th Conference on Design Automation (DAC '02)*, pp. 219–224, ACM, New Orleans, La, USA, 2002.

[12] F. Angiolini, L. Benini, and A. Caprara, "Polynomial-time algorithm for on-chip scratchpad memory partitioning," in *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES '03)*, pp. 318–326, ACM, San Jose, Calif, USA, 2003.

[13] S. Udayakumaran and R. Barua, "Compiler-decided dynamic memory allocation for scratch-pad based embedded systems," in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '03)*, pp. 276–286, ACM, San Jose, Calif, USA, 2003.

[14] G. E. Suh, L. Rudolph, and S. Devadas, "Dynamic partitioning of shared cache memory," *Journal of Supercomputing*, vol. 28, no. 1, pp. 7–26, 2004.

[15] M. Chu, R. Ravindran, and S. Mahlke, "Data access partitioning for fine-grain parallelism on multicore architectures," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '07)*, pp. 369–378, IEEE Computer Society, Washington, DC, USA, 2007.

[16] C.-G. Lyuh and T. Kim, "Memory access scheduling and binding considering energy minimization in multi-bank memory systems," in *Proceedings of the 41st Annual Conference on Design Automation (DAC '04)*, pp. 81–86, ACM, San Diego, Calif, USA, 2004.

[17] S. Meftali, F. Gharsalli, F. Rousseau, and A. A. Jerraya, "An optimal memory allocation for application-specific multiprocessor system-on-chip," in *Proceedings of the 14th International Symposium on System Synthesis (ISSS '01)*, pp. 19–24, ACM, Montréal, Canada, 2001.

[18] O. Ozturk, M. Kandemir, M. J. Irwin, and S. Tosun, "Multi-level on-chip memory hierarchy design for embedded chip multiprocessors," in *Proceedings of the 12th International Conference on Parallel and Distributed Systems (ICPADS '06)*, pp. 383–390, IEEE Computer Society, Washington, DC, USA, 2006.

[19] O. Ozturk, M. Kandemir, G. Chen, M. J. Irwin, and M. Karakoy, "Customized on-chip memories for embedded chip multiprocessors," in *Proceedings of the Conference on Asia South Pacific Design Automation (ASP-DAC '05)*, pp. 743–748, ACM, Shanghai, China, 2005.