SAMS: Single-Affiliation Multiple-Stride Parallel Memory Scheme

Chunyang Gou, Georgi Kuzmanov, Georgi N. Gaydadjiev
Computer Engineering Lab
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology, The Netherlands
{C.Gou, G.K.Kuzmanov, G.N.Gaydadjiev}@tudelft.nl

ABSTRACT

In this paper, we analyze the problem of supporting conflict-free access for multiple stride families in parallel memory schemes targeted for SIMD processing systems. We propose a Single-Affiliation Multiple-Stride (SAMS) scheme to support both unit-stride and strided conflict-free vector memory accesses. We compare our scheme against other previously proposed techniques using buffers and inter-vector out-of-order access. The main advantage of our proposal is that the atomic parallel access is supported without limiting the vector lengths. This provides better support when short vectors are considered. Our scheme also has the merit of better memory module utilization compared to the solutions with additional modules. Synthesis results for reconfigurable Virtex2-Pro FPGA technology indicate that the address translation of the SAMS scheme has efficient hardware implementation, which has a logic delay of less than 3 *ns* and trivial hardware resource utilization.

Categories and Subject Descriptors

B.3.m [**Hardware**]: MEMORY STRUCTURES—*Miscellaneous*;
C.1.2 [**PROCESSOR ARCHITECTURES**]: Multiple Data Stream Architectures (Multiprocessors)—*Single-instruction-stream, multiple-data-stream processors (SIMD)*

General Terms

Design, Experimentation

Keywords

Parallel memory schemes, multimodule memory, memory access, stride

1. INTRODUCTION

One of the most critical design challenges in SIMD processors is imposed by the memory subsystem, which is required to deliver sustained high bandwidth at reasonable latency [6, 13]. To meet these challenges, memory subsystems with multiple memory modules have been widely considered. Parallel (or multimodule) memories were introduced in the early years of building high performance processors [2] and later extensively adopted in vector supercomputers [18, 7]. Nowadays, there is a trend that general purpose systems are utilizing parallel memories in their memory hierarchy, such as the multibank on-chip caches in Niagara [11] and Opteron [10], multislice caches in Power processors [21, 19, 14], parallel on-chip eDRAM banks in VIRAM processor [13], and interleaved DRAM banks in Rambus and other commercial-off-the-shelf monolithic DRAM chips. For simplicity of the module assignment hardware implementation, the number of memory modules was chosen as a power of two in most of these real systems. For efficient hardware utilization, the designers prefer systems with non-redundant memory schemes, i.e. schemes where each and every memory module should be referenced by any memory access. This paper addresses non-redundant memory systems with a power of two memory modules. We introduce the SAMS (Single-Affiliation Multiple-Stride) parallel memory scheme, which overcomes the problem of conflict-free[1] access across stride families[2] in multimodule parallel memory systems. The specific contributions of our proposal are:

- We propose the SAMS scheme: to our best knowledge, this is the first in-order parallel memory scheme, which supports both strided[3] and unit-stride(precise definitions provided in Section 2.3) conflict-free vector access;
- Synthesis results on Virtex2-Pro reconfigurable technology suggest short critical path (in the proximity of 4 ns) of the address translation logic. This is a strong indication for the feasibility of the proposed scheme in practical parallel memory systems.

The remainder of the paper is organized as follows. In Section 2, we present the background and motivation of the proposed scheme. In Section 3, the construction procedure and the mathematical equations of the SAMS scheme are described, and the initial hardware implementation and synthesis results of the scheme are illustrated in Section 4. The

[1]Please refer to Section 2.1 for detailed explanation of "conflict-free access".
[2]See the definitions in Section 2.3.
[3]By "strided" we refer to even strides in this paper, as odd strides (including unit stride) conflict-free accesses are well supported by the simple low-order interleaving scheme [22].

major differences between our proposal and related works are described in Section 5, followed by some discussions about the design space and application of SAMS in Section 6. Finally we conclude the paper with some suggestions on future work in Section 7.

2. BACKGROUND AND MOTIVATION

In this section, we will introduce some background on parallel memory schemes, which play a central role in parallel memory systems. We also present some of the key existing techniques in parallel memory schemes. Then we will present the limitation in nonredundant parallel memory schemes, which motivates us to this work.

2.1 Parallel Memory Schemes

The parallel memory schemes are the main means to determine the performance and the hardware complexity of the parallel memory subsystems. Given a specific physical memory organization and resources, these schemes determine the mapping from the linear address space to the physical location identifiers, such as the module number and row address. In other words, the memory translation scheme determines how to distribute data to different memory banks, in order to better service memory references. Vector access, defined by an address stream with a constant offset between any two consecutive addresses, is one of the most important memory reference patterns in SIMD applications. Traditional parallel memory schemes in vector computers provide conflict-free access for a *single* stride family. To solve the module conflicts encountered with the cross stride family accesses, several enhancements have been previously proposed, including the use of dynamic memory schemes [5, 4], use of buffers [3], and use of more memory modules (i.e. memory scheme with redundant modules) [3], and out-of-order vector access [22].

These traditional multimodule memory schemes assume that the memory access time is atomic and it is much larger than the processor cycle time. Consequently, to keep up with the data access demand of the fast processors, different memory schemes were proposed to make multiple memory modules work together in an *interleaved* manner to service single cycle data accesses from the processor. Particularly, if the number of memory modules equals to their latency counted in processor cycles, the scheme is called a *matched* memory system, as in this case, the interleaved memory modules are capable of servicing the processor at the throughput of one datum per processor cycle, if the data to be referenced in one access are located in different memory modules. This condition of uniform distribution of memory references is called **conflict-free** access in traditional interleaving schemes. In conflict-free access, data could be accessed in parallel as there is no module conflict during the memory access.

With the advance of the semiconductor technologies, however, the parallel memory modules of traditional vector computers could be (partially) integrated into the processor chip working as a local store like the Cell processor [9]. Thus, on the one hand, for medium-sized on-chip memory arrays (such as the L1 caches in GPPs), it is not difficult to support one access per processor cycle. On the other hand, to exploit the available on-chip resources more efficiently, multiple SIMD data paths are deployed in modern processors. Furthermore, accessing multiple memory data items in a single processor cycle is desirable in order to use the multiple

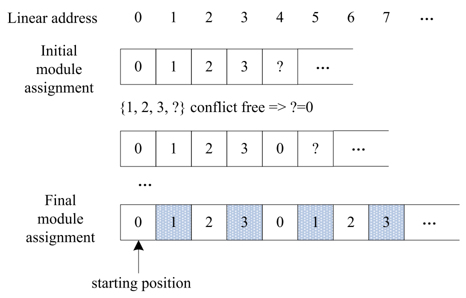

Figure 1: Inherent limitation in multimodule memory assignment

data paths efficiently. Therefore, we assume a memory organization where the multimodule SRAM memories are deployed as on-chip SIMD buffers[4] and each of them services one piece of data reference every clock cycle. In this work, we address such memory organizations and refer to them as "*fully-parallel* multimodule memory systems". In fully-parallel multimodule memory systems, the processor could access at most as many data items as the number of memory modules in one cycle, when the access is conflict-free. This is actually the goal of the underlying parallel memory schemes in such memory organizations.

2.2 Motivation: The Limitation in Nonredundant Parallel Memory Schemes

In traditional matched parallel memory schemes, it is impossible to simultaneously support both parallel unit-stride and arbitrarily strided access orders [22]. Figure 1 illustrates an example with four memory modules. Under the constraint of unit-stride conflict-free access, the module assignment function of the scheme is completely fixed. Note in Figure 1 the constant repeat of module assignment pattern of the first four addresses. When the system is accessed with stride 2, half of the memory modules are not utilized (note the shadowed cells in Figure 1). One additional limitation, not illustrated in Figure 1, is that any interleaving scheme optimized for even-stride conflict-free access could not support conflict-free unit-stride access at an arbitrary base addresses[5].

There is a large number of strided vector accesses in many scientific and engineering applications which have significant impact on the performance of the workloads on traditional vector supercomputers [1]. In the meanwhile, we certainly could not neglect the unit-stride access pattern, as it is the most common one in vectorized scientific and engineering applications [8, 12, 20]. Even in vectorized SPEC95 benchmarks it is the second most frequent stride [16]. Furthermore, there are many occasions in which simultaneous support of both unit-stride and strided memory accesses is desired, as the same data block is accessed with different stride types. When we have to access data in parallel memories with both unit and even stride, the problem occurs that we have to either modify the interleaving scheme (that is, to redistribute data to memory modules in a different way), or to have the scheme optimized for conflict-free access with

[4]Three are already many GPPs with multimodule on-chip SRAMs, see [14, 9, 11] for examples.

[5]Refer to Section 2.3 for the definition of "base address".

one type of access while suffer from the non-conflict-free access with the other. The former would incur data flushing into and reloading from the lower level memory in the memory hierarchy whenever there is a change of access stride, whereas the latter would introduce processor cycles wasted on waiting for the vector access.

As far as unit-stride access on stride-optimized parallel memory scheme is concerned, it is interesting to examine the *affiliation* properties of the scheme. We define affiliation in the following section, along with other relevant definitions.

2.3 Definitions

For the sake of clarity, we now give the definitions of key terms used in this paper. First, we define some terminologies used in general parallel memory systems, and then the specific ones used in our SAMS scheme will follow.

Definition 1. A sequence of independent memory access stream issued by the SIMD processor in parallel is called a **vector access**. A vector access could be either regular (with constant stride) or irregular (such as the scatter/gather memory access). However we only discuss regular vector accesses in this paper.

Definition 2. Base address is the first memory address in a given regular vector access stream.

Definition 3. Stride is the constant interval between subsequent memory addresses in a given regular vector access stream.

Definition 4. Unit stride denotes stride 1.

Definition 5. A **stride family** is a set of infinite number of strides, $\{S \| S = \sigma \cdot 2^s,\ s \in \mathbb{N},\ \sigma \text{ is odd}\}$. This follows the definitions given in [5, 4, 22].

Definition 6. The exponential part of the stride family $\{S \| S = \sigma \cdot 2^s,\ s \in \mathbb{N},\ \sigma \text{ is odd}\}$, s, is called the **stride family number**. The stride family number completely defines the set of strides belonging to a stride family. For example, stride family with family number 0 (we will say "stride family s" for short of "stride family with family number s" hereafter) is the stride set $\{1, 3, 5, 7, \cdots\}$ while stride family 1 is $\{2, 6, 10, 14, \cdots\}$.

Now we will give some definitions specific in our proposed SAMS scheme. Suppose a is the linear address and $m(a)$ is the module assignment function of a parallel memory scheme with N memory modules.

Definition 7. If address a satisfies $m(a) = m(a+\delta)$ (where $\delta < N$), address a has **forward-affiliation**.

Definition 8. If address a satisfies $m(a) = m(a-\delta)$ (where $\delta < N$), address a has **backward-affiliation**.

Definition 9. Forward-affiliation and backward-affiliation always occur in pairs. For instance, if address a has forward-affiliation $(m(a) = m(a + \delta))$ then the address $a + \delta$ has backward-affiliation. We call address a and its affiliated address $(a + \delta)$ an **affiliation-pair**.

Now, let us examine the meaning of affiliation. If there exist forward/backward-affiliations in a memory scheme, then the scheme is not conflict-free for parallel N unit-stride accesses at arbitrary base addresses. For instance, unit-stride accesses starting at addresses with forward-affiliation will result in module conflicts.

Definition 10. If address a is associated with only one single instance of affiliation (backward- or forward-), then it is a **single-affiliation** address.

Definition 11. If there exist addresses in a nonredundant parallel memory scheme with single-affiliation and none of them has multi-affiliation, then it is a **single-affiliation scheme**.

Note that in single-affiliation schemes, a single-affiliation address belongs to only an affiliation-pair. Single-affiliation parallel memory schemes make sure the module conflicts under unit-stride access are moderate in the sense that, if an address in linear address space causes module conflict within access at one base address, then it will never cause any other conflict within access at a different base address. We will illustrate how to make use of Single-affiliation parallel memory schemes to construct a memory system which is capable of supporting conflict-free strided accesses from multiple stride families.

3. SAMS: SINGLE-AFFILIATION MULTIPLE-STRIDE CONFLICT-FREE PARALLEL MEMORY SCHEME

In this section, we propose SAMS, the Single-Affiliation Multiple-Stride conflict-free parallel memory scheme. SAMS aims at supporting conflict-free unit-stride and strided memory accesses simultaneously, by first constructing a single-affiliation interleaving scheme, and then making data lines wider to solve the module conflicting problem in unit-stride access, which exists in the single-affiliation scheme.

3.1 Relax the Constraint from *Conflict-Free to Low Degree of Affiliation*

It has been observed that when a nonredundant parallel memory scheme is configured for conflict-free unit-stride access, it is incapable of supporting conflict-free strided access. Now that it is difficult to achieve the goal of adding conflict-free unit-stride access support at once, we propose to do this in two steps. First, we relax the constraint from *conflict-free* to *low degree of affiliation*, for unit-stride access. Second, for each and every memory module, we rearrange the groups of conflicting data items in unit-strided access into wide data lines so that they could be referenced during one access. The philosophy behind this idea is to restructure the memory modules. Traditionally, the memory module in a multimodule memory system is treated as a linear (one-dimensional) structure, and a memory interleaving scheme maps the linear address space from processor's view into multiple linearly structured storage units. In our approach, instead of one-dimensional, we model each memory module as a two-dimensional structure. Therefore, the aforementioned limitation in parallel memory schemes could hopefully be resolved by introducing a new dimension of access parallelism, namely the module data line. If the conflicting data items are located in the same data line, then they could be accessed in parallel with proper shuffling and selecting operations. The reason why low-affiliation schemes are preferable is that, as the degree of affiliation increases, the module data line grows wider because it has to be wide enough to accommodate all conflicting items. Consequently, the hardware for choosing the proper one(s) from the data line and shuffling data items from different modules becomes more complicated.

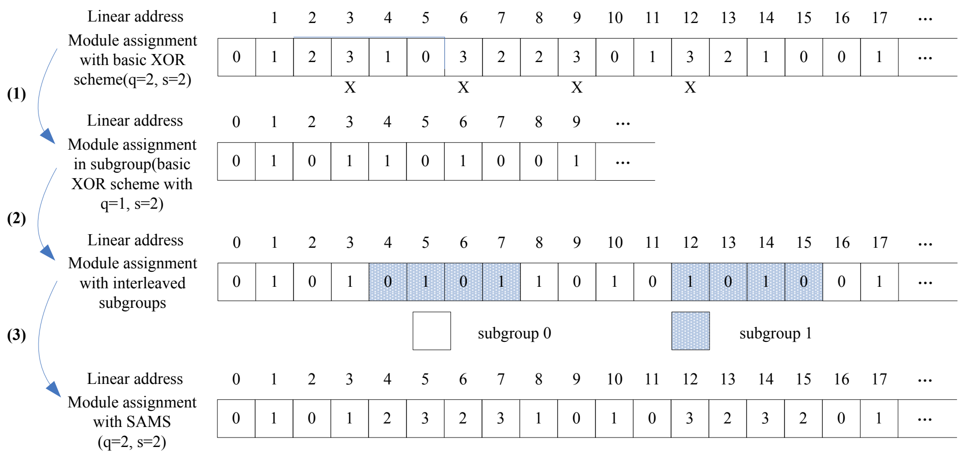

Figure 2: Example of construction of SAMS single-affiliation scheme with $q = 2,\ s = 2$

3.2 Hierarchical Single-Affiliation Parallel Memory Scheme

We propose to construct a single-affiliation scheme for SAMS hierarchically. When $s > q$, SAMS adopts Harper's XOR scheme [4] as it is a single-affiliation one in this case. However, as it is not a single-affiliation scheme when $s \le q$, some modifications must be considered. Figure 2 illustrates one example of how to build SAMS single-affiliation scheme based on Harper's XOR scheme [4] (referred as "basic XOR scheme" in the figure). The construction process is described as follows:

(1) Divide 4 (i.e. 2^q) memory modules into 2 (i.e. 2^{q-s+1}) subgroups, with each group deployed with basic XOR scheme configured with 2 (i.e. 2^{s-1}) modules and stride 4 (i.e. 2^s).

(2) Interleave the two groups, at the granularity of 4 (i.e. 2^s). Now, the module assignment looks like that of the XOR scheme configured with 2 (i.e. 2^{s-1}) modules and stride 8 (i.e. 2^{q+1}).

(3) Combine the two groups and make uniform module index by merging the subgroup module index and the group index.

As Figure 2 suggests, there are cases of four items affiliated with each other (marked with "X"), when the basic XOR scheme is used. The affiliation problem is resolved when we shrink the number of modules in a group, which in turn introduces several (2 in the figure) subgroups of modules. Therefore, steps (2) and (3) in Figure 2 interleave them at the subgroup level and merge them into a unified scheme. After that, we could finally get a single-affiliation scheme, as shown at the bottom of the figure.

In the following, we provide the mathematical description of the above construction process.

- *module assignment function:*

$$m(a) = \begin{cases} a\%2^q, & s = 0 \\ \left\langle a_q \cdots a_s,\ \left(a \otimes T_{H_{s-1,q+1}}\right)\%2^{s-1}\right\rangle, & 1 \le s \le q \\ \left(a \otimes T_{H_{q,s}}\right)\%2^q, & s > q \end{cases}$$

$$(s,\ q \in \mathbb{N})$$

where, a is the n bit linear address, and 2^q is the number of memory modules in the SAMS scheme; s is the *stride family number*, which is the exponential part of the stride family $\{S \| S = \sigma \cdot 2^s,\ \sigma\ odd\}$ to be supported with conflict-free access by the scheme; a_i is the $i-th$ bit of a; $m(a)$ is the module assignment function which has q bits. The notation $x\%y$ means x modulo y, and $< \ldots,\ \cdots >$ denotes binary bits concatenation. $T_{H_{x,y}}$ is the XOR scheme address transformation matrix taken from [4]. $T_{H_{x,y}} = \prod_{k=0}^{\min(x,y)-1} T_{k+\max(x,y),\,k}$, where $T_{i,j}$ is defined to be the identity matrix with a single off-diagonal 1 in $T(i,j)$. The binary matrix T is arranged in a form that the bottom-right element is $T(0,0)$, and the row index grows when moving up and the column index grows when moving left so that the top-left element is $T(l-1,l-1)$ (assume T is $l \times l$ in size). For example,

$$T = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = T_{1,0} \cdot T_{2,1}$$

The $\otimes$ symbol in this paper is used for binary vector-matrix multiplication. For instance, consider

$$a = 7,\ T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

then

$$a \otimes T = [1\ 1\ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = [1\ 1\ 0] = 6\ .$$

The objective of SAMS module assignment function is to make sure that, on one hand, the scheme is conflict-free for stride family $\{S \| S = \sigma \cdot 2^s$, with σ *odd*$\}$; while on the other hand, there are at most two data references going to the same module on a parallel unit-stride access.

3.3 Solving Module Conflicts in Single-Affiliation Schemes

As described in Section 3.1, the construction of a single-affiliation scheme is just the first step of SAMS. To cope with the module conflicts in the single-affiliation scheme, we have to make the modules data lines wider in order to accommodate the conflicting data items. Furthermore, we have to arrange the data properly in the two-dimensional storage, as shown in Figure 3. Note 2^n in the figure is the capacity of the multimodule memory system. Part (1) of the figure shows the linear address distribution in 4 memory modules which satisfies both unit-stride and stride-4 family conflict free access, and part (2) illustrates the linear address distribution which satisfies both unit-stride and stride-8 family conflict free access. We could see from the figure that the idea of SAMS interleaving scheme virtually introduces a third dimension of address flexibility, the offset in the row (data line), besides the module index and row address. The guideline of items placement in module row and offset is simple yet effective: to pack the conflicting items into the same row while maintaining the natural order of the items in the local modules. As there is at most a pair of conflicting items located in the same module on a parallel unit-stride access, a row with width of two is enough for holding them.

The SAMS interleaving scheme consists of three functions: (1) the module assignment function which assigns an item in linear address space to a specific module; (2) the row assignment function which determines the row in which the item is placed; and (3) the offset assignment function which calculates the offset of the item in the row. Since we have presented the module assignment function in Section 3.2, we introduce the row assignment and the offset assignment functions below:

- *row assignment function:*

$$r(a) = \begin{cases} \frac{a}{2^{q+1}}, & s = 0 \\ \frac{a}{2^{q+1}}, & 1 \le s \le q \\ \left(\left(\frac{a}{2^q} + 1\right) \% 2^{n-q}\right)/2, & s > q \end{cases}$$

$$(s,\, q \in \mathbb{N})$$

- *offset assignment function:*

$$o(a) = \begin{cases} a_q, & s = 0 \\ a_{s-1}, & 1 \le s \le q \\ \overline{a_q}, & s > q \end{cases}$$

$$(s,\, q \in \mathbb{N})$$

The notations x/y and $\frac{x}{y}$ mean the quotient of integer division between x and y. Note that n is the number of bits of the linear address of the 2^q memory modules. $r(a)$ has $n-q-1$ bits, while $o(a)$ is a single bit, because we consider only two pieces of data per data line in the SAMS scheme.

4. HARDWARE DESIGN AND EXPERIMENTAL EVALUATION

Above, we have presented the formula of the SAMS scheme. For any parallel memory scheme to be practically useful, it is important to have efficient hardware implementation as the scheme logic is in the critical path of every memory access. In this section, we will examine the hardware implementation issues of the proposed scheme.

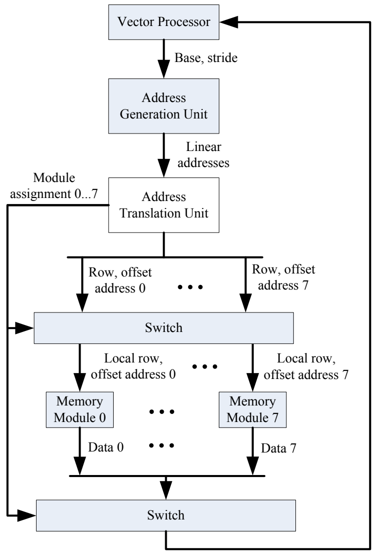

Figure 4: Parallel memory system based on SAMS

Figure 4 illustrates the organization of parallel memory system based on our SAMS scheme. The vector processor core issues memory access commands (base address and stride) to the Address Generation Unit(AGU), where the 2^q(in Figure 4 $2^q = 8$) linear addresses are calculated in parallel and then sent downto the SAMS memory system. The eight linear addresses are resolved by the Address Translation Unit(ATU) into eight module assignments, eight row addresses and eight row offset addresses. After that, the eight groups of row-offset pair and eight data items from input data port (on a memory write) go to the address & data switch and get routed to the proper memory modules according to their corresponding module assignments. In case of a read memory access, after one clock cycle the eight read data are fed back to the vector processor via the data switch at the bottom of Figure 4. We will focus on the hardware implementation of ATU, as it is the core of the parallel memory scheme and the overall components of Figure 4 are more or less independent of the parallel memory scheme itself.

As we have described in Section 3, the Address Translation Unit maps the linear addresses $a_i (0 \le i \le 2^q - 1)$ to the module-row-offset triples $m(a_i)$, $r(a_i)$ and $o(a_i)$. Figure 5 illustrates the logic implementation of the address translation process. Note, in Figure 5, the bits selections without the involvement of s are simply *static* wire selections. The latter which are completely fixed as n and q are fully determined by hardware (such as $a[n-1:q]$ in Figure 5 (b)), whereas those with involvement of s result multiplexors (such as a_{s-1} in Figure 5 (c)). And the comparisons in $r(a)$ and $o(a)$ logic(i.e. $s \le q$, $s \ne 0$ and $1 \le s \le q$) could be done and stored a priori, therefore they are not in the critical path. Consequently, the critical path of the row assignment logic is an $n-q$ bit CLA followed by a 2-to-1 multiplexer, and that of the offset assignment is an $(n-q)$-to-1(incurred by a_{s-1}, note $0 \le s \le n-q$) multiplexer and a 2-to-1 multiplexer. For the module address assignment function $m(a)$, we have analyzed that after collapsing and merging all the multiplexors in Figure 5 (c), we could get the simplified hardware implementation with q independent $(n-q+1)$-to-1 multi-

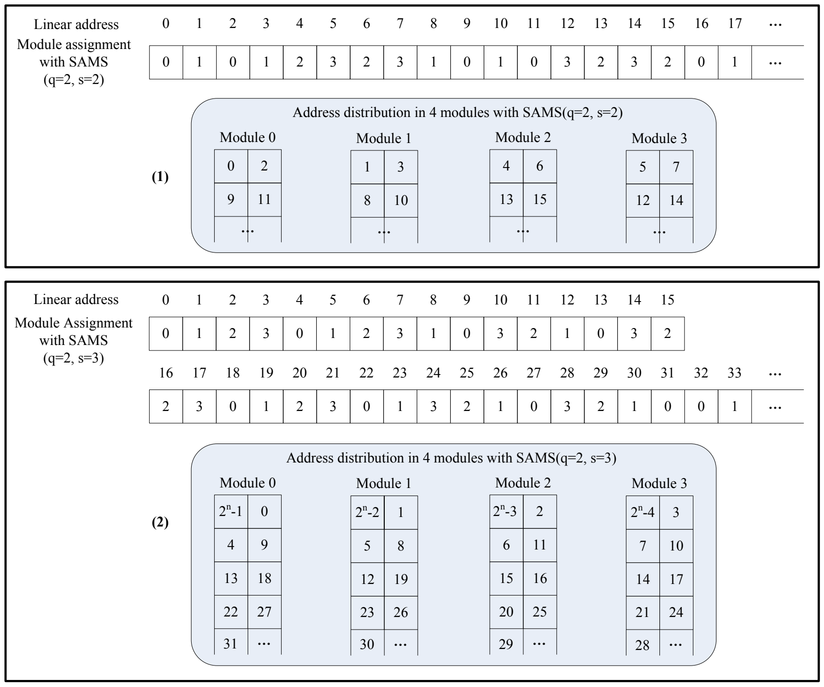

Figure 3: SAMS data arrangement example with 4 modules

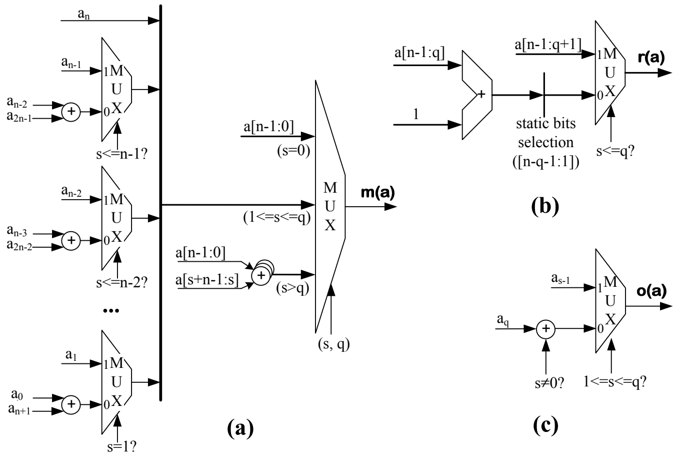

Figure 5: SAMS address translation logic: (a)Module assignment, (b)Row assignment, (c)Offset assignment

plexors fed by 2-input XOR gates. Hence, the critical path of the module assignment function is a 2-input XOR gate followed by a $(n-q+1)$-to-1 multiplexer. Notice that the module, row and offset assignment functions work independently, therefore the critical path of ATU is the longest one among the three, which is the $n-q$ bit Carry Look-ahead Adder followed by a 2-to-1 multiplexer in the row assignment function.

We have implemented the ATU in Verilog and synthesized it using Xilinx ISE 9.1i. The target FPGA device is Virtex2-Pro XC2VP30-7FG676. Table 1 summarizes the performance results of our design in terms of delay and hardware utilization. The experiment is done under different configurations with various module capacities (denoted by n) and number of modules (denoted by q). For example, $n = 23$ means the address space (i.e. the capacity) of the multimodule memory system is $8M$ (2^{23}) words and $q = 3$ means there are 8 modules. We could see that the SAMS address translation logic has low critical path delay, which is in the proximity of 4 ns. In addition, the FPGA logic resources consumption is trivial - less than 1%. It is also shown in the table that the critical path delay and resource consumption scale well with the capacity of the parallel memory and the number of memory modules.

5. RELATED WORKS

To cope with module conflicts of vector accesses across stride families, several techniques have been proposed in the literature, including the use of buffers [3], dynamic memory schemes [5, 4], memory modules clustering [3] and intra-stream out-of-order access [22]. Under the fully-parallel multimodule memory model, these methods still work but are subject to some limitations.

The use of buffers [3] is probably most straightforward solution as it tolerates the module conflicts by simply buffering the input addresses and output data and collecting the required data after some delay. The buffer depth required depends on misalignment between the parallel memory scheme used and the vector access stride. If the access stream is distributed evenly between the memory modules of the system, then the peak throughput of one data per memory module in one cycle might be achieved after a transient startup time. However, since the startup disparity is unavoidable, this solution introduces significant time penalties in case of short access streams. Moreover, the use of buffers and the logic for collecting the correct data items from the buffers could cause substantial hardware overheads.

The dynamic scheme proposed in [5, 4] works well only when the same data set is accessed with single stride family. However, if the data set is to be accessed using different stride families, the penalty of flushing and reloading data between the memory modules and the lower level in the memory hierarchy may not be amortized in some cases, which would result in performance degradation of the system.

The memory modules clustering [3] introduces inefficient use of module control logic and data routing resources, as a large portion of memory modules may remain idle during each parallel memory access. For instance, under the assumption that the number of modules is a power of two number, the amount of memory modules used for conflict-free access of two unmatched stride families may be no more than 50% of the available modules. This results in waste of logic resources and power in some cases.

The out-of-order vector access [22] is based on the observation that a long, strided memory reference stream with module conflicts in sequential order could become conflict-free, if properly reordered. For instance, in a multimodule system with conflict-free stride-4 access support, a stride-2 stream with 16 memory references could be accomplished by two stride-4 streams with 8 memory references each. Basically the original stride-2 stream is split into two stride-4 sub-streams and the memory system is accessed with by the alternating sub-streams. In this case, the access is conflict-free. The problem with intra-vector out-of-order access is that it requires long vectors for proper operation. In addition, as data items are read out of order, data permutation logic may introduce additional penalties [6].

The most distinctive aspect of our scheme compared to the previous solutions is that it avoids the module conflicts when the memory reference patterns go back and forth between unit-stride and strided accesses, and thus truly-parallel data access is supported. Unlike the out-of-order vector access scheme, our proposal preserves the data sequence required by the vector load/store units, thus atomic parallel access is achieved for short vectors and peak performance could be sustained for vectors as short as 2^q elements. The SAMS is a memory scheme with no redundancy and high utilization of module resources. On the other hand, the SAMS scheme is complementary to the existing techniques, which means that it could also take their advantages to improve system performance. For instance, it adopts the idea of the dynamic scheme where s can be configured by the software at run time for different stride family access in different execution phases. For long vectors, it could also be augmented with out-of-order intra-vector access scheme to support conflict-free access for a wider spectrum of stride families.

6. DISCUSSIONS

The improvement of this work compared to existing multimodule memory scheme techniques rests upon the use of two-dimentional memory modules as building blocks of our parallel memory system. This allows us to overcome the limitation of nonredundant parallel memory schemes, and to exploit the data line access parallelism, which is the key to tolerate the module conflicts raised by the unit-stride access. It should be noted that although we choose Harper's XOR scheme as our starting point to construct a single-affiliation multi-stride parallel memory scheme, the idea of SAMS could be generally applied to other interleaving systems, such as the row-rotation scheme [5]. The key point is to apply the two steps introduced in Section 3.1, just as described in Sections 3.2 and 3.3 for the XOR scheme.

It should be noted that SAMS is just one of the set of parallel memory schemes, which provide conflict-free access support for cross stride family vector accesses, under the configuration of 2D memory modules with wide data lines. It has not yet been proved that SAMS scheme is optimal in terms of the number of strides supported, and the complexity of hardware implementation. Therefore, it could be worthwhile to explore the design space of memory schemes

[6]Data permutation is not required in the original proposal [22] as there the assumed memory organization is that single datum is read out from the multiple memory modules per cycle, whereas in the organization considered in this paper multiple data items (equal to the number of memory modules) are read per cycle.

Table 1: Delay and hardware usage of ATU

Configuration		Delay (ns)			Hardware Used	
n	q	Logic Delay	Wire Delay	Total	Slices	LUTs
8	3	1.43	1.01	2.44	18	26
16	3	2.54	0.93	3.47	47	71
25	3	2.86	0.95	3.81	75	119
27	3	2.28	1.75	4.03	76	125
32	3	2.83	1.40	4.23	96	148
64	3	4.28	0.95	5.33	180	292
23	3	2.79	0.95	3.74	67	108
23	4	2.75	0.98	3.73	78	126
23	5	2.71	0.99	3.70	88	140

under the configuration of wide memory modules, to hopefully find a scheme with better performance compared to SAMS.

Even inside the SAMS scheme itself, there is still space for performance improvement. As presented in the paper, we have elaborated on the module assignment function; however we chose the row assignment and offset assignment functions straightforwardly. With the module assignment function fixed, there are no means to enlarge the number of supported conflict-free strides as the module conflict patterns are fixed with a fixed module assignment function. We also found that the row assignment function does not affect neither the module conflicts nor the hardware implementation of the SAMS scheme. However, the offset assignment function, which determines the relative positions of the data items in the same row, do have some impact on the hardware implementation. Namely, the offset assignment function determines the permutation patterns of accessed data which should be supported by the data routing circuitry of the SAMS scheme. Therefore, investigation for other offset assignment functions could also be helpful for better hardware implementation.

Regarding the application, the SAMS scheme could be applicable wherever the data level parallelism is exploited, to boost the performance of data intensive applications with both unit-stride and strided memory accesses. For instance, it could be adopted as on-chip local store for SIMD processors, such as the SPUs in the Cell processor [9], to improve the flexibility of memory access. It could also be considered for integration as on-chip buffer for GPP SIMD extensions, where the data alignment and permutation problem, which results from the lack of flexible memory access support, remains a bottleneck for many applications [15, 17]. It should be noted that the integration of a SIMD buffer into a GPP introduces coherence problem, as it will be deployed at the same level as the data caches in memory hierarchy. However, this problem could be solved by either snooping mechanisms or by the use of non-cacheable regions in the address space.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed the SAMS scheme, which improves the previous memory interleaving schemes by providing additional support for conflict-free unit-stride access, which is the most common access pattern in vectorized scientific and engineering applications. In the SAMS approach, we created a hierarchical way of composing single-affiliation memory scheme from given multi-affiliation scheme. Furthermore, we added a new dimension of data access parallelism by representing each memory module as a 2D storage to resolve the unit-stride access conflicts in the single-affiliation scheme. In this way, SAMS provides atomic access time for unit stride access, while it could preserve all benefits of the existing cross-stride-family parallel memory schemes. Experimental results indicate that the address translation of the SAMS scheme has efficient hardware implementation which makes it a promising technique to be used for low latency on-chip parallel memory buffers.

The SAMS scheme could be extended to provide conflict-free access for even more stride families or to be used in two-dimensional strided access environment, by deploying a wider data line with more than two data items. However, it should be noted that a trade-off exists between the hardware complexity and the performance benefits due to the wide data lines. The SAMS scheme has the same drawback as proposals with redundant memory modules, as far as the data routing circuitry is concerned. Therefore, if the implementation of an ordinary scheme needs a $2^q \times 2^q$ crossbar, then a $(k \cdot 2^q) \times 2^q$ crossbar can be required in the SAMS scheme. We are investigating the correlation of the data routing patterns of different modules, and other candidates of offset assignment functions and their impact on data routing, in order to reduce the complexity. In addition, we are also working on the fusion of the SAMS scheme with other existing techniques, such as out-of-order vector access, to achieve better support for conflict-free access for wider range of stride families.

8. ACKNOWLEDGMENTS

This research has been supported by the European Comission in the context of the FP6 project SARC (FET-27648) and by the Dutch Technology Foundation STW, applied science division of NWO and the Technology Program of the Dutch Ministry of Economic Affairs (project DCS.7533).

9. REFERENCES

[1] D. Baily. Vector computer memory bank contention. *IEEE Trans. Computers*, 36:293–298, Mar. 1987.

[2] P. Budnik and D. Kuck. The organization and use of parallel memories. *IEEE Trans. Computers*, C(20):1566–1569, Dec. 1971.

[3] D. T. Harper III. Block, multistride vector and FFT accesses in parallel memory systems. *IEEE Trans. Parallel and Distributed Systems*, 2(1):43–51, 1991.

[4] D. T. Harper III. Increased memory performance during vector accesses through the use of linear

address transformations. *IEEE Trans. Computers*, 41(2):227–230, 1992.

[5] D. T. Harper III and D. A. Linebarger. Conflict-free vector access using a dynamic storage scheme. *IEEE Trans. Computers*, 40(3):276–283, 1991.

[6] R. Espasa, M. Valero, and J. E. Smith. Vector architectures: past, present and future. In *Proceedings of the 12th international conference on Supercomputing*, pages 425–432, 1987.

[7] S. Hammond, R. Loft, and P. Tannenbaum. Architecture and application: The performance of the NEC SX-4 on the NCAR benchmark suite. In *Proceedings of the ACM/IEEE Conference On Supercomputing 1996*, pages 22–22, 1996.

[8] W.-C. Hsu and J. Smith. Performance of cached DRAM organizations in vector supercomputers. *ACM SIGARCH Computer Architecture News*, 21:327–336, May 1993.

[9] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the Cell multiprocessor. *IBM J. Res. & Dev.*, 49(4/5):589–604, 2005.

[10] C. Keltcher, K. McGrath, A. Ahmed, and P. Conway. The AMD Opteron processor for multiprocessor servers. *IEEE Micro*, pages 66–76, Mar. 2003.

[11] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded SPARC processor. *IEEE Micro*, pages 21–29, Mar. 2005.

[12] L. Kontothanassis, R. Sugumar, G. Faanes, J. Smith, and M. Scott. Cache performance in vector supercomputers. In *Proceedings of the ACM/IEEE Conference On Supercomputing 1994*, pages 255–264, 1994.

[13] C. Kozyrakis. *Scalable Vector Media-Processors for Embedded Systems*. PhD Thesis, UC Berkeley, Berkeley, CA, USA, May 2002.

[14] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden. IBM POWER 6 microarchitecture. *IBM J. Res. & Dev.*, 51(6):639–662, 2007.

[15] D. Nuzman, I. Rosen, and A. Zaks. Auto-vectorization of interleaved data for SIMD. In *PLDI '06: Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation*, pages 132–143, New York, NY, USA, 2006. ACM.

[16] A. Pajuelo, A. Gonzalez, and M. Valero. Speculative dynamic vectorization. In *Proceedings of the 29th Ann. Int'l Symp. Computer Architecture*, pages 271–280, 2002.

[17] G. Ren, P. Wu, and D. Padua. Optimizing data permutations for SIMD devices. *SIGPLAN Not.*, 41(6):118–131, 2006.

[18] R. M. Russell. The CRAY-1 computer system. *Communications of the ACM*, pages 63–72, Jan. 1978.

[19] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner. POWER 5 system microarchitecture. *IBM J. Res. & Dev.*, 49(4/5):505–521, 2005.

[20] T. Sun and Q. Yang. A comparative analysis of cache designs for vector processing. *IEEE Trans. Computers*, 48:331–344, Mar. 1999.

[21] J. M. Tendler, J. S. Dodson, J. S. Fields, H. L. Jr., and B. Sinharoy. POWER 4 system microarchitecture. *IBM J. Res. & Dev.*, 46(1):5–25, Jan. 2002.

[22] M. Valero, T. Lang, M. Peiron, and E. Ayguade. Conflict-free access for streams in multimodule memories. *IEEE Trans. Computers*, 44:634–646, 1995.