Interleaved parallel schemes : improving memory throughput on supercomputers

André Seznec, Jacques Lenfant

400 ex.

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1992

25ème anniversaire

N° 1656

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

INTERLEAVED PARALLEL SCHEMES : IMPROVING MEMORY THROUGHPUT ON SUPERCOMPUTERS

André SEZNEC
Jacques LENFANT

Avril 1992

★RR-1656★

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Interleaved Parallel Schemes: improving memory throughput on supercomputers *

André Seznec, Jacques Lenfant
Programme 1
Projet CALCPAR

20 mars 1992
Publication Interne n° 646 - 14 pages

IPS, un schéma un rangement améliorant le débit mémoire d'un supercalculateur vectoriel

Résumé

Sur certains supercalculateurs, plusieurs processeurs vectoriels partagent une mémoire très fortement entrelacée en mode MIMD. Quand tous les processeurs travaillent sur une même boucle vectorielle simple, une partie importante du débit potentiel de cette mémoire peut être perdue à cause de l'asynchronisme des processseurs.

Un mode de synchronisation SIMD des processeurs lors des accès aux vecteurs en mémoire peut être utilisé pour limiter cette perte. Mais une partie importante du débit mémoire peut être gachée lors des accès à des vecteurs rangés avec une raison paire.

Dans cet article, nous présentons un schèma de rangement des données dans une mémoire entrelacée: IPS. IPS assure une distribution équitable des éléments d'un vecteur sue une mémoire fortement entrelacée pour un large spectre de vecteurs. Nous montrons de plus comment organiser les accès à la mèmoire pour que le routage des vecteurs vers les processeurs requièrent un minimum de passes à travers le réseau d'interconnexion.

Abstract

On many commercial supercomputers, several vector register processors share a global highly interleaved memory in a MIMD mode. When all the processors are working on a single vector loop, a significant part of the potential memory throughput may be wasted due to the asynchronism of the processors.

In order to limit this loss of memory throughput, a SIMD synchronization mode for vector accesses to memory may be used. But an important part of the memory bandwidth may be wasted when accessing vectors with an even stride.

In this paper, we present IPS, a interleaved parallel scheme, which ensures an equitable distribution of elements on a highly interleaved memory for a wide range a vector strides. We show how to organize access to memory, such that unscrambling of vectors from memory to the vector register processors requires a minimum number of passes through the interconnection network.

*à paraitre dans les Proceedings of the 19th International Symposium on Computer Architecture (IEEE-ACM), Mai 1992

Interleaved Parallel Schemes: improving memory throughput on supercomputers*

André Seznec, Jacques Lenfant
IRISA, Campus de Beaulieu
35042 Rennes Cedex, FRANCE
e-mail : seznec@irisa.fr

Abstract

On many commercial supercomputers, several vector register processors share a global highly interleaved memory in a MIMD mode. When all the processors are working on a single vector loop, a significant part of the potential memory throughput may be wasted due to the asynchronism of the processors.

In order to limit this loss of memory throughput, a SIMD synchronization mode for vector accesses to memory may be used. But an important part of the memory bandwidth may be wasted when accessing vectors with an even stride.

In this paper, we present IPS, a interleaved parallel scheme, which ensures an equitable distribution of elements on a highly interleaved memory for a wide range a vector strides. We show how to organize access to memory, such that unscrambling of vectors from memory to the vector register processors requires a minimum number of passes through the interconnection network.

Keywords

Vector register processor, SIMD synchronization, Interleaved Parallel Scheme, Interconnection Network

*à paraitre dans les Proceedings of the 19th International Symposium on Computer Architecture (IEEE-ACM), Mai 1992

1 Introduction

As vector parallelism is easiest to detect and to exploit, a large number of vector processors have been built and commercialized.

Vector processors usually resort to the pipeline architecture (e.g., vector register machines as the Cray Series) or to the SIMD architecture where a single sequencing unit controls several identical processing elements (e.g. the Connection Machine).

On many commercial supercomputers (e.g., Cray series, Alliant series, ..), several vector register processors share a global highly interleaved memory in a MIMD mode. Even when all the processors are working on a single vector loop, a significant part of the potential memory througput may be wasted due to asynchrony of the processors. In order to limit this loss of memory throughput, one should synchronize the processors in a SIMD fashion.

In this paper, we consider a SIMD synchronization mode that could be implemented on a shared memory multiprocessor built with multiple vector register processors accessing vectors in memory.

When working in this mode, both SIMD and vector register machine performance on a specific vector instruction sequence will depend highly on the effective throughput of memory, i.e., on an adequate distribution of the vector elements over the memory banks and also on avoiding conflicts in the interconnection network during the unscrambling of the vector.

In section 2, we recall classical difficulties for accessing vectors in parallel on SIMD computers and vector register machines. In section 3, we point out that it may be easier to obtain maximum memory throughput for vector accesses on a shared memory multiprocessor built with vector register processors than on SIMD

computers built with scalar registers or vector register uniprocessors.

In section 4, the Interleaved Parallel Scheme, IPS(d,q,n), is proposed. Using this new mapping induces more equitable distribution over memory banks for a wider set of vectors than when using the usual mappings.

However, equitable distribution on memory banks is not sufficient to guarantee parallel accessibility.

In section 5, we show that the parallel accessibility of vectors in memory is ensured by this storage scheme **and** SIMD synchronization of the processors. We show how to organize the unscrambling of data from memory to the processors avoiding conflicts in the interconnection network and obtaining effective high memory throughput on vector accesses for a very large range of access strides.

2 Accessing memory on SIMD machines and vector register processors

Many manufacturers of vector processors have adopted the following definition for a vector.

Definition: A vector is an ordered set of words whose addresses form an arithmetic series.

For example, rows, columns and diagonals of a matrix stored columnwise are vectors.

2.1 Mapping vectors onto a parallel memory

In both SIMD machines and vector register processors, several memory banks are used to build a large memory system with high potential thoughput. To obtain maximum memory throughput for a vector instruction, memory access conflicts must be avoided.

The usual mapping of data onto memory consists of mapping address A in memory bank A *mod* N at local address A / N: this mapping is sometimes referred to as low order interleaving. When using this mapping, the distribution of the elements of a vector V stored with a stride R has the following property:

> $V(i)$ and $V(j)$ are stored in the same memory bank iff
> $i = j \bmod N/GCD(N,R)$

Then $N/GCD(N,R)$ consecutive elements of the vector are stored in distinct memory banks and then may be accessed in parallel.

To ensure conflict free parallel accesses to a maximum set of vectors, one may use a prime number of memory banks. This was used in the Burroughs Scientific Processor [KUC82]. Then address computation requires arithmetic modulo a prime number, however for simplicity in construction one generally prefers to use a power of two memory banks. Then by simply reordering the address lines, bank interleaved addressing is accomplished.

From now on, we shall assume that the number of memory banks is a power of two: $N = 2^n$.

As the ability to access a slice of a vector in parallel depends on the distribution of data in memory banks, we define here the equitable distribution of a set of data among the memory banks:

Definition: The distribution of a set S of $X * N$ elements on the N memory banks is equitable iff X elements of S are stored in each memory bank.

2.2 Vector register machines: time-multiplexed memory

In vector register machines, an interleaved memory is used because of the difference between processor cycle time and memory cycle time.

Let us consider a memory consisting in N memory banks. A memory bank is busy due to a request for t cycles. For simplicity, we consider a vector register machine with only one memory access port (as e.g. Cray 1 or Cray 2). In order to guarantee that no conflicts may arise between successive vector accesses, a delay of t cycles must occur between two successive requests to the same bank.

2.3 SIMD machines : space-multiplexed memory

Let us consider a SIMD processor with N memory banks and P processing elements (processors).

The structure is illustrated in Figure 1. Memory

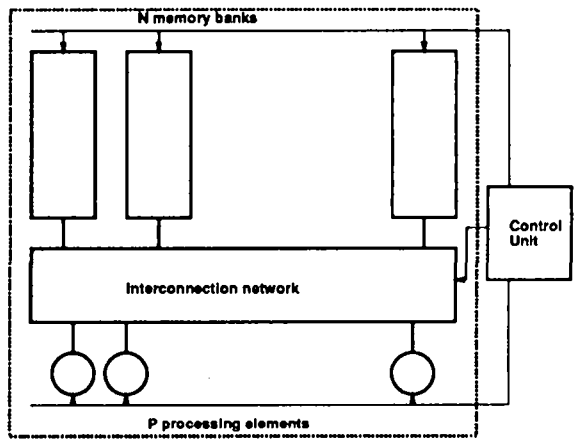

Figure 1: Simplified structure of a SIMD computer

banks are numbered from 0 to $N-1$ and processors are numbered from 0 to $P-1$.

Memory is accessed by the processors through the interconnection network.

For simplicity's sake, we also assume that $P = N = 2^n$ processors. If the vector stride is $R = 2^k r$ with r odd and $k \leq n$, then 2^k cycles are needed for accessing a slice of N consecutive elements of a vector.

In [TAM85], it is pointed out that the distribution of the strides of accessed vectors in real applications is not random, but that about 80 % of the vectors are accessed with stride one, while the remaining 20% are accessed with approximatively random stride.

Let us consider the following distribution of vector accesses:

- 80 % of the vectors are accessed with stride one
- 10 % of the vectors are accessed with an odd stride distinct from one
- for $k \geq 1$, $10/2^k$ % of the accessed vectors have strides of the form $2^k r$

Then accessing 100 vectors of N elements will require at the average $90 + (n+1) * 10$ cycles: e.g., for $N = 512$, i.e., $n = 9$, (resp. $N = 64$, i.e., $n = 6$) the effective memory throughput is only of 53% (resp. 62.5 %) of the potential throughput.

As a key point, note that one of the major obstacles to the systematic use of the type COMPLEX in FORTRAN applications is the two-strided access it induces for accessing the vector of real (resp. imaginary) parts, resulting in half of the memory bandwidth being wasted.

In order to remove this penalty for accesses to vectors with an even stride, memory bandwidth is sometimes overdimensioned i.e., $N = 2P$ or $N = 4P$.

2.4 Using skewing schemes

In order to increase effective throughput, one can try to use skewing schemes [BUD71]. The following skewing scheme was proposed by Harper and Jump [HAR87] :

> let $m(A)$ be the number of the memory bank where a word at address A is stored. m(A) is given by:
>
> $$m(A) = (A + (A/N)) \ mod \ N$$

Figure 2 illustrates this skewing scheme.

This skewing scheme was proposed for improving the throughput of the interleaved memory of a vector register machine.

Using this skewing scheme on a vector register machine associated with buffers at input and output of each memory bank may dramatically improve the effective throughput of an interleaved memory on a vector register machine (see [HAR87]).

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
15	8	9	10	11	12	13	14
22	23	16	17	18	19	20	21
29	30	31	24	25	26	27	28
36	37	38	39	32	33	34	35
43	44	45	46	47	40	41	42
50	51	52	53	54	55	48	49
57	58	59	60	61	62	63	56

Figure 2: Skewing scheme of Harper and Jump ($N = 8$)

For SIMD computers, this skewing scheme allows parallel access to some vectors stored with even stride, but it does not allow parallel access to slices of N consecutive elements for most of the one-strided vectors, e.g. the vector starting at address 1 (see Figure 2).

3 Accessing vectors on shared memory supercomputers: a new challenge

3.1 Synchronizing processors in a SIMD fashion

On many commercial supercomputers (e.g., Cray Research Systems, ...), several vector register processors share a global memory in a MIMD mode.

The memory organization in these machines may be illustrated by figure 3: this memory is both space and time multiplexed. Several physical memory banks are connected to the same port of the interconnection network; we will refer to this set of memory banks as a logical memory bank.

In such a memory organization, the clasical mapping of data is:

> Word at address A is mapped to local address $A/2^{n+d}$ in physical bank $(A/2^n)$ *mod* 2^d in logical bank A *mod* 2^n.

Achieving high performance on vector applications on these machines depends highly on the effective throughput of the memory for vector accesses.

Even when all the processors are working on a single vector loop, a significant part of the potential memory throughput may be wasted due to asynchronism of the processors: access to vector V0 by processor P0 may interfere with access to vector V1 by processor P1 when strides and/or base addresses of the vectors are different.

In order to limit this loss of memory throughput, one may imagine synchronizing the sequencing of the distinct processors in a SIMD fashion, or at least to implement a specific synchronization mode for which the

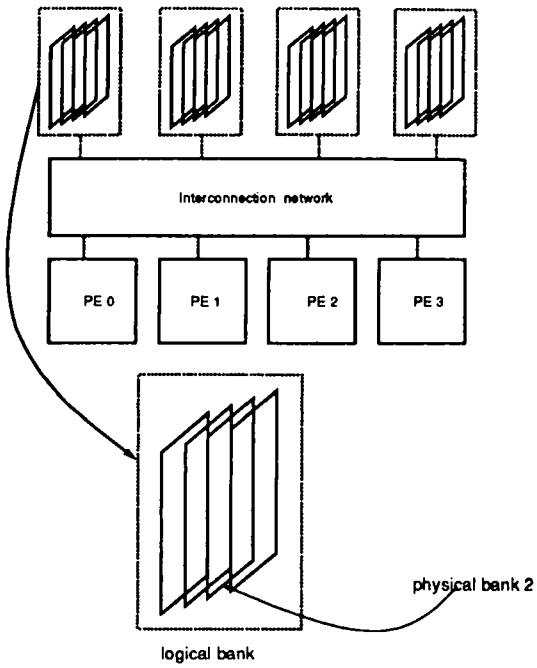

Figure 3: Memory organization on a vector register processor based multiprocessor

accesses to memory of the distincts processors are synchronized in a SIMD fashion. There is no systematic increase of the memory throughput for all the vector sections [1], but simulations show that on average there would be an increase of memory throughput.

Considering the same distribution of the vector strides as in section 2.3, for 64 (resp. 8) processors, 64 (resp. 8) logical memory banks and 8 physical banks per logical banks busy for 8 cycles by a memory request, only 38 % (resp. 41 %) of the theoretical memory throughput would be realized by a a quasi-infinite sequence of vector accesses without any synchronization of the processors versus 2/3 (resp. 74 %) of the theoretical memory throughput when using SIMD synchronization on the vector requests to memory.

It must be noted that these simulations are rather optimistic for 64 processors: in our simulation, a 64*64 crossbar network was assumed and we supposed that conflicts were resolved on the fly (i.e., a rejected request is submitted again at the next cycle). Implementation of such a large fast crossbar network and its associated control logic cannot be envisaged at the moment.

[1] e.g. consider the usual mapping of data in memory and a vector sequence accessing two two-strided vectors one with an even base address, the second with an odd base address

Distribution of the elements

When synchronizing the processors in a SIMD fashion, the distribution of the operations among the processors is naturally done as follows: element x of a vector is processed by processor $x \ mod \ 2^n$.

In the next sections, we show that the effective memory throughput on vector accesses may be improved by using a skewed storage scheme which will allow a maximum memory throughput on a wider family of vectors than allowed by the usual mapping.

3.2 Some notations

From now, we consider a multiprocessor where such a SIMD synchronization mode is implemented with the following parameters:

1. Each processor is a vector register machine: it may receive one word of data from memory per cycle.
2. The size of the vector register is $M = 2^m$ words.
3. The number of processors is $N = 2^n$.
4. The number of logical memory banks is also $N = 2^n$
5. The processors are connected to the memory banks through a N entry, N exit interconnection network.
6. A logical bank is built with $D = 2^d$ physical memory banks. We shall consider that a physical bank can deliver a word of data per D cycles.

Definition: A vector V is q-permissible if its storage stride is of the form $2^k r$ with $k \leq q$ and r odd.

3.3 The new challenge

In this architecture, the granularity of a vector instruction on the machine becomes $2^n * 2^m$ elements. To obtain the maximum throughput on vector accesses to the memory, the challenge has been changed:

> *slices of M*N consecutive elements of a vector have to be accessed from the memory and unscrambled in a minimum time to the vector registers of the distinct processors.*

Let us point out that the elements of the vectors which are accessed in parallel do not have to belong to the same slice of 2^n consecutive elements.

In the next section, we present the Interleaved Parallel Scheme IPS(d,q,n), $q \leq n$. When using this scheme, any slice of 2^{q+d+n} consecutive elements of any q-permissible vector is equitably distributed among the 2^{d+n} physical memory banks. Moreover in section 5, we show how to organize the access to a slice of $2^n * 2^m$ consecutive elements of a q-permissible vector in order to busy the distinct resources (memory banks, interconnection network, ..) during only 2^m cycles.

4 The Interleaved Parallel Scheme IPS(d,q,n)

4.1 The Logical Parallel Scheme

Let $0 < q \leq n$, let us consider the decomposition of the binary representation of an address A in four bit strings A_3, A_2, A_1 and A_0 respectively of $s-(n+q)$ bits, q bits, $n-q$ bits and q bits (see Figure4): $A = A_3 2^{s-(n+q)} + A_2 2^n + A_1 2^{n-q} + A_0$. We shall note also $A = (A_3, A_2, A_1, A_0)$.

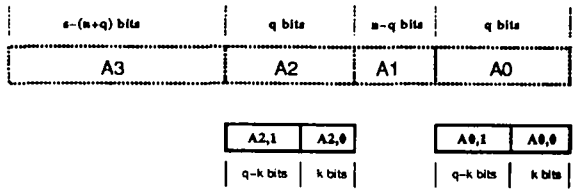

Figure 4: Decomposition of an address A into different bit strings

We define the Logical Parallel Scheme LPS(q,n) as follows:

Address A is mapped at local address (A_3, A_2) in logical bank $(A_1, A_2 \oplus A_0)$.

Other skewing schemes using exclusive-OR were considered in other contexts in [RAU89, NOR87, FRA85]. From now on, we assume that the LPS(q,n) skewing scheme is used for mapping data on to the memory. In order to achieve maximum throughput on a vector access, an equitable distribution over the logical memory banks is needed.

Theorem 1: Let V be a q-permissible vector, then for any arbitrary slice of 2^{n+q} consecutive elements of vector V, the distribution of the elements on the 2^n logical memory banks is equitable, i.e., 2^q per logical memory bank.

Proof:
Let V be a vector stored with stride $2^k r$ and base address F.
Let T be the slice of the first 2^{n+q-k} consecutive elements of the vector V,
let f be the application defined by :

$$f: \{0,1,..,2^{n+q}-1\} \rightarrow \{0,1,..,2^s-1\}$$
$$i \rightarrow L = \{0,1,..,2^s-1\}$$

$f(i)$ is the address of element $V(i)$
Let us consider the previous binary decomposition of an address A: A_2 (resp. A_0) may be subdivided in two substrings $A_{2,1}$ and $A_{2,0}$ (resp. $A_{0,1}$ and $A_{0,0}$) of $q-k$ bits and k bits respectively.

Let us consider the applications P_1, P_2 and P_3 defined by :

$$P_1: L \rightarrow \{0,1,..,2^{n+q}-1\}$$
$$A \rightarrow (A_{2,1}, A_{2,0}, A_1, A_{0,1})$$
$$P_2: L \rightarrow \{0,..,2^k-1\}$$
$$A \rightarrow A_{0,0}$$
$$P_3: L \rightarrow \{0,1,..,2^{n+q}-1\}$$
$$A \rightarrow (A_{2,1}, A_1, A_{0,1} \oplus A_{2,1}, A_{0,0} \oplus A_{2,0})$$

A second form of $P_1 of$ is : $P_1 of(i) = r * i + (F \ / \ 2^k)$;
A second form of $P_2 of$ is : $P_2 of(i) = F \ mod \ 2^k$;

Then $P_1 of$ is a bijection and $P_2 of$ is a constant application, this induces that $P_3 of$ is a bijection.

As $(A_1, A_{0,1} \oplus A_{2,1}, A_{0,0} \oplus A_{2,0})$ is the index of the logical memory bank where the word at address A is stored, 2^{q-k} elements of the slice T are stored in each memory bank. Q.E.D.

4.2 Physically Parallel Scheme

A logical memory bank is built with $D = 2^d$ physical memory banks; an equitable distribution of the elements in a vector slice among the logical memory banks is not sufficient to guarantee a maximum throughput for the memory on vector accesses. We must also guarantee an equitable distribution of the data among the physical memory banks. The usual low order interleaving at the logical memory bank level does not work as shown by Figure5: the physical memory banks may be partitionned into "odd" banks and "even" banks.

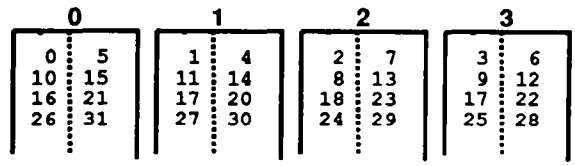

Figure 5: Low order interleaving of physical memory banks using $LPS(1,2)$

That is why, we also need to skew the storage inside each logical memory bank using the Physically Parallel Scheme PPS(d,q,n) described below:

The word at address A is mapped to physical bank number

$p_{d,q,n} = (A_3 \ mod \ 2^{min(d,q)} \oplus A_2 \ mod \ 2^d)$

We called the combination of the two schemes on physical and logical banks the Interleaved Parallel Scheme IPS(d,q,n)

IPS(1,2,2) is illustrated in Figure6.

Theorem 2: Let V be a q-permissible vector, then for any arbitrary slice T of 2^{n+q+d} consecutive elements of

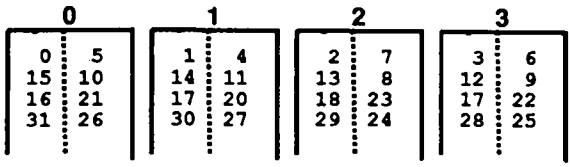

Figure 6: The IPS(1,2,2) skewing scheme

vector V, the distribution of the elements on the 2^{n+d} physical memory banks is equitable, i.e., 2^{q+d} per physical memory bank.

Proof: We give the proof for the case $d \leq q$. The proof for case $q < d$ is quite similar.

Let V be a vector stored with stride $2^k r$ and base address F.
Let T be the slice of the first 2^{n+q-k} consecutive elements of the vector V,
let f be a mapping from $S_k = \{0, 1, .., 2^{n+q} - 1\}$ in the space of addresses $L = \{0, 1, .., 2^s - 1\}$, where $f(i)$ is the address of element i in slice T,
and f is defined by $f(i) = 2^k r * i + F$.

Let us consider mappings P_3 and P_4 defined by

$P_3 : L \rightarrow \{0, .., 2^d - 1\}$
$A \rightarrow A_3 \ mod \ 2^d$
$P_4 : L \rightarrow \{0, .., 2^{q+n} - 1$
$A \rightarrow (A_2, A_1, A_0)$

For $0 \leq i < 2^{n+q}$, the restriction of $P_4 of$ to the set $\{i + 2^{n+q-k}x\}_{0 \leq x < 2^d}$ is a constant mapping :

$P_4 of(i + 2^{n+q-k}x) = ((2^{n+q}x * r + f(i)) \ mod \ 2^{n+q} = f(i) \ mod \ 2^{n+q}$.

Then, for $0 \leq i < 2^{n+q}$, the elements $T(i + 2^{n+q-k}x)_{0 \leq x < 2^d}$ are stored in the same logical memory bank

For $0 \leq i < 2^{n+q}$, the restriction of $P_3 of$ to the set $\{i + 2^{n+q-k}x\}_{0 \leq x < 2^d}$ is a bijection: $P_3 of(i + 2^{n+q-k}x) = [(f(i) + rx2^{n+q})/2^{n+q}] \ mod \ 2^d = (rx + Y_i) \ mod \ 2^d$

As element x of the slice T is stored in physical memory bank number $(P_4 of(x)) \oplus ((P_3 of(x)/2^n)$, then for $0 \leq i < 2^{n+q} - 1$, the elements $T(i + 2^{n+q-k}x)_{0 \leq x < 2^d}$ are stored in the different physical memory banks of a single logical memory bank.

Thus the elements of the slice T of vector V which are stored in logical memory bank I are equitably distributed among the physical memory banks. As we already know (theorem 1) that the slice T is equitably distributed among the logical memory banks, the slice T is equitably distributed among the 2^{n+d} physical memory banks. Q.E.D.

5 Parallel accessibility of data

In the previous section, we have shown that the Interleaved Parallel Scheme ensures an equitable distribution of data among the physical and logical memory banks for all the q-permissible vectors.

Paradoxally, when there is no SIMD synchronization of the processors, simulations show that, when using IPS(d,q,n) the effective memory throughput on a sequence on vector accesses is approximatively the same as the throughput obtained when using the usual mapping: all the benefits of a better data distribution are lost due to conflicts in the interconnection network, physical banks, ... *This explains why skewed storage schemes are not used on current vector register processor based multiprocessor.*

In order to feed the processors at a rate of one data element per cycle and per processor, we need to unscramble the elements from the memory to the processors through the interconnection network at a maximum rate, i.e., 2^n elements per cycle.

When looking at Figure 6, it is quite clear that slices of 2^n consecutive elements cannot be unscrambled by a permutation network in a single pass: there are conflicts in destinations. But as previously pointed out, the challenge is not to unscramble 2^n consecutive vector elements in a single pass through a 2^n entry, 2^n exit interconnection network, but to unscramble 2^{n+m} consecutive elements in 2^m passes through the interconnection network.

In this section, we shall show that SIMD synchronization of processors and the use the Interleaved Parallel Scheme IPS(d,q,n) in conjunction with our earlier results on controlling interconnection networks controlling interconnection networks for vector accesses [LEN85, SEZ87] allow us to unscramble any slice of 2^{n*m} consecutive elements of a q-permissible vector in a minimum number of passes through a 2^n entry, 2^n exit interconnection network.

5.1 Previous results on vector unscrambling through interconnection networks

Controlling an interconnection network for performing a permutation may require a quite complex hardware mechanisms. The use of a crossbar network (as in the BSP [KUC82]) seems efficient as a first approach. These networks can perform all the permutations between their P entries and their N exits. But a crossbar network requires $N * P$ switches; fast crossbar networks for large N and P are not be built because they cannot be easily partitioned: a crossbar network cannot be envisaged in vector register processor based multiprocessor with more than 15 or 20 processors.

In order to overcome this difficulty, multistage interconnection networks (like the Omega network [LAW75] or the Benes network [BEN68]) have been proposed: these interconnection networks can be partitionned and pipelined. The Benes network $B^{(n)}$ can perform any arbitrary permutation on its 2^n entries. It requires $(2n-1)$ stages consisting in 2^{n-1} $2*2$ switches and can be pipelined. But the algorithms to control this network for arbitrary permutations are time and space consuming [LEE85]: these algorithms cannot be used at execution time in vector applications because the permutation to be executed may change at each cycle. Pre-computing control of the interconnection network at code generation time may seem attractive; but the needed permutations must be known at compile time, this is generally incompatible with the use of separate compilation and library routines.

To overcome this difficulty, Lenfant [LEN85] defined FUM's (frequently used permutations). FUM's are a set of permutations on $E^{(n)} = \{0, 1, .., 2^n-1\}$ depending on a few parameters that are sufficient to execute most of the useful data manipulations on vectors when the usual mapping of data is used. Lenfant also presented efficient algorithms to compute the routing of Benes network for this family.

5.1.1 Frequently used permutations

In this section, we consider that the usual mapping of data on memory banks is used.

The family $\{\lambda^{(n)}_{J,K}\}$:

For executing the assignment $B := A$, when A and B are two vectors both stored with an odd increment, the interconnection network has to perform a permutation of the family $\{\lambda^{(n)}_{J,K}\}$ on slices of 2^n elements of A. For j odd, $\lambda^{(n)}_{J,K}$ is defined by $\lambda^{(n)}_{J,K}(x) = Jx + K \ mod\ 2^n$.

The Bit Permute Complement permutations (BPC): Permutations of the family $\{\lambda^{(n)}_{J,K}\}$ are not sufficient for accessing vectors stored with an even increment.

Let us consider a vector A stored with a stride $2^k r$ where r odd and $k \leq n$ and a vector B stored with an odd increment.

Assigning a slice of 2^{n-k} consecutive elements of a vector A in vector B may be decomposed in three steps:

1. Access to the 2^n first elements of the vector A' derived from vector A as follows: $A'(0)$ and $A(0)$ have the same address, A' is stored with increment r.
2. Alignment of the elements of vector A on the 2^{n-k} first positions
3. Realignment of the result vector in order to store the 2^{n-k} first elements in vector B.

The three steps induce permutations through the interconnection network: steps 1 and 3 induce permutations of the family $\{\lambda^{(n)}_{J,K}\}$ and step 2 may be realized by $\sigma^k_{(n)}$ where $\sigma_{(n)}$ is the perfect shuffle on the segment $E^{(n)} = \{0, 1, .., 2^n - 1\}$.

The perfect shuffle, its powers, the bit-reversal and some other useful permutations belong to the class of Bit Permute Complement permutations (BPCs) [NAS81]:

Definition: A permutation M on $E^{(n)}$ is a BP (bit permute permutation) if there exists a permutation T on $\{1, 2, .., n\}$ such that, given $S = T^{-1}$ and $x_n x_{n-1}..x_1$ the binary representation of x, $x_{S(n)}x_{S(n-1)}..x_{S(1)}$. A permutation Ψ is BPC on $E^{(n)}$ if there exists a BP M, an integer x in $E^{(n)}$ such that $\Psi = \tau^{(n)}_x oM$, $\tau^{(n)}_x$ is the exclusive or by x.

The permutations of the family $\{\lambda^{(n)}_{J,K} o\tau^{(n)}_x o\ M\ o\lambda^{(n)}_{i,h}\}$ allow us to execute the assignment B:=A for all vectors in a minimum number of passes through the interconnection network.

Permutations of this family are referred to as EBPC's (Extended Bit Permute Complement permutations) [SEZ86, SEZ87].

In [LEN85], Lenfant gave algorithms with time complexity in $O(n)$ for controlling the 2^n entry, 2^n exit Benes network for the family $\{\lambda^{(n)}_{J,K}\}$ and the BPC's.

Seznec [SEZ86, SEZ87] proposed the Sigma network $\Sigma^{(n)}$ (Figure7) which is essentially the Benes preceeded and followed by a link realizing the bit-reverse permutation i.e, the BP induced by a symmetry of bits. Algorithms with time complexity $O(n)$ for controlling the 2^n entries, 2^n exits Sigma network $\Sigma^{(n)}$ for the EBPC's were also presented.

When implementing these algorithms in hardware, control of the Sigma network may be computed at execution time for the EBPC's (see [SEZ86]). [2]

5.2 Unscrambling q-permissible vectors

In this section, we consider that IPS(d,q,n) is used for memory.

The following result shows that synchronizing the processors in a SIMD fashion will allow us to unscramble all q-permissible vectors in a minimum number of passes through an interconnection network.

Theorem 3 : Let V be a q-permissible vector, then any arbitrary slice of 2^{n+q} consecutive elements of the vector V may be routed through 2^n entry, and 2^n exit

[2] In order to be able to route all the vectors from memory to the processors, permutations of family $\{\tau^{(n)}_x o\ \sigma^k_{(n)}\ o\lambda^{(n)}_{i,h}\}$ are sufficient i.e.,. $2^n * n * 2^{n-1} * 2^n$ permutations e.g., for N=64, 786432 permutations, for N =512, more than $600 * 10^6$ permutations!

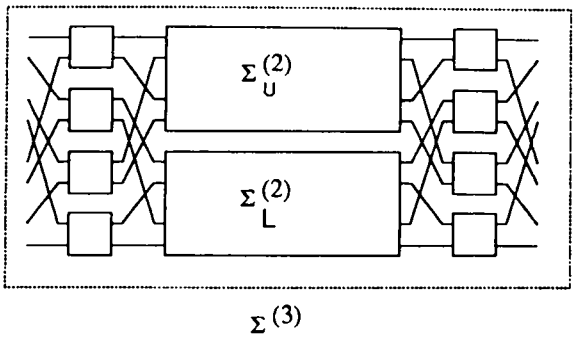

Figure 7: Recursive definition of the Sigma network$\Sigma(3)$

interconnection network in 2^q passes (i.e.,., the minimum possible number) using only EBPC's.

Proof: Let V be a vector stored with stride $2^k r$ and base address F.
Let T be the slice of the first 2^{n+q-k} consecutive elements of the vector V,
let f be the mapping from $S_k = \{0, 1, .., 2^{n+q} - 1\}$ in the space of addresses $L = \{0, 1, .., 2^s - 1\}$ defined by $f(i) = 2^k r * i + F$. $f(i)$ is the address of element i in slice T.

Let us consider applications P_6 and P_7 defined by :

$$\begin{array}{lll} P_6: & L & \rightarrow \{0, .., 2^n - 1\} \\ & A & \rightarrow (A_{2,0}, A_1, A_{0,1}) = (A/2^k) \ mod \ 2^n \\ P_7: & L & \rightarrow \{0, .., 2^{q-k} - 1\} \\ & A & \rightarrow A_{2,1} = (A/2^{n+k}) \ mod \ 2^{q-k} \end{array}$$

For $0 \leq i < 2^n$, for $0 \leq x < 2^{q-k}$, $P_6 of(i + x2^n) = \lambda^{(n)}_{r,(F/2^k)}(i)$

For $0 \leq i < 2^n$, for $0 \leq x < 2^{q-k}$, $P_7 of(i + x2^n) = [[(2^k ri + F) + 2^{k+n} rx]/2^{k+n}] \ mod \ 2^{q-k}$
i.e., $P_7 of(i + x2^n) = (rx + C_i) \ mod \ 2^{q-k}$

Then for $0 \leq i < 2^n$, the restriction of $P_7 of$ to the elements accessed by Processing Element i (i.e., elements $i + x2^n$, $0 \leq x < 2^{q-k}$) is a bijection)

Then, for $0 \leq i < 2^n$, for $0 \leq t < 2^{q-k}$, there exists $0 \leq X(i,t) < 2^{q-k}$ such that $P_7 of(i + 2^n X(i,t)) = t$

Then, for $0 \leq t < 2^{q-k}$, the element $V(X(i,t))$ is stored in the logical memory bank $(2^k t + (F \ mod \ 2^k)) \oplus (\sigma^k_{(n)} o \lambda^{(n)}_{r,(F/2^k)}(i))$

Then the set of elements $(V(X(i,t)))_{0 \leq i < 2^n}$ may be routed in a single pass through the interconnection network using a EBPC i.e., : $(\tau^{(n)}_{2^k t \oplus (F \ mod \ 2^k)} o \sigma^k_{(n)} o \lambda^{(n)}_{r,(F/2^k)})^{-1}$.
Q.E.D.

This result shows how to organize subslices of 2^n non-consecutive elements of the vectors in order to access these subslices in parallel and to route the elements to their destination processors.

The set of permutations induced on the interconnection network by IPS(d,q,n) is the set induced by the usual low order interleaving scheme.

5.3 Using the IPS(d,q,n) skewing scheme

In this section, we suggest how to access a slice of $2^m * 2^n$ elements of a vector V stored with stride $2^k r$, r odd, $k \leq q$ and starting at base address F when using the proposed skewing scheme.

We assume that $m > d$, i.e., that a vector load hits several times each physical bank (when the data distribution is equitable for the vector).

We assume that the IPS(d,q,n) skewing scheme is used with parameter $q = min(n, m - d)$.

For the sake of simplicity, we assume here that $q = m - d$.

Accessing a slice T of 2^{m+n} consecutive elements of vector V may be executed in three steps as follows:

1. Address computation:
   For each processor, addresses for all the accessed elements in the slice are computed:
   in order to allow parallel access to the memory by all the processing elements, sub-slices of 2^n elements (one for each processor) are organized.
   In each processor i, the addresses of the vector elements are stored in a special address vector register Va as follows:
   the address B of element $T(i + 2^n x)$ is stored in Va at local address :
   $a_{T,i}(x) = (A_2 \oplus x \ mod \ 2^k, p_{d,q,n}(A))$
   The results from previous sections ensure that $a_{T,i}$ is a one-to-one mapping from $\{0, 1, .., 2^m - 1\}$ onto itself.
2. Access to memory:
   The addresses are sent to memory in usual order (i.e., increasing indices in the vector registers):
   Addresses of same rank $y = y1 * 2^{k+d} + x * 2^d + y0$ in the vector registers Va of the distinct processors are sent at the same cycle to memory; there is no conflict with the logical memory banks (theorem 1) and the permutation used for sending this set of addresses is: $\tau^{(n)}_{2^k y1 \oplus (F \ mod \ 2^k)} o \sigma^k_{(n)} o \lambda^{(n)}_{r,(F/2^k)}$ (theorem 3).
   Moreover there is no conflict with the physical banks: physical bank x in logical bank i receives one request in the subslice at rank y iff $y = x \ mod \ 2^d$.
3. Data are read from memory and routed back to the processors. In each processor, the data are reordered in the destination vector register by by $a^{-1}_{T,i}$.

These three steps may be pipelined, but step 2 cannot overlap the end of the execution of step 1.

Nevertheless, accessing two vectors may be chained; no inter-access delay is needed between two consecutive vector accesses: a throughput of one element per cycle per processor may be obtained in a sequence of accesses to q-permissible vectors.

If the number of processors is quite small (less or equal to 16), then using a crossbar may be envisaged. Otherwise another interconnection network must be used, for example the Sigma network.

Note that if a Sigma network Σ^n is used as the interconnection network, the performance demanded of the control unit of the interconnection is to compute the command for the permutation $\sigma^k_{(n)} o \lambda^{(n)}_{r,(F/2^k)}$ during step 1; the command for $\tau^{(n)}_{2^k y 1 \oplus (F \bmod 2^k)} o \sigma^k_{(n)} o \lambda^{(n)}_{r,(F/2^k)}$ may then be deduced on the fly by very simple hardware mechanism (see [SEZ87]).

An example

Let us consider, a vector register processor based multiprocessor with $N = 64$ processors, 64 logical memory banks (i.e., $n = 6$) consisting of 8 physical banks (i.e., $d = 3$), a vector register size of 64 elements (i.e., $m = 6$).

The IPS(3,3,6) skewing scheme may be used; then access to slices of $64 * 64 = 4096$ consecutive elements of any vectors stored with strides of the form r, $2r$, $4r$ or $8r$ with r odd will be executed at full speed.

The number of cycles needed to access a slice of 4096 consecutive elements of a vector stored with a stride $2^k r$,r odd is:

- $k \leq 3$: 64 cycles
- $3 < k \leq 12$: $2^{k-3} * 64$ cycles
- $k \geq 12$: $512 * 64$ cycles

If we consider the same distribution of vector accesses as in section 2.1, then the average number of cycles needed for accessing 100 vector slices of 4096 elements is : $64 * (98.75 + 9 * \Sigma_{3<k\leq 12} 10 * 2^{k-3}/2^k + 10 * 512/2^{12}) =$ $64 * 111.25$ cycles, i.e., about 90 % of the maximum achievable throughput compared with the 53% achieved when using low order interleaving.

For 8 processors, 8 logical banks and 8 physical banks per logical bank, the average memory throughput on vector accesses would be of 93 % of the maximum throughput.

6 Conclusion

High performance vector processors generally are generally organized as vector register processors or SIMD machines. For these architectures, a highly interleaved structure of memory is used; the performance may be limited by conflicts in memory accesses and by routing in the interconnection network.

In this paper, we have considered a shared memory multiprocessor based on vector registers processors. In order to feed the processing elements with data, the memory must be interleaved in a two-way fashion: space multiplexing (2^n data must be accessed in parallel) and time multiplexing (memory cycle time is longer than processor cycle time). When the distinct processors are working on the same vector sequence, synchronizing processors' accesses to vectors in memory limits the contention on the interconnection network and may be an alternative to the use of overdimensioned memory bandwidth in current commercial supercomputers.

But using the usual low order interleaving scheme on memory and a power of two as number of memory banks would lead to unsatisfactory memory throughput on vectors stored with an even stride.

Using SIMD synchronization of vector register processors changes the challenge of accessing 2^n consecutive vector elements in parallel in a SIMD computer to the challenge of accessing a slice of $2^n * 2^m$ consecutive vector elements in 2^m cycles.

The IPS(d,q,n) skewing scheme allows access in a minimum time to a wide range of vectors (all vectors stored with stride of form $2^k r$ with $0 \leq k \leq q$):

- Slices of 2^{n+m} consecutive vector elements are equitably distributed over the memory banks.
- Data may be routed in a minimum number of passes through an interconnection network: the permutations induced on the interconnection network are EBPC's. Realistic control algorithms for executing the EBPC's on the Sigma network were given in [SEZ87].

We have also noted is that simulations have shown that using the Interleaved Parallel Scheme does not degrade the memory throughput when the distinct processors execute independent vector accesses streams in MIMD mode on the memory.

When using SIMD synchronization of vector register processors and the IPS(d,q,n) skewing scheme on memory, the granularity of a vector access to memory is a slice of 2^{d+q+n} elements, i.e., access time to a slice of x consecutive elements of a vector, $x \leq 2^{d+q+n}$ is approximatively the same as for the complete slice of 2^{d+q+n} elements; but when usual low order interleaving is used, the granularity of a vector access to memory is a slice of 2^{d+n} elements and delays between consecutive vector accesses must be inserted. Moreover there exist vector applications with a very high granularity: the commercial success of the Connection Machine (64 K processors) proves it.

Acknowledgement

The authors are endebted to the program committee member G.Q. Macquire for carefully polishing the english in the final version of the paper.

References

[BEN68] V.E. Benes, "Mathematical Theory of connecting networks and telephone traffic", New York: Academic, 1968

[BUD71] P.Budnick, D.Kuck "The organization and use of parallel memories" IEEE Transaction On Computers, Dec. 1971

[FRA85] J.M. Frailong, W.Jalby, J.Lenfant "XOR-schemes: a flexible organization in parallel memories" Proceedings of 1985 International Conference on Parallel Processing, Aug. 1985

[KUC82] D.J.Kuck, R.A.Stokes, "The Burroughs Scientific Processor (BSP)" IEEE Transactions on Computers, May 1982.

[HAR86] D.T. Harper, J.R. Jump "Performance evaluation of vector accesses in parallel memories using a skewed storage scheme ", Proceedings of the 13^{th} International Symposium on Computer Architecture, June 1986

[HAR87] D.T. Harper, J.R. Jump "Vector accesses in parallel memories using a skewed storage scheme " IEEE Transactions on Computers, Dec. 1987

[LAW75] D.H. Lawrie "Access and alignment of data in array computer" IEEE Transactions on Computers, Dec. 1975

[LEE85] K.Y. Lee, "On the rearrangeability of a (2 log N - 1) stage permutation network" IEEE Transactions on Computers, May 1985.

[LEN78] J.Lenfant, "Parallel permutations of data : A Benes network control algorithm for frequently used permutations" IEEE Transactions on Computers, July 1978.

[LEN85] J.Lenfant, " A versatile mechanism to move data in an array processor" IEEE Transactions on Computers, June 1985

[NAS81] D.Nassimi, S.Sahni "A self-routing Benes network and permutation algorithms" IEEE Transactions on Computers, May 1981.

[NOR87] A.Norton, E.Melton "A class of boolean linear transformations for conflict-free power-of-two stride access", Proceedings of the International Conference on Parallel Processing, 1987

[RAU89] B.Rau, M.Schlander, D. Yen " The Cydra 5 stride insensitive memory system", Proceedings of the International Conference on Parallel Processing, 1989

[SEZ86] A.Seznec, "An efficent routing control unit for the Sigma network $\Sigma^{(4)}$", Proceedings of the 13^{th} International Symposium on Computer Architecture, June 1986

[SEZ87] A.Seznec, "A new interconnection network for SIMD computers: the Sigma network $\Sigma^{(n)}$" IEEE Transactions on Computers, July 1987

[TAM85] H.Tamura, Y.Shinkai, F.Isobe "The Supercomputer FACOM VP system" Fujitsu Sc. Tech. J., March 1985

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

PI 635 A NOTE ON CHERNIKOVA'S ALGORITHM
Hervé LE VERGE
Février 1992, 28 pages.

PI 636 ENSEIGNER LA TYPOGRAPHIE NUMERIQUE
Jacques ANDRE, Roger D. HERSCH
Février 1992, 26 pages.

PI 637 TRADE-OFFS BETWEEN SHARED VIRTUAL MEMORY AND MESSAGE-PASSING ON AN iPSC/2 HYPERCUBE
Thierry PRIOL, Zakaria LAHJOMRI
Février 1992, 26 pages.

PI 638 RUPTURES ET CONTINUITES DANS UN CHANGEMENT DE SYSTEME TECHNIQUE
Alan MARSHALL
Mars 1992, 510 pages.

PI 639 EFFICIENT LINEAR SYSTOLIC ARRAY FOR THE KNAPSACK PROBLEM
Rumen ANDONOV, Patrice QUINTON
Mars 1992, 20 pages.

PI 640 TOWARDS THE RECONSTRUCTION OF POSET
Dieter KRATSCH, Jean-Xavier RAMPON
Mars 1992, 22 pages.

PI 641 MADMACS : A TOOL FOR THE LAYOUT OF REGULAR ARRAYS
Eric GAUTRIN, Laurent PERRAUDEAU
Mars 1992, 12 pages.

PI 642 ARCHE : UN LANGAGE PARALLELE A OBJETS FORTEMENT TYPES
Marc BENVENISTE, Valérie ISSARNY
Mars 1992, 132 pages.

PI 643 CARTESIAN AND STATISTICAL APPROACHES OF THE SATISFIABILITY PROBLEM
Israël-César LERMAN
Mars 1992, 58 pages.

PI 644 PRIME MEMORY SYSTEMS DO NOT REQUIRE EUCLIDEAN DIVISION BY A PRIME NUMBER
André SEZNEC, Yvon JEGOU, Jacques LENFANT
Mars 1992, 10 pages.

PI 645 SKEWED-ASSOCIATIVE CACHES
André SEZNEC, François BODIN
Mars 1992, 20 pages.

PI 646 INTERLEAVED PARALLEL SCHEMES : IMPROVING MEMORY THROUGHTPUT ON SUPERCOMPUTERS
André SEZNEC, Jacques LENFANT
Mars 1992, 14 pages.

PI 647 COMMUNICATING PROCESSES AND FAULT TOLERANCE : A SHARED MEMORY MULTIPROCESSOR EXPERIENCE
Michel BANATRE, Maurice JEGADO, Philippe JOUBERT, Christine MORIN
Mars 1992, 40 pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399