

# **Towards Memory Centric Computing: A Flexible Address Mapping Scheme**

Authors: Jan van Lunteren

IBM Zurich Research Laboratory  
Säumerstrasse 4  
CH-8803 Rüschlikon, Switzerland  
[jvl@zurich.ibm.com](mailto:jvl@zurich.ibm.com)

Published in: Proceedings of the IEEE Canadian Conference on Electrical and  
Computer Engineering CCECE'99, vol. 1, pp. 385-390, Edmonton,  
Alberta, May 1999.

© 1999 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

# Towards Memory Centric Computing: A Flexible Address Mapping Scheme

J. van Lunteren  
IBM Research Division  
Zurich Research Laboratory  
8803 Rüschlikon, Switzerland  
jvl@zurich.ibm.com

## Abstract

*Main memory increasingly affects computer system performance due to the exponentially growing performance gap between microprocessor and DRAM technology. A novel address mapping scheme is presented that, compared to conventional schemes, allows a much more efficient use of main memory with respect to performance as well as flexible support of different memory configurations. The mapping scheme can be used to build a high-performance memory that can adapt its operation to the way it is being used.*

## 1 Introduction

The exponentially growing performance gap between microprocessor and DRAM technology combined with the increasing memory need of programs make main memory within computer systems a resource that must be treated with care [1].

Current techniques for improving main memory performance are based on the application of multiple independent memory banks in parallel and the exploitation of fast output buffers that are available within EDO and SDRAM technologies. The effective performance gain achieved by these techniques is dependent on how well an address mapping function distributes memory accesses over the various memory banks, and how well it concentrates memory accesses that are mapped on the same memory bank within single rows of the DRAM cell arrays.

General-purpose computer systems typically apply a static mapping function that is based on sequential interleaving. This has worked reasonably well until now due to the simple implementation of this scheme for a power-of-2 number of memory banks and due to the good match with the sequential nature of instruction fetches that are an important part of the total

access traffic to main memory. However, the growing performance gap between microprocessor and DRAM technology might now have reached a level at which a mapping scheme that allows dynamic adaptation to specific memory reference behavior of individual workloads can achieve significant performance gains over a static mapping scheme that only takes some general access characteristics into account. At the same time, it is desirable to have a mapping scheme that supports more flexible memory configurations. Sequential interleaving in its basic form only supports a power-of-2 number of equally sized memory banks, and requires special logic to support more flexible memory configurations, resulting in a nonuniform mapping throughout the address space with corresponding performance variations.

This paper presents a mapping scheme that fulfills these goals. The mapping scheme is introduced in Section 2. Section 3 discusses several mapping properties and features that are supported by the scheme. Section 4 describes the support of various memory configurations. Section 5 reasons how the mapping scheme can be used to build a main memory that can adapt to patterns that occur in the access traffic. The final Section 6 summarizes the paper. Throughout this paper, the main focus is on the distribution of memory accesses over the memory banks. However, the presented method can also be used to concentrate successive memory accesses to a single bank within single rows of the DRAM cell array in order to exploit the fast output modes of EDO and SDRAM technologies. This will not be discussed here.

## 2 Adjustable Address Mapping

The address mapping scheme presented is based on a small lookup table [2]. The concept is shown in figure 1. The mapping scheme involves the selection

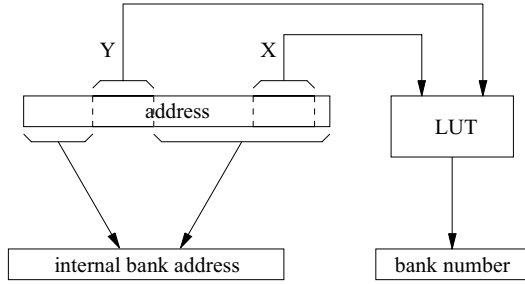


Figure 1: Address mapping based on lookup table.

of two portions X and Y within an address, which are used to index a lookup table for obtaining a bank number. An internal bank address is obtained by removing the Y field from the address.

The X and Y fields do not need to cover adjacent bit positions within the address. It is possible that various bits that constitute the X and Y fields are interleaved. It is not allowed that the same bit is part of both the X as well as the Y field. The size of the Y field,  $y$ , is determined by the number of memory banks,  $M$ , according to

$$y = \lceil \log M \rceil.$$

In a typical implementation, the X and the Y field have fixed locations. However, it is also possible to make the selection of the X and Y fields configurable. The bank numbers contained in the lookup table can be binary-coded, requiring  $\log M$  bits per entry for  $M$  memory banks. Alternatively a bit vector can be used in which each bit position corresponds to exactly one memory bank. In this case  $M$  bits are required per entry. To achieve a valid address mapping, each row in the lookup table that corresponds to one X value should contain each bank number exactly once. Dependent upon the dynamics with which the mapping must be updated, the lookup table can be realized in PROM, flash memory or SRAM technologies.

### 3 Mapping Properties and Features

This section will show how the size and positions of the X and Y fields and the contents of the lookup table determine the type and the properties of the mappings that are supported.

For reasons of simplicity is assumed that most of the X and Y fields cover adjacent bit positions, where the least significant bit of the X field is located at position  $p$  and the least significant bit of the Y field is located at position  $q$ . The notion of an address

sequence with a fixed increment, i.e., a stride, will be used. For example, an address sequence

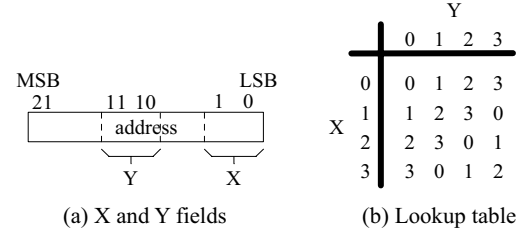
$$a, a + s, a + 2s, \dots, a + ks$$

is said to have a stride of  $s$ .

#### 3.1 Interleaving of power-of-2 strides

Figure 2 illustrates a basic mapping applied on a memory consisting of four memory banks. Figure 2(a) shows the positions of the X and Y fields (with  $p = 0$  and  $q = 10$ ), figure 2(b) shows the contents of the lookup table, and figure 2(c) shows the resulting mapping in which at each memory location the address is written that is mapped on it.

The columns and rows of the lookup table determine the properties of the mapping. A column corresponding to a constant Y value determines how an address sequence with successive X values is mapped over the memory banks. Such an address sequence has a stride of  $2^p$ . In figure 2 an address sequence from 0 to  $2^{10} - 1$  with a stride of  $2^0 = 1$  will result in an Y value equal to 0. According to the first column of the lookup table shown in figure 2(b) this stride-1 address sequence is interleaved over all four memory banks, which is illustrated in figure 2(c), e.g., the address sequence 0, 1, 2, 3, 4. Owing to the mapping mechanism, these addresses are mapped on successive



		0	1	2	3
FFFFh		3FF7FFh	3FFBFFh	3FFFFh	3FF3FFh
400h		1000h	1400h	1800h	1C00h
internal bank address	4	4	404h	804h	C04h
	3	403h	803h	C03h	3
	2	802h	C02h	2	402h
	1	C01h	1	401h	801h
	0	0	400h	800h	C00h
		0	1	2	3

(c) Mapping

Figure 2: Interleaving of two power-of-2 strides.

internal bank addresses. Similarly, each row in the lookup table, corresponding to a constant X value, determines how an address sequence with successive Y values is mapped over the memory banks. Such an address sequence has a stride of  $2^q$ . In figure 2 an address sequence 0, 400h, 800h, C00h, 1000h, with a stride of 400h results in an X value equal to 0. According to the first row of the lookup table, this address sequence is interleaved over all four memory banks. Owing to the mapping mechanism, each group of four successive addresses in this sequence is mapped on the same internal bank address.

This example illustrates the unique feature of the mapping scheme, namely its ability to simultaneously interleave address sequences with two different power-of-2 strides over all memory banks. Section 3.2 will show that it is also possible to simultaneously interleave more than two power-of-2 strides.

The total size of the lookup table equals  $4 \times 4 \times 2$  bits, which is only 4 bytes.

### 3.2 Multiple Parallel Mappings

By extending the X field with some bits, it is possible to create different segments within the address space and apply different mappings on these segments.

An example is shown in figure 3. In figure 3(a) one additional bit is added to the X field as the most significant bit, denoted X1, which is located at bit position 6 within the address. The segments within the address space for which X1 equals 0, e.g., the address ranges from 0 to 3Fh, from 80h to BFh, and so on, are mapped according to the upper part of the lookup table (X between 0 and 3) as shown in figure 3(b). The segments for which X1 equals 1, e.g., the address ranges from 40h to 7Fh, from C0h to FFh, and so on, are mapped according to the lower part of the lookup table (X between 4 and 7).

The two mappings that are simultaneously applied in figure 3 map a stride-1 address sequence with an interleave factor of 4 (e.g., the address sequence 0, 1, 2, 3), respectively, with an interleave factor of 2 (e.g., the address sequence C40h, C41h, C42h, C43h).

By extending the X field with more bits, additional different segments can be created within the address space that can be mapped differently. The location of these segments within the address space is determined by the bit positions of these additional X field bits, e.g., bit X1 in figure 3(a).

Section 3.1 showed that two power-of-2 strides can be interleaved over all memory banks. Figure 3 shows that even more power-of-2 strides can be interleaved simultaneously. Besides strides  $2^0 = 1$  and  $2^{10} = 400h$ ,

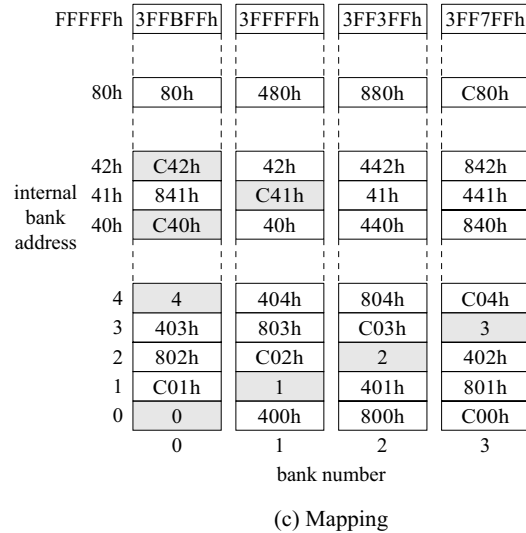
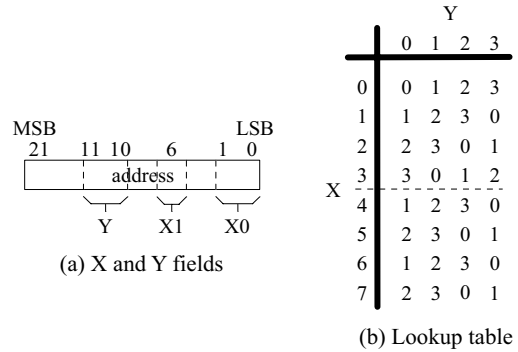


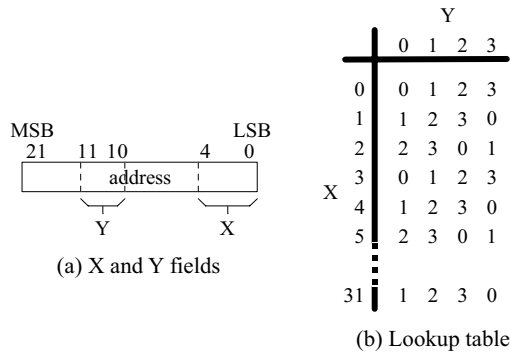
Figure 3: Two parallel mappings.

the power-of-2 stride  $2^6 = 40h$  is also two-way interleaved (e.g., the address sequence 0, 40h, 80h). This is realized by coordinating the two mappings as shown in figure 3(b) such that within the same column different bank numbers are contained for X values that are only different in the most significant bit X1 (e.g., compare the bank numbers corresponding to X values 0 and 4 in each of the four columns).

In this way it is possible to simultaneously interleave even more than three power-of-2 strides over two or more memory banks as desired, through a larger extension of the X field. Of course this will require a larger lookup table.

### 3.3 Complex Mapping Emulation

A second application of an extended X field is to emulate complex mapping schemes, such as prime degree interleaving [3], skewed-storage, and pseudo-random interleaving [4].



internal bank address	0	1	2	3
FFFFh	3FFFFh	3FF3FFh	3FF7FFh	3FFBFFh
20h	20h	420h	820h	C20h
1Fh	C1Fh	1Fh	41Fh	81Fh
1Eh	1Eh	41Eh	81Eh	C1Eh
5	805h	C05h	5	405h
4	C04h	4	404h	804h
3	3	403h	803h	C03h
2	802h	C02h	2	402h
1	C01h	1	401h	801h
0	0	400h	800h	C00h

(c) Mapping

Figure 4: Emulation of prime degree interleaving.

Figure 4 shows an example in which the X field is extended to 5 bits. The lookup table is configured such that a prime degree interleaving scheme is emulated for stride-1 address sequences. For example, the address sequence 0, 1, 2, 3, ..., 1Fh is 3-way interleaved as is shown in figure 4(c). This 3-way interleaving usually requires expensive logic for modulo calculation. The 3-way interleaving is *emulated*. After each block of 32 successive addresses, the mapping repeats itself as shown in figure 4(c). By extending the X field, the interleaving scheme can be emulated more accurately for longer address sequences. The total size of the lookup table in this example equals  $4 \times 32 \times 2$  bits, which is 32 bytes.

## 4 Memory Configurations

Efficient use of available memory resources requires that memory configurations can be composed of any number of memory banks having a variety of bank sizes.

The most popular mapping scheme used today in general-purpose computers is based on sequential interleaving in which the least significant bits of the address are used as bank number and the remaining bits form the internal bank address. The basic version of this mapping scheme requires a power-of-2 number of equally sized memory banks. Memory configurations with a non-power-of-2 number of unequally sized memory banks are usually supported by dividing the storage into partitions covering equally sized parts of a power-of-2 number of memory banks and applying sequential interleaving on each of these partitions. For example, a memory configuration consisting of 7 memory banks is divided into three partitions consisting of 4 memory banks, 2 memory banks, and 1 memory bank. The partitioning is often implemented using expensive subtract and compare logic, which is custom designed for a given memory system. A second disadvantage is that parts of the address space are mapped differently with varying interleave factors, resulting in non-uniform memory performance throughout the address space.

The presented mapping scheme overcomes most of these disadvantages. It supports any number of memory banks in an integral way. A limited variety of bank sizes is supported. To support a non-power-of-2 number of non-equally-sized memory banks it is required, however, that the Y field cover the most significant address bits. Figure 5 shows the support of a non-

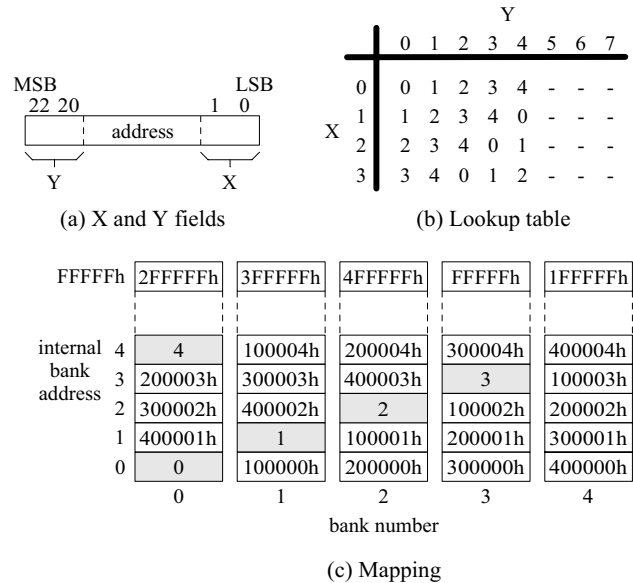


Figure 5: Support of a non-power-of-2 number of memory banks.

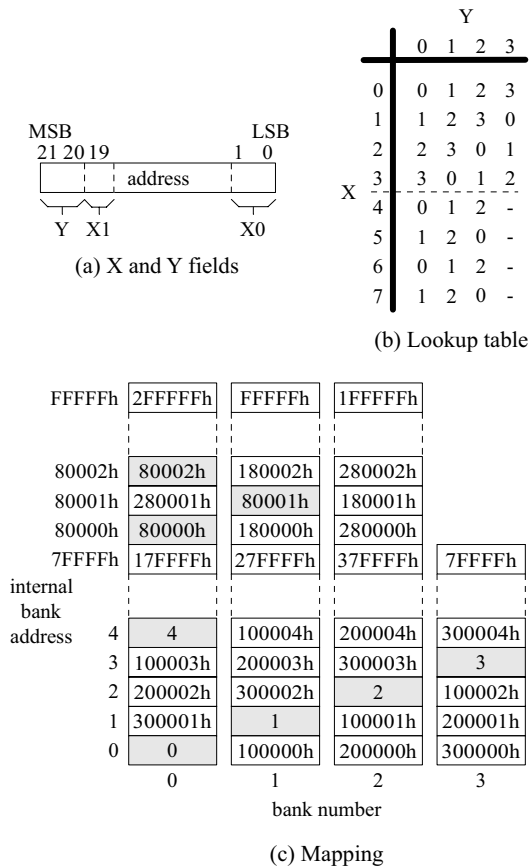


Figure 6: Support of non-equally-sized memory banks.

power-of-2 number of memory banks. Figure 6 shows an example of the support of non-equally-sized memory banks. This is based on dividing the address space into two segments, which are mapped on the lower part of all 4 memory banks and on the upper part of the first 3 memory banks as shown in figure 6(c).

## 5 Adaptive Memory

The mapping method can be used to build a main memory that can adapt itself to access patterns that occur in successive accesses to memory at different granularities in more or less dynamic ways. The intention is to achieve a better distribution of the memory accesses over the available memory banks in order to reduce bank conflicts and therefore reduce the average access time to memory. This section will discuss briefly how can be adapted to different types of access patterns, and how various forms of adaptation with different dynamics can be realized.

### 5.1 Adaptation to Access Patterns

Access patterns occur at different granularities, which are typically smaller than the entire stream of accesses to main memory. Only at the correct granularities will the corresponding access patterns be visible and can be adapted to by the address mapping scheme. The following two levels will now be differentiated:

1. Virtual page level. At this level, reference patterns occur within streams of accesses to parts of the address space comprised of single or multiple virtual pages.
2. Processor level. At this level, reference patterns occur within streams of accesses that relate to a single processor.

At the *virtual page level*, access patterns due to instruction fetching and data referencing occur. Instruction fetches typically have a sequential nature and result in accesses to address sequences with a stride 1. This type of access patterns goes well with traditional sequential interleaving based mapping schemes. Data references within scientific programs operating on dense arrays or matrices usually possess very regular patterns, whereas the references in pointer-intensive and sparse numerical computations are less predictable. It is well known how good mappings can be created for the first type of data references [5].

The mapping scheme can be adapted to the access patterns that occur at the virtual page level by mapping the individual segments within the address space that are used to store instructions and data structures, in the appropriate way for the access patterns to the stored information. This can be done at granularities as small as a single virtual page by creating multiple parallel mappings as described in Section 3.2. A segment within the address space containing data structures with regular access patterns can be mapped using emulation of well-known mappings such as prime degree interleaving as discussed in Section 3.3.

Each of the applied mappings also has to take into account the access patterns that occur at the *processor level*. These patterns can occur due to cache operation. An example is the replacement of a dirty cache line in a copy-back cache. As the cache index for the cache line that is replaced and the new cache line will be the same, pairs of read and write accesses will occur to addresses that differ from each other by a multiple of

$$\frac{\text{cache size}}{\text{cache line size} \times \text{set associativity}}$$

A mapping that is suitable for an address space segment containing instructions only and has to take into account the sequential nature of instruction fetching in combination with the access patterns that occur due to dirty cache line replacement of a copy-back cache can be based on the basic mapping described in Section 3.1. In this case the position  $q$  of the least significant bit of the Y field must be selected equal to

$$q = \log \left( \frac{\text{cache size}}{\text{cache line size} \times \text{set associativity}} \right).$$

The resulting mapping interleaves stride-1 address sequences over all memory banks and reduces the chance that the read and write accesses involved in a cache line replacement will go to the same memory bank to  $\frac{1}{M}$  in a memory system with  $M$  memory banks. This can be realized with the presented mapping scheme, but not with conventional mapping schemes based on sequential interleaving.

Simulations with programs from the SPEC benchmark suite have demonstrated that latency reductions of the order of 25% can be achieved for (high-end) computer systems and workloads today by adapting in the described way to the access patterns that are due to both instruction fetching and cache line replacement [2]. Improved mappings will allow further performance gains which become more important as the performance gap between microprocessor and DRAM technology continues to grow.

## 5.2 Static and Dynamic Adaptation

The mapping scheme can be adapted to access patterns by modifying the contents of the lookup table. This can be realized in more or less dynamic ways.

A more static adaptation can be used to adapt to system configuration updates (e.g., memory, processor and cache) or can be provided by the computer manufacturer as a form of workload tuning service for customers that run specific workloads. In this case the lookup table can be implemented in flash memory technology.

A simple dynamic adaptation can be realized using a memory controller that supports a set of predefined mappings for the entire address space or specific segments, which are selected through a configuration register. A somewhat more advanced method would be to perform the selection based on predetermined information of the workload and operating system, which is provided by the manufacturers for the most popular applications.

More dynamic versions of adaptation involve the use of monitoring and online learning functions to se-

lect a suitable mapping for a given workload. Alternatively the input for the adaptation could be obtained from off-line program analysis tools.

## 6 Summary and Conclusion

In this paper a new address mapping scheme was presented. The mapping scheme is based on a lookup table, which allows mappings to be changed dynamically. The mapping scheme supports several unique mappings that are not possible with traditional mapping schemes, and which provide very good interleaving performance for address sequences with multiple power-of-2 strides. Multiple mappings can be applied in parallel on distinct segments within the address space, which allows a better match with the reference behavior of single programs at granularities as small as a virtual page. The mapping scheme supports any number of memory banks with a limited variety of different bank sizes. The mapping scheme can be used to build a high-performance memory system that can adapt its operation to the way it is being accessed.

## References

- [1] D.C. Burger, J.R. Goodman, and A. Kägi, "Memory Bandwidth Limitations of Future Microprocessors," *Proceedings of the 23rd Annual International Symposium on Computer Architecture, Computer Architecture News*, Vol. 24, No. 2, pp. 78-89, May 1996.
- [2] J. van Lunteren, "Enhanced Computer Performance Through Adaptive Main Memory," Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, December 1998, ISBN 90-386-0500-5.
- [3] Q.S. Gao, "The Chinese Remainder Theorem and the Prime Memory System," *Proceedings of the 20th Annual International Symposium on Computer Architecture, Computer Architecture News*, Vol. 21, No. 2, pp. 337-340, May 1993.
- [4] B.R. Rau, "Pseudo-randomly Interleaved Memory," *Proceedings of the 18th Annual Symposium on Computer Architecture*, pp. 74-83, May 1991.
- [5] P.P. Budnik and D.J. Kuck, "The Organization and Use of Parallel Memories," *IEEE Transactions on Computers*, Vol. 20, pp. 1566-1569, December 1971.