A novel implementation of tile-based address mapping

Sambuddhi Hettiaratchi and Peter Y.K. Cheung

Department of Electrical and Electronic Engineering
Imperial College of Science, Technology and Medicine, London
E-mail: s.hetti@ic.ac.uk, p.cheung@ic.ac.uk

Abstract

Tile-based data layout has been applied to achieve various objectives such as minimizing cache conflicts and memory row switching activity. In some applications of tile-based mapping, the size of the tile can be assumed to be a power of two. In this paper, this 'power of two' assumption has been used to drastically simplify the tile-based address mapping functions. Once optimized, the implementation of the non-linear tile-based mapping consumes 60% less power than the implementation of the linear row-major mapping. This result is very interesting because one would normally expect a power penalty in the address generation stage of the more sophisticated tile-based mapping. Moreover, on average tile-based mapping implementation takes 10% less area and incurs virtually no additional delay over row-major mapping implementation.

1. Introduction

Multi-dimensional data arrays are a convenient way of organizing data in high-level hardware description languages. The index space of these multi-dimensional arrays must be mapped to the linear memory address space to order array elements in memory. The most commonly used layout functions are the row-major and the column-major mappings. However, there are other array linearization methods such as tile-based mapping [2, 3, 7, 5].

In [3], we applied tile-based mapping to minimize memory row switching activity. Minimizing memory row switching activity leads to a reduction in the energy consumption within the memories. However, it is not sufficient to consider the power dissipation of the memory alone. Row switching optimization changes memory address sequences and therefore affects power dissipation of the address generators. The implementation of the non-linear tile-based mapping function may consume more power than the implementation of the linear row-major mapping and offset any energy saving in the memories. In [3] the problem of finding the best tile dimensions have been formulated as a mesh partitioning problem formulation. In the mesh partitioning problem formulation each tile represents the set of data variables that will be mapped to the same memory row. Therefore, the size of the tile is equal to the size of a memory row. A careful observation of the memories reveal that the sizes of memory rows are usually powers of two. In this paper we present an interesting simplification of the tile-based address mapping equations using the assumption that the tile sizes are powers of two. We also present power, delay, and area results for this novel implementation of tile-based mapping.

'Tile-based' mapping is a loosely defined term. 'Tile-based' indicates that the data arrays are broken down into tiles, rectangle or windows. It does not define the way data variables are laid out inside the tiles and how the tiles themselves are ordered in memory. More specifically the exact tile-based mapping used in this work is 4D [2]. In 4D-tile-based mapping array data variables within the tiles and tiles themselves are ordered using row-major and column-major mappings. This gives rise to four mapping equations, from the array index space to the linear memory address space, and hence the name 4D.

The paper is organized as follows. Section 2 presents some of the reported applications of tile-based data layout. Section 3 defines the notation and technical terms used in the paper. Section 4 describes the address generator model used and Section 5 the 4D mapping equations. Section 6 illustrates how the 4D mapping equations can be simplified using the 'power of two' assumption. 4D mapping equations can only be used alone in special cases. Therefore, Section 7 presents how 4D mapping can be applied in the general case. Finally, Section 9 contains conclusions and indicates future work.

2. Previous Work

Tile-based data layout has been previously used to achieve various objectives. Some of the previous applications of tile-based mapping are briefly described here.

Chatterjee *et al.* [2] apply 4D-tile-based mapping to minimize cache conflicts. The size of the tile in their method is chosen in relation to the size of the cache. Kim and Park [5] use a tile-based mapping to increase memory bandwidth of SDRAMs in video processing applications. In their work the size of a tile is fixed to the size of a row. In both these cases it is reasonable to assume that the size of the tile is a power of two. However, Chatterjee *et al.* [2] do not report the use of the 'power of two' assumption and implement the 4D mapping in a microprocessor based environment and report low implementation costs. Although, Kim and Park [5] do not exactly use 4D mapping, it may also be possible to simplify their mapping implementation using the 'power of two' assumption. Instead, they implement their tile-based mapping as a programmable address generator with a latency of 26 clock cycles and a small power overhead.

Panda *et al.* [7] apply 4D-tile-based mapping to minimize off-chip address bus switching activity. They choose the best tile size by analyzing the loop structure to find a 'basic shape of access'. In their work it is somewhat difficult to assume that the size of a tile is a power of two. They present a hand-crafted implementation of the 4D mapping and offer an analytical power estimation for it. Power estimation shows that the implementation of 4D-tile-based address converter has a power overhead.

As mentioned in Section 1, in [3] 4D-tile-based mapping was applied to minimize memory row switching activity. However, that work concentrates on row switching minimization and do not consider the implementation of the tile-based mapping. The novel implementation of the 4D-tile-based mapping presented in this work extends the work in [3]. However, if the size of the tile can be assumed to be a power of two then simplification of 4D-tile-based implementation presented in this paper can be applied to other applications of tile-based mapping as well.

3. Notation

The data array dimension are indicated by $W \times H$, where W is the width and H is the height of the array. The tile dimensions are given by $m \times n$, where m is the width and n is the height of the tile. x and y are the horizontal and vertical array indices of a two dimensional data array respectively. The function $f(x,y)$ represent a mapping from the x-y array index space to the linear memory address space. The number of memory words contained in a row (size of a row) is given by q and also referred to as number of memory columns. The number of rows is represented by p.

Terms such as 'data layout', 'address assignment' and 'address mapping' are used interchangeably in this paper.

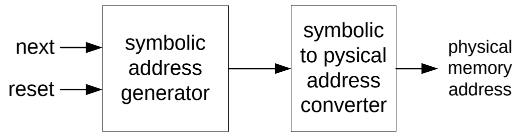
next
reset
symbolic address generator
symbolic to pysical address converter
physical memory address

Figure 1. Address generator model

4. Address Generator Model

There are many ways of implementing address generation hardware. In the model used, the address generator is functionally divided into a symbolic address generator and a symbolic to physical address mapping module (See Figure 1). The symbolic addresses of array data variables can be represented by their array indices. Therefore, the symbolic address generator can be implemented in the custom Address Calculation Unit (cACU) style preferred by Miranda *et al.* [6] at IMEC.

The symbolic to physical address converter is determined by the particular data layout method used. However, the symbolic address generator is independent of it. Therefore, the difference in energy consumption of the address generator for different address assignment schemes can be investigated without having to analyze the symbolic address generator at all. There are different degrees of inaccuracy associated with any energy estimation method, and therefore errors can be kept low by estimating energy for minimal functional units. The functional partitioning of the address generator shown in Figure 1 is general, elegant and convenient.

5. 4D-tile-based data layout equations

Figure 2 shows a data array that has been partitioned into $m \times n$ tiles on the x-y index space. When the tiles are ordered row-major and the data variables within the tiles are ordered column-major in memory, then equation $f_{rc}(x,y)$ defines a mapping from the x-y index space to the linear memory address space. Let's say that the memory address for an arbitrary data variable at position (x,y) needs to be found. On the figure there are four different regions which are numbered from 1 to 4, and Equation 1 has sub-expressions correspondingly numbered 1-4 as well. Each sub-expression gives the number of symbolic addresses contained in each region. Regions 1-2 correspond to the row-major ordering of rectangles, and regions 3-4 correspond to the column-major ordering of the data variables within the rectangles. The physical address of a particular symbolic address is found by summing the number of symbolic addresses in each region, as shown in Equation 1.

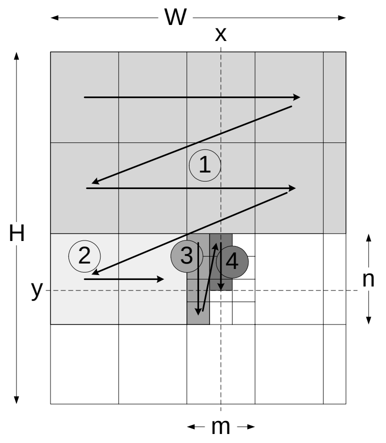

Figure 2. A tiled data array and data variable ordering for equation $f_{rc}(x,y)$. $0 \leq y < H$ and $0 \leq x < W$.

$$f_{rc}(x,y) = \underbrace{(y - y \bmod n)W}_{1} + \underbrace{(x - x \bmod m)n}_{2} + \underbrace{(x \bmod m)n}_{3} + \underbrace{y \bmod n}_{4} \tag{1}$$

Equations 2 to 4 define the other three possible 4D mapping functions. The first subscript of $f(x,y)$ refers to the ordering of the tiles, and the second subscript refers to the ordering within the tiles.

$$f_{rr}(x,y) = (y - y \bmod n)W + (x - x \bmod m)n + (y \bmod n)m + x \bmod m \tag{2}$$

$$f_{cr}(x,y) = (x - x \bmod m)H + (y - y \bmod n)m + (y \bmod n)m + x \bmod m \tag{3}$$

$$f_{cc}(x,y) = (x - x \bmod m)H + (y - y \bmod n)m + (x \bmod m)n + y \bmod n \tag{4}$$

6. Simplification of 4D layout equations

Although the simplification presented in this paper is applicable to all four 4D mapping equations, due to space limitations only one equation is considered here. In order to demonstrate the simplifying effect of the 'power of two' assumption, Equation 1 is rearranged to Equation 5.

$$f_{rc}(x,y) = W(y - y \bmod n) + xn + y \bmod n \tag{5}$$

If it can be assumed that the number of memory columns is a power of two, Equation 5 can be significantly simplified. However is this 'power of two' assumption reasonable?

A careful observation of the configurations of external memories would reveal that both the number of memory columns and rows are almost always power of two [4, 8]. This power of two organization is, in fact, due to binary coding of memory addresses. The row address consists of the $log_2(p)$ MSBs, and the the column address consists of the $log_2(q)$ LSBs. What about embedded memories? Usually, embedded memories are designed using memory compilers. So it is quite normal for a designer to request an unusual size of memory. However, even in this case, because the column addresses are usually the LSBs, the number of columns is a power of two number. Even, if the number of memory columns were not usually power of two, there is no reason why the memories cannot be designed to satisfy the 'power of two' assumption. The 'power of two' assumption is a very reasonable one and, therefore, was used to optimize the 4D address mapping equations.

Since the size of a tile ($m \times n$) is equal to the number of memory columns (q), if q is a power of two number then both m and n are power of two numbers. When n is a power of two number, then the following simplifications happen. $y \bmod n$ simply means take the $log_2(n)$ LSBs of y. Therefore, $y - (y \bmod n)$ equates to setting the $log_2(n)$ LSBs of y to zero. Thus far, a mod function and a subtractor have been eliminated. The xn multiplication, also reduces to a left shift of x by $log_2(n)$ bit positions, with the $log_2(n)$ LSBs of x taking '0' values. Since $log_2(n)$ LSBs of x take '0' values, and $y \bmod n$ is a $log_2(n)$ bit number, the xn multiplication and its addition to $y \bmod n$ can be implemented by simply concatenating x and $log_2(n)$ LSBs of y. After these simplifications, only one multiplier and one adder is required to implement Equation 5. This is further demonstrated in Example 1. Note that, row-major mapping also requires one multiplication and one addition (See Equation 6).

$$f_r(x,y) = (W \times y) + x \tag{6}$$

Example 1 Consider an 80×80 data array, which is stored in a memory with 256 rows and 32 columns. Given that the optimal tile/rectangle shape for the access pattern is $m = 8$ and $n = 4$, Figure 3 shows the optimized implementation of the symbolic to physical address mapping function (Equation 5), for this example.

Note that although the multiplier has two 7-bit inputs the output is a 13-bit number. This is due to the restricted range of the operands, $0 \leq x < 80$, $0 \leq y < 80$. The restricted range of the operands also ensures that when the 9-bit and 13-bit numbers are added together the output can be represented by 13-bits. ■

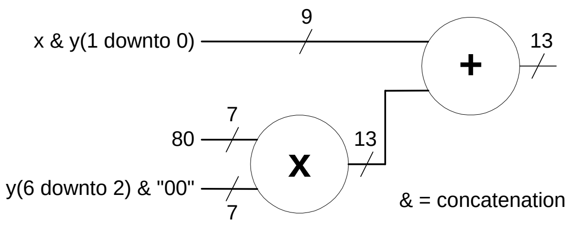

Figure 3. Mapping function implementation for Example 1. $0 \leq x < 80$ **and** $0 \leq y < 80$**.**

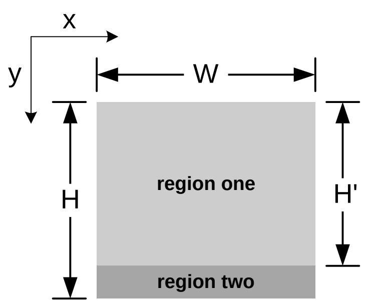

Figure 4. Division of transition mesh into two regions for the general case of 4D address mapping. $0 \leq x < W$ **and** $0 \leq y < H$**.**

7. General Case of Symbolic to Physical Address Conversion

In the special case where H is an integer multiple of n, $(H \bmod n) = 0$, the address converter module can be represented by Equation 5 alone. In general, Equation 5, although a very important part, is only one constituent part of the symbolic to physical address mapping module.

The first step in the solution to the general case is to divide the transition mesh into two regions as shown in Figure 4. The first segment is $W \times H'$, where $H' = H - (H \bmod n)$. The second segment is $W \times (H \bmod n)$. The symbolic addresses in the first region are assigned addresses using Equation 5. The symbolic addresses in the second region are assigned addresses using row-major scheme.

As the two regions use different address mapping equations, it is necessary to chose which region a particular symbolic address belongs to. This is done by checking if $y < H'$. At first glace it may seem that a '$<$' operator is required to implement the checking, however this is not so.

If a symbolic address is in the second region, then $H' \leq y < H' + n$. Moreover, n is a power of two and H' is a multiple of n. Therefore, all numbers from H' to $H' + n - 1$ are equal to H', if $log_2(n)$ LSBs of y are set to zero. This

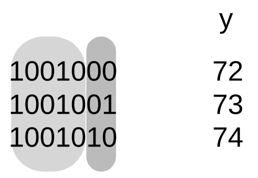

Figure 5. Looking at upper bits is sufficient to identify if $y < 72$**, where** $0 \leq y < 75$**.**

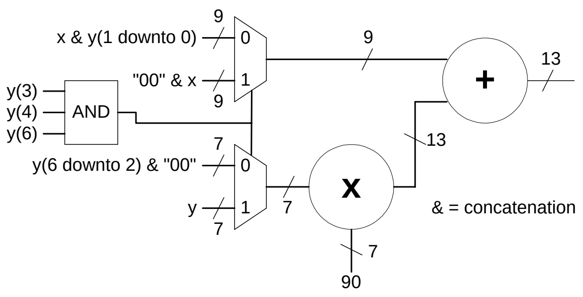

Figure 6. Symbolic to physical address converter implementation for Example 3. $0 \leq x < 90$ **and** $0 \leq y < 90$

fact is further illustrated in Example 2. As a result of this property, it is possible to simply look at the y index after setting $log_2(n)$ LSBs to zeros. If this modified number is equal to H'; then $y \geq H'$, and a data variable in region two is accessed.

Example 2 Consider a 75×75 data array, which is stored in a memory with 256 rows and 32 columns. The optimal tile/rectangle size for the access pattern has been found to be $m = 8$ and n = 4. $H' = 72$, so when `y` is 72,73 or 74, row-major mapping should be selected. When `y` is 72,73, or 74, `y[6 downto 2]` bits are identical as shown in Figure 5,($0 \leq y < 75$).■

Both simplified 4D-tile-based and row-major address assignment schemes use one multiplier and one adder. Therefore, the adder and the multiplier can be shared by using multiplexors. Example 3 shows a symbolic to physical address converter implementation for the general case of 4D-tile-based address assignment method.

Example 3 Consider a 90×90 data array, which is stored in a memory with 256 rows and 32 columns. Given that the optimal tile/rectangle shape for the access pattern is $m = 8$ and n = 4, Figure 6 shows the optimized implementation of the symbolic to physical address mapping function for this example.

Note that although the multiplier has two 7-bit inputs the output is a 13-bit number. This is due to the restricted range of the operands, $0 \leq x < 90$, $0 \leq y < 90$. The restricted range of the operands also ensure that when the 9-bit and 13-bit numbers are added together the output can be represented by 13-bits. ■

8. Experimental Results

8.1. Power Results

In this section results are presented which demonstrate that the 4D-tile-based address assignment hardware, when optimized using the 'power of two' assumption, actually consumes less power than row-major address assignment hardware. This result is surprising. Intuitively, one would expect a power penalty in the address generation stage of the more sophisticated 4D-tile-based mapping over that of relatively simple row-major mapping.

The power estimation procedure can be briefly described as follows. The sum-of-products specification of symbolic to physical address mapping was synthesized using Synopsys Design Compiler to a commercial 0.18 μm technology library. The gate level netlist was then extracted from Synopsys as a VHDL file and simulated in ModelSim. Using ModelSim the toggle counts and signal probabilities for each node of the circuit were collected and fed to Synopsys Design Power. Design Power then annotates the gate level netlist with the toggle and signal probability information and produces an average power estimate.

Figure 7 shows the power dissipation of the row-major and 4D-tile-based mapping modules for 'compress' access sequence. The mesh partitioning based method on average dissipated 66% less power compared to the row-major method. Moreover, for every one of the data array sizes, 4D-tile-based hardware dissipated less power than row-major hardware.

The reduction in the power dissipation that can be seen for the 4D-tile-based case is two-fold. The multiplier in Figure 6 performs a constant multiplication. The variable inputs are `y(6 downto 0)` for the row-major case and `y(6 downto 2) & "00"` for the 4D-tile-based case. For the 'compress' access sequence, the two least significant bits of `y` switch much more frequently than other bits. In fact, the two LSBs account for about 75% of all the `y` input bit switching activity. The multiplier, in the 4D-tile-based case, is subjected to 75% less input switching activity compared to row-major case. Therefore, the multiplier dissipates considerably less power.

The adder has two inputs, one of which is the output of the multiplier. The multiplier output would, in general, switch fewer times in the 4D-tile-based case. The second input to the multiplier is `x` for the row-major case and `x & y(1 downto 0)` for the 4D-tile-based case. Note that, in the 4D-tile-based address mapping hardware, the two LSBs of `y` which were not connected to the input of the multiplier are now connected to the input of the adder. So the adder would be subjected to the 75% of `y` input switching activity that the multiplier was not. `x` input switches about 50% more frequently than `y` input for this access sequence. In the row-major case the `x` inputs are connected to lower bit positions in the adder in comparison to 4D-tile-based case. The adder is a ripple-carry adder, and therefore, switching activity in the LSBs of the adder, generally, result in higher overall switching activity than switching activity in MSBs due to carry propagation.

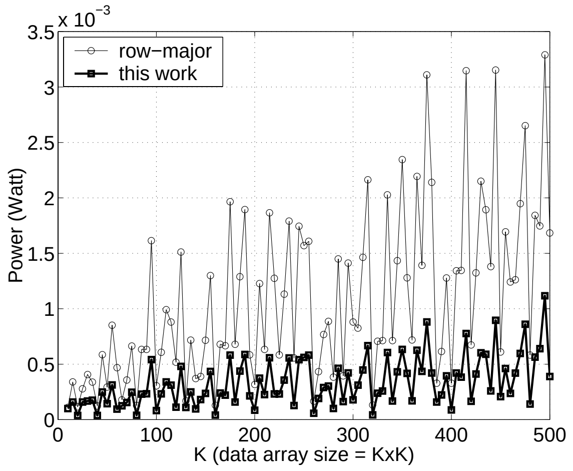

Figure 7. Data array size against power in Watts for symbolic to physical address mapper. 'compress' access sequence. @ 200MHz and 1.71V.

Example	Average % power reduction
Compress	66.4
DCT	38.7
GSR	71.2
SOR	64.4

Table 1. Average percentage power dissipation reduction results over row-major mapping.

Table 1 shows power dissipation reduction results for a number of example access sequences. The average percentage power reduction for the example access sequences is 60%.

The 4D-tile-based mapping was used in the mesh partitioning based address assignment method [3] to save power in the memory cell array. However, in this work it has been shown that the 4D-tile-based method, when optimized under the ‘power of two’ assumption can also save power in the symbolic to physical address converter.

8.2. Area and Delay Results

The area and delay results of the address converter, for the two address mapping methods are determined by the data array sizes. For the 4D-tile-based method, m and n values also influence area and delay.

On average, for $m = 8$ and $n = 4$, 4D-tile-based symbolic to physical address mapping hardware is only 0.7% slower than row-major hardware. For some data array sizes, 4D-tile-based address mapping hardware is even faster than row-major hardware.

Area estimates from Synopsys Design Compiler indicate that on average 4D-tile-based method achieves 10% area reduction in the address mapping hardware compared to the row-major method. In the special case where $(H \bmod n) = 0$, 4D-tile-based hardware takes considerably less area than row-major mapping module (on average 32%). This is due to the fact that, in the 4D-tile-based case, the $log_2(n)$ LSBs of one of the multiplier inputs are always set to ‘0’. So the constant multiplier can be simplified. Also, because $log_2(n)$ LSBs of one of the multiplier inputs are always set to ‘0’s, the same number of outputs are always ‘0’s. Therefore, the first $log_2(n)$ stages of the ripple carry adder can be eliminated. However, in the case where $(H \bmod n) \neq 0$, due to the extra multiplexors and multiplexor control signal generators, 4D-tile-based address mapping module generally takes more area. However, the amount of extra hardware, especially as data array size increases is very small (on average, 2%).

9. Conclusion

Tile-based data layout has been used to achieve various objectives. In some applications of the tile-based mapping it can be assumed that the size of the tile is a power of two. In this paper this ‘power of two’ assumption has been used to significantly simplify the 4D-tile-based mapping functions. Once optimized the more sophisticated 4D-tile-based mapping achieves 60% power reductions and 10% area savings with virtually no performance penalty over row-major mapping implementation.

In this work the address generator was functionally partitioned into a symbolic address generator and a symbolic to physical address mapper. This is a convenient and elegant implementation. However, in ASICs, by combining the symbolic address generator and symbolic to physical address mapper into one unit, further optimizations may be possible. For example, when memory accesses are performed in regular loop structures, it may be possible to apply techniques such as induction variable analysis and strength reduction [1] to replace multiplications with addition operations. The contribution made in this paper by using the ‘power of two’ assumption to simplify the 4D mapping functions is still valid even in this case. The complexity of the 4D mapping function has been reduced to one constant multiplication and one addition: the same complexity of the row-major mapping. Therefore, any strength reduction optimization that is performed on the row-major mapping equation can also be applied to mesh partitioning based mapping equations.

The current hardware implementation of the 4D-tile-based mapping is application specific, therefore a different address mapping unit is required for each data array, unless two or more data arrays have the same tile dimensions. It is interesting future work to investigate the effects on power dissipation as more programmability is added to the address mapper. Moreover, the ‘power of two’ assumption about the tile size can be used to simplify other types of tile based mappings in the future.

Acknowledgement

This work was funded by LSI Logic Corporation (USA) and Overseas Research Students Awards Scheme (UK).

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[2] S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra, and M. Thottethodi. Nonlinear array layouts for hierarchical memory systems. In *Int. Conf. on Supercomputing*, pages 444 – 453, 1999.

[3] S. Hettiaratchi and P. Y. Cheung. Mesh partitioning approach to energy efficient data layout. In *Proc. of the DATE*, pages 1076–1081, Mar. 2003.

[4] T. Hirose, H. Kuriyama, S. Murakami, K. Yuzuriha, T. Mukai, K. Tsutsumi, Y. Nishimura, Y. Kohno, and K. Anami. A 20-ns 4-Mb CMOS SRAM with hierarchical word decoding architecture. *IEEE J. of Solid-State Circuits*, 25:1068 – 1074, Oct. 1990.

[5] H. Kim and I.-C. Park. High-performance and low-power memory-interface architecture for video processing applications. *IEEE Trans. on Circuits and Systems for Video Technology*, 11:1160–1170, Nov. 2001.

[6] M. Miranda, F. Catthoor, and M. J. H. D. Man. High-level address optimization and synthesis techniques for data-transfer-intensive applications. *IEEE Trans. on VLSI Systems*, 6(4):677 – 686, Dec. 1998.

[7] P. R. Panda and N. D. Dutt. Low-power memory mapping through reducing address bus activity. *IEEE Trans. on VLSI Systems*, 7(3):309–319, Sept. 1999.

[8] T. Seki, E. Itoh, C. Furukawa, I. Maeno, T. Ozawa, H. Sano, and N. Suzuki. A 6-ns 1-Mb CMOS SRAM with latched sense amplifier. *IEEE J. of Solid-State Circuits*, 28:478 – 483, Apr. 1993.