A High-Performance Hardware-Efficient Memory Allocation Technique and Design

Hasan Çam, Mostafa Abd-El-Barr, and Sadiq M. Sait
King Fahd University of Petroleum and Minerals
Computer Engineering Department
Dhahran 31261, Saudi Arabia

Abstract

This paper presents a hardware-efficient memory allocation (EMA) technique designed to eliminate both internal and external fragmentation that appear in the buddy system. *EMA can allocate a free memory block of* any *size in any part of memory. Hardware implementation of EMA is introduced, but only part of its circuits is shown in the paper due to the space limitation. Simulation results show that EMA utilizes memory space more efficiently than the previously known techniques.*

1 Introduction

Dynamic memory allocation is an important issue in the design of computer systems. It has been reported that dynamic memory management consumes 23%−38% of the time in six allocation-intensive *C* programs run on 17-SPECmarks SPARC architecture with 80 MB of memory [1]. Object-oriented programs have a very high object creation rate and, therefore, the speed of memory allocation is crucial for improving the system performance. It is highly desirable to have a fast and efficient memory allocator that allocates free space in blocks of exactly the prescribed length.

A number of memory allocation algorithms have been implemented in hardware, but each one of these has some drawbacks. The *buddy* system, introduced by Knowlton [2], is a fast and simple memory allocation technique. It allocates memory in blocks whose lengths are power of 2 and, therefore, suffers from high internal and external fragmentation. If the requested block size is not a power of 2, then the size is rounded up to the next power of two. This may leave a big chunk of unused space at the end of an allocated block [3], thereby resulting in *internal fragmentation.* Many researchers have focused on improving the performance of the buddy system using either software techniques [4, 5] or hardware techniques [3]. Chang and Gehringer [6] have recently proposed a modified hardware-based buddy system which eliminates internal fragmentation. This paper presents an efficient hardware technique designed to detect any available free block of requested size in the well-known memory allocation technique *buddy system* and to eliminate internal fragmentation.

2 Efficient Memory Allocation (EMA) System

We assume that the memory is divided into a number of chunks, each having the same number of words. A memory block consists of one or more chunks. A bit-vector is used to represent the status (free or used) of all memory chunks such that bits 0 and 1 correspond to the case of a chunk being free or used, respectively.

Next, we introduce algorithm EMA which receives two types of requests, namely, allocation and deallocation, along with the size k of the requested block.

Algorithm EMA
Step 1. If request is deallocation, go to Step 5; otherwise, (i.e., allocation), go to Step 2.
Step 2. (i) Detect all free blocks of size $2^{\lceil \log_2 k \rceil}$, (ii) activate the address registers of these free blocks.
Step 3. (i) Detect the free block with the highest starting address among those free blocks, (ii) return the starting address of this free block to the memory manager (indicating that the first k chunks of the block are free).
Step 4. Invert (from 0 to 1) all those k bits corresponding to the first k chunks of the free block. *End of Allocation.* Stop
Step 5. Invert (from 1 to 0) all those k bits whose leftmost bit's address equals the given starting address of the block to be deallocated.

Detection of Free Blocks (*Step 2*): The *or*-gate prefix circuit shown in Figure 1(a) is used to detect all free blocks of size $2^{\lceil \log_2 k \rceil}$. The circuit assumes a memory of size N chunks. Any node at level L_i represents an OR gate. For $i \geq 1$, the number of nodes at level L_i of the *or*-gate prefix circuit is 2^{i-1} less than the number of nodes at level L_{i-1}. An example circuit for $N = 10$ is shown in Figure 1(b).

The *or*-gate prefix circuit can detect any free block if its size is a power of 2, wherever the free block is located in the memory. This is an advantage over the *or*-gate tree [6] which can only detect those free blocks of size j whose starting address is a multiple of j, where j is a power of 2. To determine the free block whose first chunk's address is the greatest, a so-called level selector line S_i is placed into level i, $0 \leq i \leq n$, of the *or*-gate prefix circuit as shown in Figure 2. There are $n+1$ level selectors labeled $S_0,\ S_1, \ldots, S_n$ for a 2^n-bit vector. When a block of size k is requested, only level selector S_i, for $i = \lceil \log_2 k \rceil$, is asserted. For any free block of size 2^i, there will be exactly one corresponding *or*-gate node with value 0 at level L_i of the *or*-gate prefix circuit.

Detection of the Free Block with the Highest Address (*Step 3*): The bits of a bit-vector are labeled from left to right, starting with 0. The label of each bit is stored in its address register circuit (ARC). When more than one address register is set by the vertical lines, the selection of the address register with highest address is achieved by the hardware implementation of the *binary countdown* algorithm proposed by Fraser [7]. The detect-first tree and disabling circuits shown in Figure 2 implement the binary countdown algorithm in n steps as follows. Assume that the address of the kth block in the bit vector is represented in binary by $x_k^1 x_k^2 \ldots x_k^n$. In the jth step, $1 \leq j \leq n$, the x^j bits of the addresses of the enabled-address-registers are ORed by the detect-first tree, starting with $j = 1$ in the first step.

Bit Inversion (*Step 4*): Let SA and EA denote the *starting* and *ending* addresses, respectively, of the block determined by the circuit shown in Figure 2. Note that, given the size S of the requested block, EA can be easily computed using the formula $EA = SA + S - 1$. If the bits corresponding to the allocated block are represented by a subvector V of the bit-vector, SA and EA correspond to the addresses of the first and last bit, respectively, of V. To indicate that the bits of V are allocated, they are inverted to

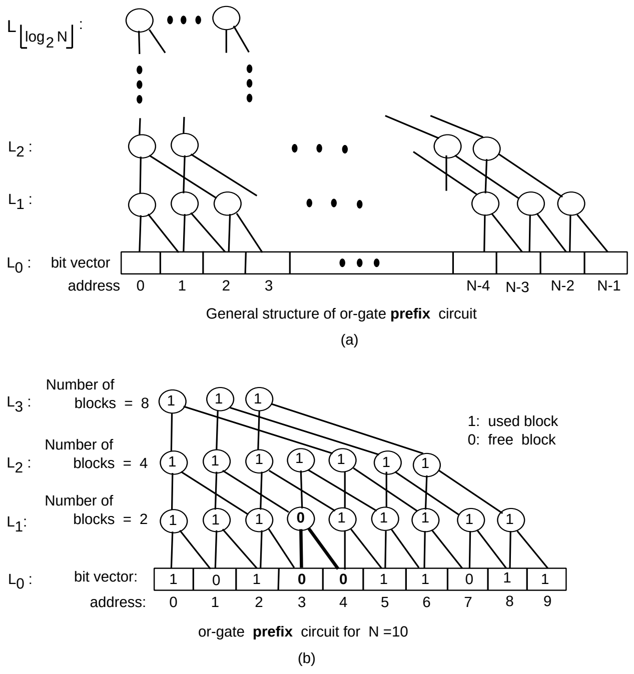
$L_{\lfloor \log_2 N \rfloor}$:
L_2 :
L_1 :
L_0 :
bit vector
address 0 1 2 3 N-4 N-3 N-2 N-1
General structure of or-gate **prefix** circuit
(a)
L_3 : Number of blocks = 8
L_2 : Number of blocks = 4
L_1: Number of blocks = 2
L_0 : bit vector: 1 0 1 0 0 1 1 0 1 1
address: 0 1 2 3 4 5 6 7 8 9
1: used block
0: free block
or-gate **prefix** circuit for N =10
(b)

Figure 1: (a) General structure of an or-gate prefix circuit for a memory of size N chunks. (b) An *or*-gate prefix circuit for a bit-vector of length $N = 10$.

1. As soon as the free block with the highest address is determined, the allocated blocks can be used, while the bit inversion is simultaneously done to update the status of the bit vector.

Memory Deallocation (*Step 5*): In case of *memory deallocation*, the starting address SA and the size S of the block to be deallocated are given. The only difference between memory allocation and memory deallocation is that the starting address SA is given and, bit-inverters invert the bits from 1 to 0 to indicate that they are free.

3 Simulation Results

We have conducted a number of experiments to evaluate EMA. In each experiment, a supply of synthetic allocations/deallocations was created with their execution times. EMA is compared with the buddy system [3] and the memory allocator (CGMA) of Chang and Gehringer [6]. To evaluate EMA, we define a measurement called *memory allocation efficiency* (MAE) as being the ratio of the number of requested

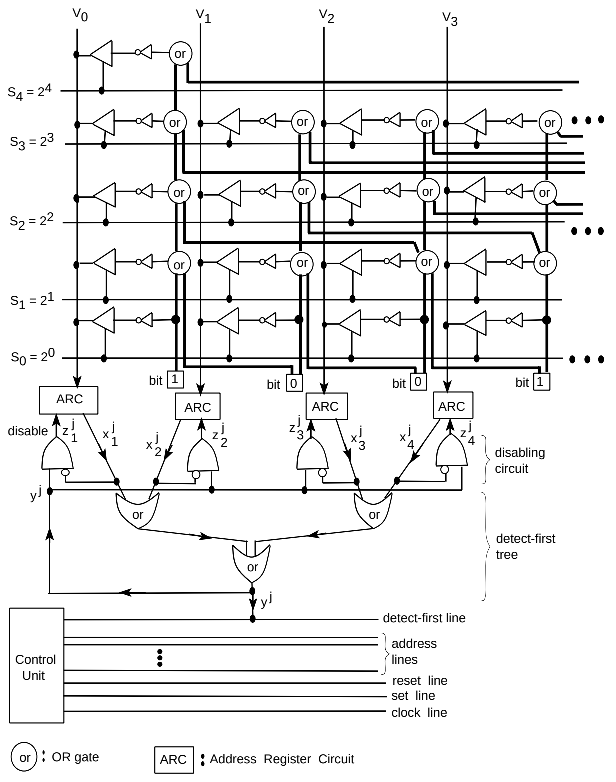

Figure 2: The detect-free-block (DFB) circuit for determining the free block with the highest starting address, for a 16-bit vector; only four bits of the vector are shown due to space limitations. The lines of the *or*-gate prefix circuit are illustrated by the thick solid lines.

memory blocks to the number of allocated memory blocks. Figure 3 shows MAE versus memory space.

4 Conclusion

EMA is fast and flexible enough to allocate/deallocate a free block in any part of memory. This leads to better utilization of memory space, thereby allowing more memory blocks to remain free than is possible with the known hardware memory allocators.

Acknowledgment

The authors wish to acknowledge the support provided by King Fahd University of Petroleum and Minerals under Project No. COE/ARRAYS/177.

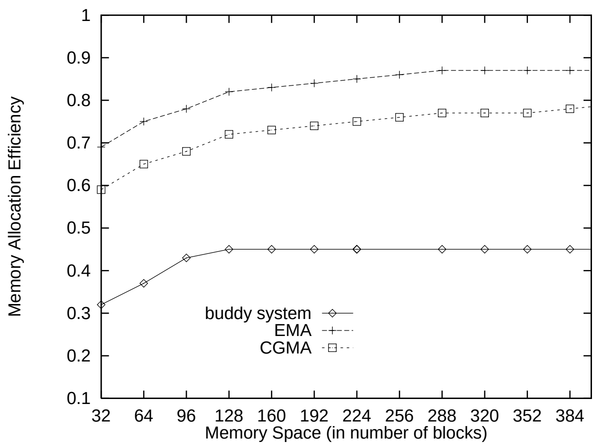

Figure 3: Comparison of memory allocation efficiency versus memory space for EMA, CGMA, and buddy system.

References

[1] B. Zorn, "The measured cost of conservative garbage collection," *Software-Practice and Experience,* Vol. 23, No. 7, pp. 733-756, July 1993.

[2] K.C. Knowlton, "A fast storage allocator," *Comm. ACM,* Vol. 8, pp. 623-625, Oct. 1965.

[3] E.V. Puttkamer, "A simple hardware buddy system memory allocator," *IEEE Trans. Computers,* Vol. 24, No. 10, pp. 953-957, Oct. 1975.

[4] I.P. Page and J. Hagins, "Improving the performance of buddy systems," *IEEE Trans. Computers,* Vol. 35, No. 5, pp. 441-447, May 1986.

[5] R.E. Barkley and T.P. Lee, "A lazy buddy system bounded by two coalescing delays per class," *Proc. 12th Symp. Operating Systems Principles,* Vol. 23, No. 5, pp. 167-176, Dec. 1989.

[6] J.M. Chang and E.F. Gehringer, "A high performance memory allocator for object-oriented systems," *IEEE Trans. on Computers,* Vol. 45, No. 3, pp. 357-366, March 1996.

[7] A.G. Fraser, "Towards a universal data transport system," *Advances in Local Area Networks,* K. Kummerle, F. Tobagi and J.O. Limb (Eds), New York: IEEE Press, 1987.