

## **Organization for a Parallel Optical Memory Interface**

Gregory Deatz and Miles Murdocca

Department of Computer Science, Hill Center

Rutgers University, New Brunswick, NJ 08903

(908) 445-2001(phone), (-0537 fax)

deatz@paul.rutgers.edu (Deatz); murdocca@cs.rutgers.edu (Murdocca)

**NOTE: This appeared as: Deatz, G. and M. Murdocca, “Organization for a Parallel Optical Memory Interface,” in *Optical Computing*, vol. 10, 1995 OSA Technical Digest Series, (Optical Society of America, Washington, DC, 1995), Salt Lake City, pp. 80-82.**

**Miles Murdocca is now with IIUSA: [murdocca@iisatech.com](mailto:murdocca@iisatech.com)**

### **Abstract**

An arbitrarily sized region of interest is read from or is written to a parallel mass storage device in logarithmic time, in a concept optically addressed memory architecture.

# Organization for a Parallel Optical Memory Interface

Gregory Deatz and Miles Murdocca

Department of Computer Science, Hill Center

Rutgers University, New Brunswick, NJ 08903

(908) 445-2001(phone), (-0537 fax)

deatz@paul.rutgers.edu (Deatz); murdocca@cs.rutgers.edu (Murdocca)

## 1. Introduction

Scientific computation typically involves processing large amounts of data in which operations using main memory and mass storage are frequent. A performance bottleneck to and from main memory arises because only a small number of input and output pins are provided on electronic memory integrated circuits, which means that large portions of the memory cannot be accessed in parallel. This bottleneck is compounded by optical memories, in which entire pages can be brought into a system in parallel. Here, we describe a concept architecture that improves the access time to a parallel memory through the use of an optical interface.

Figure 1a illustrates the model for the optical memory interface. At the lowest level, data objects of various sizes are stored in an optical recording medium. Rectangular areas are illuminated in parallel, and the read-out beams from the storage plane are redirected to a staging plane where data objects tile a regularly shaped area. In Figure 1a, three rectangularly shaped data objects that are arbitrarily placed in the storage plane are imaged onto the staging plane so that they fit tightly within a single rectangle.

After the data objects are organized in the staging plane, they are distributed in parallel in the optical distribution plane to a host system through a parallel read/write window. The reverse process is used when writing in parallel. The beam redirection can be performed with a beam-blocking approach [1], in which the beams are fanned to a number of locations, and selective blocking controls the target locations of the beams. In an alternative low latency approach, the redirection can be performed with beam-steering [2].

An advantage of this memory organization is that once the beam-blocking/beam-steering mechanism is configured, parallel reads and writes can be made indefinitely without incurring the overhead of reconfiguration. Parallel memory traffic patterns repeat [3], and so even a slow reconfiguration mechanism can be effective if a high bit rate is maintained after reconfiguration.

## 2. The Model

Figure 1b shows the external view of the parallel memory interface. The ADDRESS port is used for a logical address that is internally mapped to a physical location. The correspondence between logical and physical

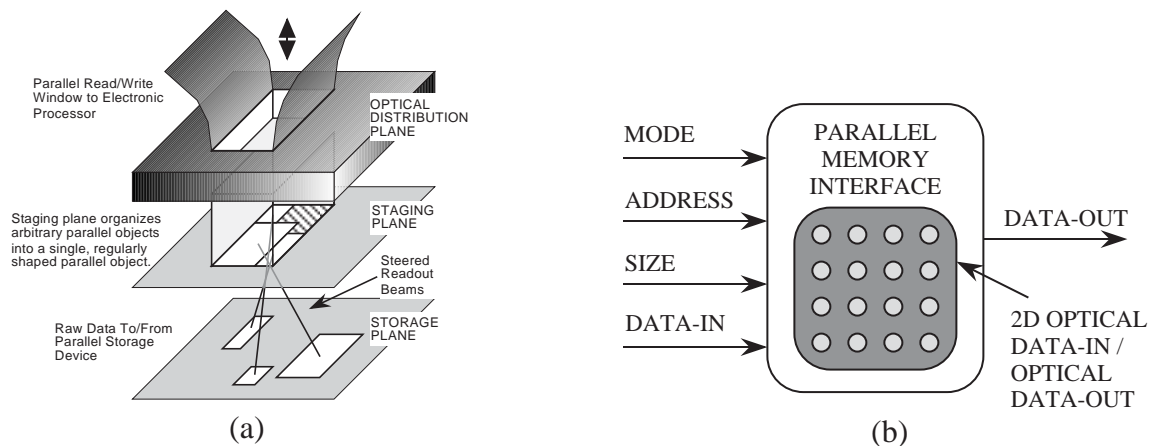


Figure 1: (a) Model for the reconfigurable optical memory interface; (b) external view.

addresses may change during operation. For parallel addressing, the ADDRESS port holds the starting address of a block, and the SIZE port holds the size of the block that is being accessed, excluding the first address. Thus, to access the first four logical addresses in the memory in parallel, the ADDRESS port should be 0, and the value on the SIZE port should be 3.

The DATA-IN and DATA-OUT ports transfer scalar (single word) data between the memory and an electronic host. The OPTICAL DATA-IN and OPTICAL DATA-OUT ports transfer block data between the memory and an optical storage device. The value at the MODE port can take on one of six values:

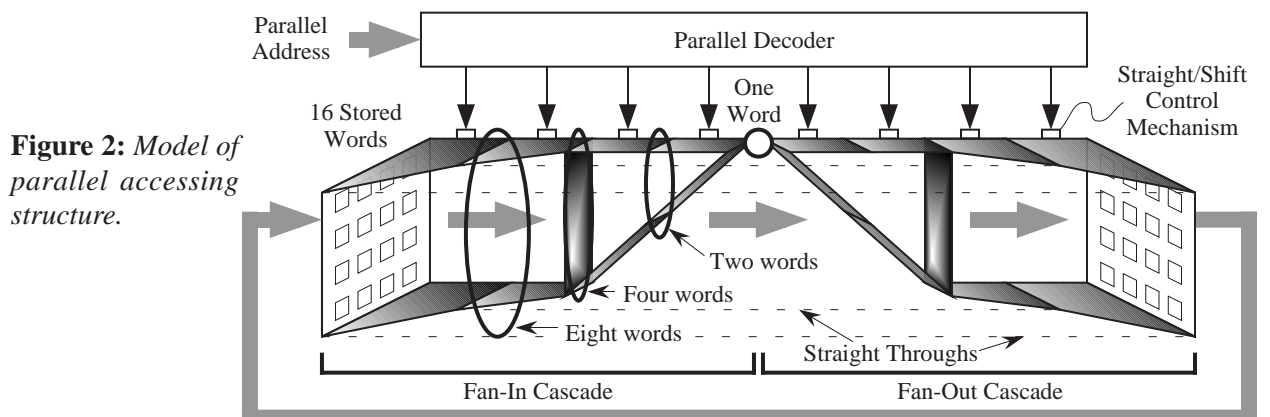
- 0 (Read) or 1 (Write): Perform a scalar Read or Write on the memory location at the ADDRESS port.
- 2 (Parallel Read) and 3 (Parallel Write): Perform a block Read or Write. The block appears at the OPTICAL DATA-OUT port or is read from the OPTICAL DATA-IN port as appropriate for the operation.
- 4 (Internal Parallel Copy - IPC): Copy a block of words from one section of memory to another.
- 5 (Contiguize): Memory locations are moved so that they are physically contiguous (that is, they fill a block), while retaining their logical addresses. This makes subsequent parallel reads, parallel writes, and IPCs on arbitrarily shaped data objects more efficient. When the Contiguize mode is maintained, the memory treats every new address as an addition to the object. When the MODE field changes, the object is then accessible in parallel. This function is useful, for example, when accessing a sparse matrix in its entirety.

Data objects are reshaped during operation to conform to a simple tree structure, as shown in Figure 2. The  $N$  word memory ( $N = 16$  for this example) is fanned in through a  $\log_2 N$  tree cascade to extract an entire subtree of words (1, 2, 4, 8, or 16 for this example). The extracted object is then distributed through a fan-out cascade. At each stage, data objects are either combined or split apart, or are sent straight through without any fan-in or fan-out, as directed by the Parallel Decoder. In this way, arbitrary data objects can be selected that are an integral power of two in size, that fall on integral power of two boundaries in the address space of the memory. Arbitrarily shaped objects placed at arbitrary boundaries in the memory, however, cannot be directly manipulated. We address this problem in the next section through a series of parallel accesses.

### 3. Reshaping Memory for Efficient Parallel Access

A four level deep decoder tree for a 16-word memory is shown in Figure 3. As an example of how the decoder tree works, the address 1011 is presented at the root node (the top level of the tree). The leftmost bit in the address is a 1 so the right path is traversed at Level 0 as indicated by the arrow. The next bit is a 0 so the left path is traversed at Level 1. The next bit is a 1 so the right path is traversed at Level 2, and the last bit is a 1 so the rightmost path is traversed and the addressed leaf 1011 is reached at Level 3.

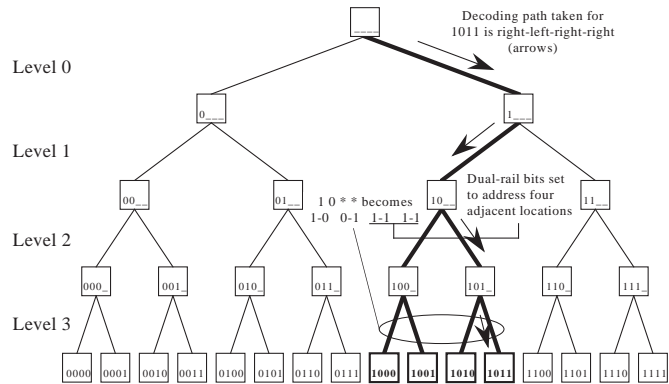
We introduce the use of *dual-rail logic*, in which a logical 0 is represented by the spatial pair 0-1 (dark-light)



**Figure 2:** Model of parallel accessing structure.

and a logical 1 is represented by the spatial pair 1-0 (light-dark). The 1011 single-rail address becomes 1-0 0-1 1-0 1-0 in dual-rail logic. If we allow both sides of a dual-rail decoder tree to be traversed simultaneously, by forcing both bits of a dual-rail bit-pair to be 1, then the size of the accessed data object doubles. This is an important aspect of the memory addressing scheme.

Figure 3 shows a decoding path when both bits of the two rightmost dual-rail bit-pairs are set to 1. The four words at locations 1000-1011 are accessed in parallel. This addressing scheme thus offers a potential for accessing a parallel memory in a useful way, rather than simply sending a raw data block to a host processor that would then be forced to reformat it.



**Figure 3:** Dual-rail parallel decoder tree.

The IPC Algorithm shown below decomposes an arbitrarily shaped region into the minimum number of subregions that are accessed in succession, making use of the dual-rail addressing scheme. The IPC Algorithm starts by assuming a block of words to be accessed fits exactly into a power of two subtree, taking into account the positions of the boundaries. It then successively decomposes the block until the sub-blocks fit within the boundaries. Adjacency is considered for Cartesian space only, as shown in Figure 3 for a four-word block, and not for Hamming space which can be more efficient.

### IPC Algorithm

```

The ADDRESS port holds the Source address encoded in dual-rail logic.
The DATA-IN port holds the Target address encoded in dual-rail logic.
The SIZE port holds the dual-rail size of the block of memory to copy.
Function FillsSubtree(Address, Size) returns TRUE if the block at Address exactly fills a power-of-2 subtree on a Size boundary; returns FALSE otherwise.
Temp ← Size
LOOP: If FillsSubtree(Source, Temp) AND FillsSubtree(Target, Temp)
    Then {
        // Read a block from the memory using a
        Parallel_Read(Source XOR Temp/2); // "disallowed" dual-rail address.
        Parallel_Write(Target XOR Temp/2); // Write the block back to Target.
        Source ← Source + Temp + 1;
        Target ← Target + Temp + 1;
        Size ← Size - (Temp + 1); Temp ← Size;
    } Else Temp ← Temp / 2;
If Size ≠ 0 then GOTO LOOP; Else DONE.

```

The work reported here was jointly supported by AFOSR and NSF on grant ECS 93-12625.

### 4. References

- [1] Murdocca, M. J., A. Huang, J. Jahns, and N. Streibl, "Optical Design of Programmable Logic Arrays," *Applied Optics*, **27**, pp. 1651-1660, (May 1, 1988).
- [2] Malcuit, M. S. and T. W. Stone, "Optically Switched Volume Holographic Elements," submitted to *Optics Letters*.
- [3] Pinkston, T. M., "Design Considerations for Optical Interconnects in Scalable Parallel Computers," *IPPS '94: Massively Par. Proc. Using Optical Interconnects*, Cancun, IEEE Comp. Soc. Press, (May 1994).