World Academy of Science, Engineering and Technology
International Journal of Computer, Electrical, Automation, Control and Information Engineering Vol:11, No:3, 2017

Parallel Vector Processing Using Multi Level Orbital DATA

Nagi Mekhiel

***Abstract*—**Many applications use vector operations by applying single instruction to multiple data that map to different locations in conventional memory. Transferring data from memory is limited by access latency and bandwidth affecting the performance gain of vector processing. We present a memory system that makes all of its content available to processors in time so that processors need not to access the memory, we force each location to be available to all processors at a specific time. The data move in different orbits to become available to other processors in higher orbits at different time. We use this memory to apply parallel vector operations to data streams at first orbit level. Data processed in the first level move to upper orbit one data element at a time, allowing a processor in that orbit to apply another vector operation to deal with serial code limitations inherited in all parallel applications and interleaved it with lower level vector operations.

***Keywords*—**Memory organization, parallel processors, serial code, vector processing.

I. Introduction

MULTIPROCESSOR system has been used to improve performance of parallel applications [1], [2]. One popular architecture is the multiprocessor with shared memory. The sharing of one memory and the synchronization overhead for managing the shared data limit the performance gain and scalability of this system [3].

Vector processing has been used in many systems with SIMD implementation using array of processors to process data at same time. The data must come from a memory system and depends on its access latency and bandwidth. This limit the performance of vector processing that must wait for data to be available and the whole vector to be delivered before it can apply a single instruction to process it. This requires fast memory like cache and also a fast network or wide bus to satisfy bandwidth requirements which adds to system complexity and cost.

The conventional vector processing is also limited in dealing with serial code that usually needed to collect the results after applying SIMD. In matrix multiplication data elements in a row is multiplied by corresponding column data elements, this could use SIMD, however the results must be added one at a time in serial fashion to get the resultant data element. Therefore a second level of vector processing that applies one operation to multiple data elements in serial fashion is needed. This is different from the conventional vector operation that adds two vectors and produces a third vector that has each element corresponds to addition of one element in first vector and a second element in second vector.

Nagi Mekhiel is with the Department of Electrical and Computer Engineering, Ryerson University, Toronto, Ontario, Canada (e-mail: nmekhiel@ee.ryerson.ca).

II. Background

The performance of processor depends on its memory system [4]. New processors use large multi-level cache system to close the huge speed gap between processor and main memory DRAM [5]. According to Amdahl's law the performance improvements of advanced processor is limited by the slow portion that cannot be improved in accessing memory and the serial code.

It is essential to improve memory performance to take full advantage of vector operations that are widely used in many applications. Special mapping of vector data elements to one cache line, with data elements that are scattered in memory, has been proposed in [6] to improve vector processing. However this mapping requires data to be first transferred from slow memory to a cache line. The mapping of data to different memory locations makes accessing them slower as they cannot be transferred in a burst mode or use a block transfer of a sequential locations. This method also cannot support parallel vector operation due to cache conflicts and also cannot apply a vector operation to collect data to replace a serial code by a single instruction.

III. Concept of Orbital Data

A shared resource will become not shared if we offer it or make it available to all processors. Processors then need not to arbitrate and wait to get the data from the shared resource. Rather than having one processor to access one location in memory at a time, we allow all locations to be available to all processors all the time. The processor waits for the memory to provide its data or instructions at a specific time [7], [8].

We propose making the contents of memory available to all processors using a shared bus or orbit. Every location in the memory is guaranteed to be delivered to the bus and all processors have all the requested data available after waiting for a time that does not exceed the time of transferring the memory section out in the bus.

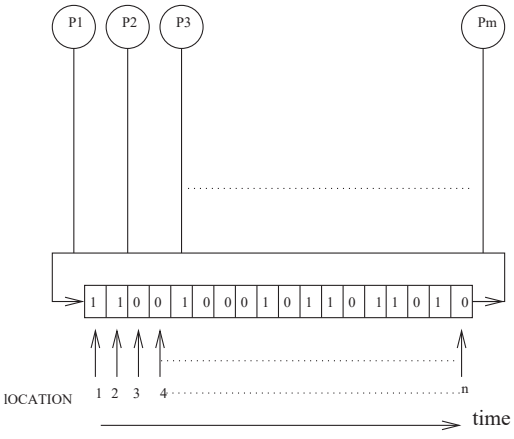
The new memory makes its content spins continuously around the bus, so that any processor can access the data at a specific time. This memory uses time as an address to access it and is considered as a Time Addressable Memory "TAM" [7], [8].

It consists of the following features:-

- It supplies all contents of memory to all processors regardless if the processor needs it or not.
- It is very fast because it is accessed sequentially in time and does not have to use a random access mode as in conventional memory.

- The access of each location is known ahead of time, so there is no waiting time for precharge, or decoding, therefore access time is hidden.
- It uses DRAM or SRAM technology with very simple organization.

Fig. 1 shows a block diagram for this concept. The contents of memory spin around all the time in a fast speed and is accessed by any processor.



TIME ADDRESSABLE MEMORY "TAM"

Fig. 1 Concept of Orbital Data

IV. Orbital Data Memory Organization and Operation

Orbital Data could be implemented in any technology. It also could be implemented using a different organization than the known array structure to optimize the cost of implementation. We assume using the known DRAM technology to implement it and the contents of DRAM is accessed sequentially in a serial fashion without the need for address or decoders. Because the time to access each location is known ahead of time, the activation time overhead is hidden and overlapped among the different banks. The access time for each location will be only the time of accessing the sense amplifier and is limited only by the external bus speed. There is no address lines, and it could use the address lines to transfer more data which will double Bandwidth. Fig. 2 shows the organization of Orbital data memory. It consists of the following components:

- DRAM Array: Organized as an array of N rows by M columns. Each cell is located at the intersection of a specific row and a specific column. The storage cell consists of one capacitor and a one transistor as in any DRAM structure.
- Row Shift Register: Has N outputs, each is responsible to activate the corresponding row similar to the row decoder outputs in DRAM. The shift register consists of N D-type Flip Flop and only one has 1 that corresponds to current active row. Each row stays active for the time to access all columns in it. Row Shift Register shifts the 1 to access the next row.
- Column Shift Register: Has M outputs, each corresponds to a column selection that allows the flow of data similar to the column decoder outputs in DRAM. It also uses D-type Flip Flops and only one has 1 that corresponds to accessed column. Column Shift Register shifts the 1 to access next column.
- Sense Amplifiers: Are used to access the data from the input output DO/DI signal as in DRAM. The direction of data flow is controlled by a /WE signal as in any DRAM.

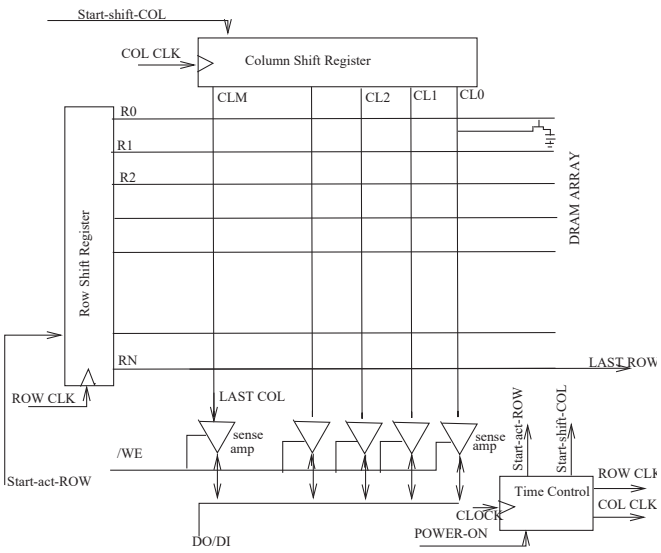


Fig. 2 Orbital Data Memory Organization

V. Orbital Data Memory System

Fig. 3 shows the basic memory system that uses multiple of chips to expand the memory. The basic system consists of N number of orbital memory chips connected in serial fashion such that the first location starts from first location of first chip and the last location is from the last location of last chip. After the access of the last location of the last chip, the access of first location of first chip occurs as shown in the block diagram.

All memory chips are connected to same CLOCK signal for synchronous design. All Do/Di signals are connected to the same bus signals as in any conventional memory expansion. The /WE signal is connected to /WE in all the chips.

The expansion of the memory system is very simple because it needs only one signal to be connected to the next memory chip. The LAST ROW signal from one chip is connected to the Start-act-ROW input of the next chip. Conventional memory expansion needs to decode some address lines to access the extra memory. Decoder adds to system complexity and delay.

If the DRAM single chip is organized as N Row by M Column, and the memory system consists of K chip, then total number of memory storage= NxMxK locations. It will take NxMxK cycles to access the full memory, and accessing the memory repeats every NxMxK cycles in a Cyclic serial fashion. The cycle time for is much faster than accessing conventional memory as it does not have a decoder and it hides Row and Column activations time.

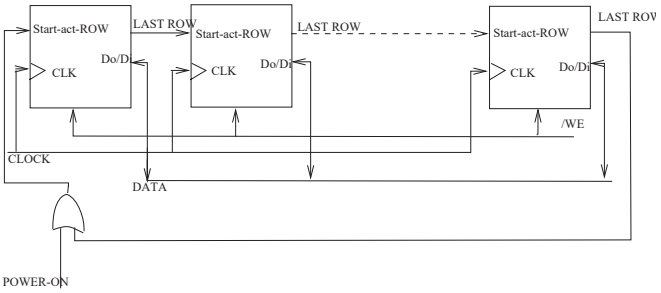


Fig. 3 Orbital Data Memory Expansion for Basic System

A. Orbital Data Memory with Multiplexer/De-Multiplexer

Fig. 4 shows the orbital memory organization using a multiplexer and De-multiplexer. The select bank address selects the specific bank to be accessed among the multiple banks. It allows output data Do of the specific bank to pass through the multiplexer to the memory bus. It also allows the input data Di from the memory bus to be passed through the De-multiplexer to be written to the selected bank.

This organization supports Out-Of-Order access patterns as the select address determines the accessed bank in any order. It also supports In Order Access patterns by using a counter and a mode signal, to access banks in order through the multiplexer or De-multiplexer.

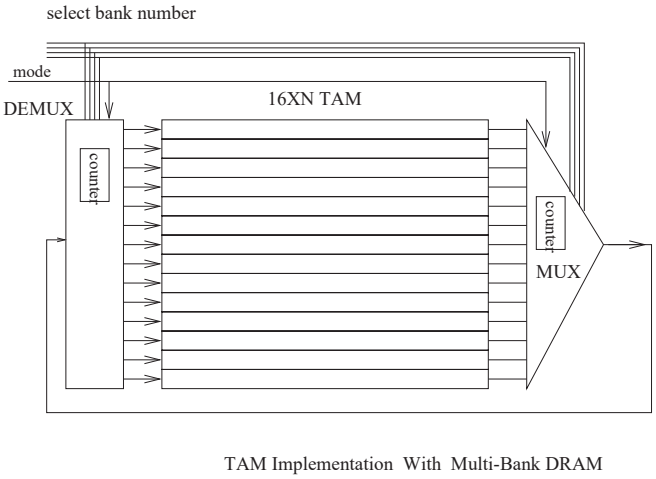


Fig. 4 Orbital Data Memory with Multiplexer/De-Multiplexer

B. Parallel Orbital Data Memory

It is possible to design a parallel Orbital memory system to access data from more than one bank at the same time. The advantage of this design is that it allows accessing data from different banks simultaneously. In a system with more than one processor, a number of processors could share one portion of memory while others access different portion of memory independent of each other and each system could have a different cycle time. The cycle time is the time it takes to access the memory section in a serial fashion until it repeats.

Fig. 5 shows a block diagram for parallel Orbital Data Memory using multiplexers/De-multiplexer. The core memory consists of multiple banks, two multiplexers and two De-multiplexers. Each multiplexer selects one bank from the memory to deliver its data to a bus. MUX1 has its output connected to DO1/DI1. MUX2 has its output connected to DO2/DI2,. The SELECT BANK1 signal selects one bank for MUX1 and SELECT BANK2 selects one bank for MUX2. The DO1/DI1 is also connected to DEMUX1 to supply DI1 to the selected bank based on SELECT BANK1 for write operations. DO2/DI2 is connected to input of DEMUX2 to supply DI2 to the second selected bank based on SELECT BANK2 signals.

C. Multi-Level Orbital Data Memory

Multi-Level allows data to be accessed from different sections of memory at different cycle time. A memory section has a number of memory locations accessed in a serial or sequential order (mapped in a linear time order). Cycle time is the time it takes to access a section of sequential accessed locations until it accessed again in a cyclic fashion.

Fig. 6 shows the concept of Multi-Level Orbital Data Memory. The whole memory spins on ORBIT0, which has the longest cycle time. Each memory location is accessed and is available to the outside bus for one bus cycle. Other sections of memory are rotating their contents at the same time in a cyclic fashion each with different cycle time. ORBIT1 has a portion of memory spinning at a faster cycle time because it contains smaller number of memory locations. ORBIT2 has the smallest number of memory locations and spins at the fastest cycle time. When the contents of memory ORBIT0 needs to be in ORBIT1 (because ORBIT1 is part of whole memory), both are aligned and become one section that belongs to ORBIT0. There is no extra memory storage for ORBIT1 or ORBIT2, they take portions of whole memory that spins at higher speed. This is because the whole memory is divided to an integer number of sections for ORBIT1 and ORBIT2.

D. Implementation of Multi-Level Orbital Data Memory

Fig. 7 shows the implementation of a multi-level Orbital Data Memory. Memory is divided to banks or sections and each section could be designed as the basic organization given above. Each section rotates its content around a special bus shown for BNK0, BNKM, BNKN. The bus for each bank has all locations of its section continuously spinning at cycle time equal number of locations in the bank multiplied by bus clock time. It combines number of banks and makes their contents available one after another in sequence. The bus output of MUX1 is also connected as input for De-MUX1. If MUX1 is selecting BNK3, then data out from BNK3 is delivered to MUX1 bus, and rerouted through De-MUX1 to be available for BNK3, while BNK1 own bus data output is connected to BNK1 to deliver data input at the same time in parallel.

MUX0 is used to access the whole memory as level 0, by selecting ,in a sequential order, the outputs from all MUX1 of memory. The bus of MUX0 is also connected to the input of De-MUX0 and rerouted to the accessed bank by one of DE-MUX1. When one bank is accessed by the bus of MUX0, this bank will only be controlled by this bus, and the other buses from MUX1 or internal bank bus are not used.

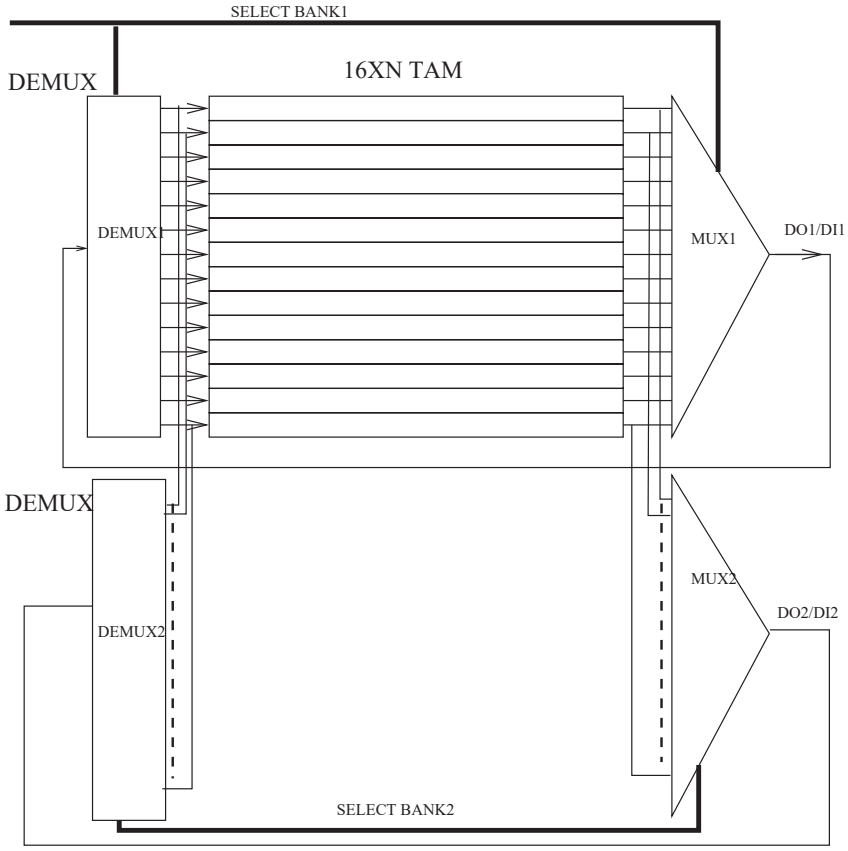


Fig. 5 Parallel Orbital Data Memory

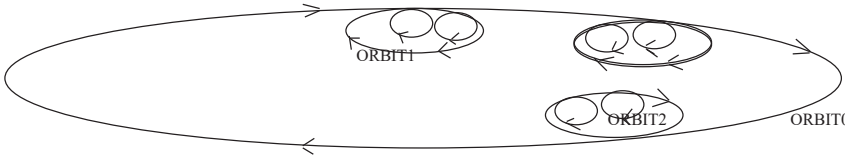


Fig. 6 Concept of Multi-Level Orbital Data Memory

VI. Multiprocessor Using Multi-Level Orbital Data Memory for Parallel Vector Operations

Fig. 8 shows multiprocessor organization using Multi-Level Orbital memory described above. Each group of multiprocessors are connected to one multiplexer/De-multiplexer to access sections of memory based on the selected memory section. Other groups of multiprocessors are similarly connected to other levels of Orbital memory system. This provides a multi-level parallelism achieved by Multi-Level orbital organization. Each multiprocessor group shares one portion of memory at a specific time without the need to exclude the other groups of multiprocessors. Third level parallelism is obtained among the multiprocessor group sharing one multiplexer/De-multiplexer MUX0, DE-MUX0. Second multiprocessor group shares MUX1/DE-MUX1 accessing a smaller portion of memory that spins at higher speed. First level of processors are connected directly to memory bank or section.

A. First Level Parallel Vector Operations in Orbital Data Memory

Fig. 9 shows a block diagram of vector operations in orbital data memory. The content of each memory location is placed in the bus for a single cycle, then all processing elements read these locations without arbitration or waiting. The Vector with N elements is stored in N sequential memory locations, and could be read or written to by N parallel processing elements in N cycles. The load and store of long vectors in conventional vector processor requires complicated and highly interleaved bank memory with careful mapping of data in to different banks. Even if vector processor uses a cache memory, it also has limitations in supporting vector load and store because the vector length causes high miss rate and costly transfer time.

The following are some of the vector operations supported and shown in Fig. 9:

- LDV R1, 0(R7): This is a load instruction for a vector with N elements to N processors. Processor P1 transfers first element of vector to its register R1 at cycle number

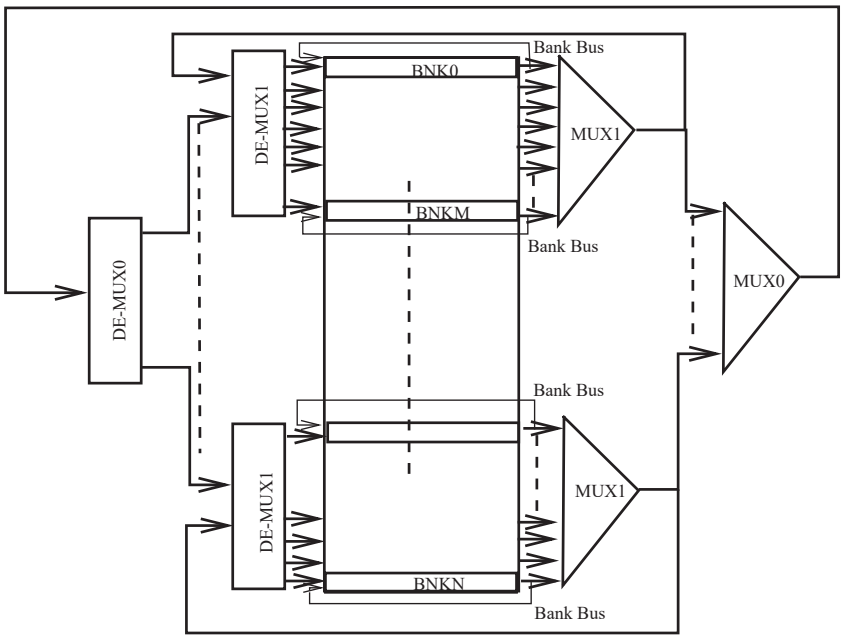


Fig. 7 Multi-Level Orbital Data Memory Implementation

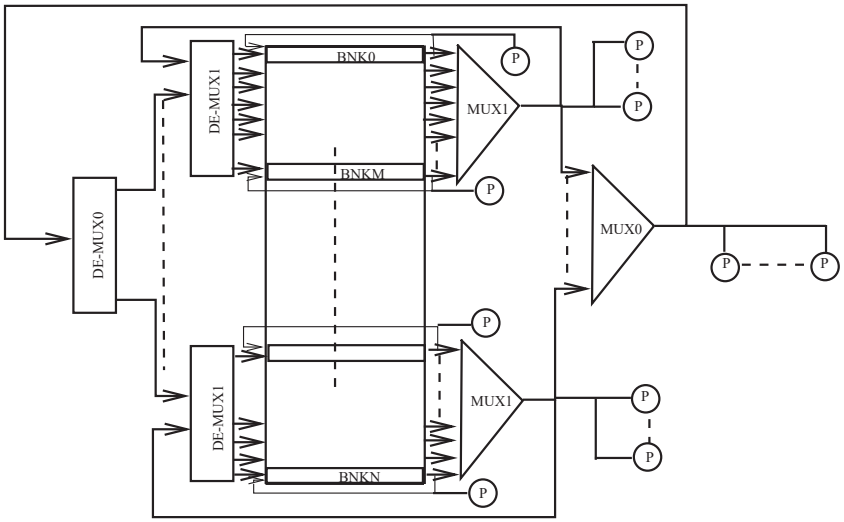


Fig. 8 Multiprocessor using Multi-Level Orbital Data Memory

equal to the content of R7. Processor P2 transfers second element of the same vector to its R1 register at cycle equal to R7 plus one. Processor PN transfers the last element of the vector to its R1 register at cycle equal R7 plus N. It takes N cycles to transfer the full vector to processors registers.

- LDV R2, 0(R8): This instruction transfers the second vector to N processors register R2 in N cycles as explained above.
- ADD R5, R1, R2: Every processor adds one element of first vector to one element of the second vector only in one cycle in parallel using SIMD. It is important to note that a conventional vector processor with pipelined function unit will take N cycle to add the two vectors.
- SDV 0(R9), R5: This instruction stores the results of the vector elements obtained from N processor SIMD ADD operation above in N locations in N cycles. P1 stores it at location R9 in time, P2 stores its R5 in R9 plus 1, .. processor N stores its R5 at R9 plus N.

The following are the advantages of using orbital data memory for vector operations in the same level:-

- Data transfers from memory to processor or processor to memory is very efficient. There is no need for bank interleaving, and there is no bank conflicts as in vector processor.
- Conventional cache suffers from high cost of transfer time and high miss rate for using long vector in cache [8] when data is scattered in main memory.
- Other vector operations could be performed between processors sharing one memory bank in the same orbit

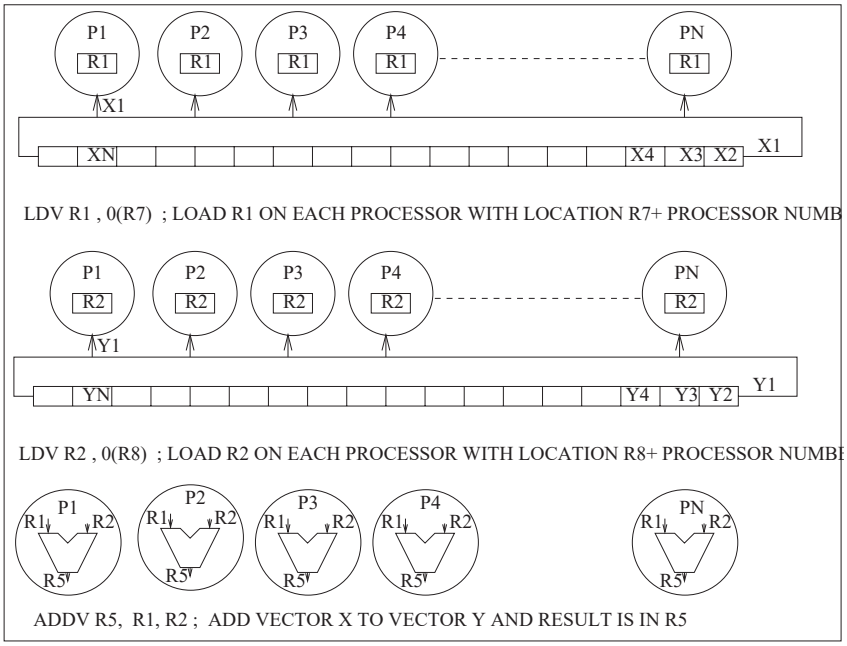


Fig. 9 Vector Operations using Orbital Data Memory

in parallel with other processors sharing another orbit or bank in the same level, both groups are connected to same multiplexer/de-multiplexer for that specific level.

B. Second Level Vector Operation in Upper Level Orbit

While each processor P1..PN is storing the results of ADD in sequential memory locations using SDV in the above example, the multiplexer selects these contents to appear at the same time in a higher level orbit to a processor P at higher orbit as shown in Fig. 8. Processor P applies a single instruction to collect the results from lower level for example, an ADD will add all these results together. This represents another level of vector operation completely done in parallel while storing these results to their locations in lower memory orbit level and completely overlapped at the same time while executing the SDV instruction. Conventional processors will use serial code to add these results one at a time using N instructions that need to read N vector elements, then transfer them to vector processor.

Another group of processors could in parallel have a similar vector operations at same time and pass their results through a second multiplexer to a higher orbit for a processor to apply a second level parallel vector operation. A third level orbit could process data from the second level orbits and apply a third level vector operation to a huge amount of data by passing these data through another higher level multiplexer that could be interleaved with vector operations of first, second level operations to occur at same time.

This system can successfully apply parallel vector operations to multiple vectors of data and at same time interleave them and overlap them with other vector operation in a higher orbit level.

VII. Conclusions

Orbital data memory allows the continuous transfer of data in different orbits with different cycle time to processors. It is simple and fast making it suitable to implement multi level and parallel vector operations for streams of data that are provided to processors at fast rate. The vector operations can be offered in parallel by different processors in each orbit and at the same time could be overlapped to support serial code that executes as single instruction for stream of data, therefore dealing with a fundamental limitations in parallel computing.

References

[1] J. Hennessy, D. A. Patterson Computer Architecture: A Quantitative Approach Morgan Kaufmann Publishers, Inc, San Francisco, CA, 1996.
[2] Agarwal, B. H. Lim, D. Kranz and J. Kubiatowicz, April: A processor architecture for Multiprocessing, in Proceedings of the 17th Annual International Symposium on Computer Architectures, pages 104-114, May 1990.
[3] D. Burger, J. R. Goodman, and A. Kagi, Memory Bandwidth of Future Microprocessors, In Proc. 23rd Annual Int. Symp. on Computer Architecture, (ISCA'96), pp.78-89, Philadelphia, PA, 1996.
[4] Saulsbury, A.; Nowatzyk, A. Missing the memory wall: the case for processor memory integration, ISCA96: The 23rd Annual International Conference on Computer Architecture, Philadelphia, PA, USA, 22-24 May 1996 p.90-101.
[5] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Camean, A. Kyker, and P. Roussel, The microarchitecture of the Pentium 4 processor, Intel Technology Journal, 5(1), pages 1-133, Feb. 2001.
[6] Eichenberger et al., International Business Machines Corporation, Armonk, NY (US) Vector Loads With Multiple Vector Elements From a Same Cache Line in a Scattered Load Operation, US 8,904,153 B2 Dec. 2, 2014.
[7] Mekhiel, Data processing with time-based memory access, US 8914612B2 Dec 16, 2014.
[8] Introducing TAM: "Time Based Access Memory", Nagi Mekhiel, IEEE Access journal, March 30, 2016. P. 1061-1073 Volume 4.