MONASH University

[Template for Applied Session submission]

Applied Session Week {Week 6}

Student ID	Student Name	Student Email
32837933	Chua Sheng Xin	schu0077@student.monash .edu

Tasks

{Answers
Question 1
A) If N=7,

PE0	PE1	PE2	PE3
A(1)	A(2)	A(3)	A(4)
B(1)	B(2)	B(3)	B(4)
C(1)	C(2)	C(3)	C(4)
A(5)	A(6)	A(7)	
B(5)	B(6)	B(7)	
C(5)	C(6)	C(7)	

In a SIMD machine with 4 PEs, the array elements are typically distributed across the PEs. For N = 7, PE0 will handle elements A(N), B(N) and C(N) when N=1,5. PE1 will handle elements A(N), B(N) and C(N) when N=2,6. PE2 will handle elements A(N), B(N) and C(N) when N=3,7. PE3 will handle elements A(N), B(N) and C(N) when N=4.

The loop will need to run twice.
Iteration 1: PEs 0 to 3 will process elements A(1) + B(1) + C(1), A(2) + B(2) + C(2), A(3) + B(3) + C(3), A(4) + B(4) + C(4) respectively.
Iteration 2: PEs 0 to 2 will process elements A(5) + B(5) + C(5), A(6) + B(6) + C(6), A(7) + B(7) + C(7).

In the second iteration, PE4 will be disabled because there are only 7 elements, meaning the last PE does not have any data to process.
B) The overheads of opcode fetches, decodes, and operand loads are reduced by mechanisms of opcode fetching, decoding and loading. The vector instruction ADD N, A, B, C adds two vectors B and C, and stores the result in vector A and the opcode fetch, decode, and operand load happen for every individual operation in scalar processing. For vector processing, the opcode is fetched and decoded only once for the entire vector operation to reduce overhead. Instead of loading individual operands for each element operation, operands for a vector operation are loaded in blocks. This bulk loading reduces the total number of memory accesses and leads to less overhead.

Chaining in vector processing allows the output of one vector operation to be directly used as an input to another vector operation without waiting for the entire operation to complete. For example, in a sequence of operations where vector A is produced by an ADD operation and then immediately used in a MUL operation, chaining enables the elements of vector A to be used in the MUL operation as soon as they are available from the ADD, thus improving performance by reducing the idle time of processing elements.

Question 2

A) Distributed Memory MIMD scales well by adding more nodes, while Shared Memory MIMD can face scalability challenges due to memory contention and synchronization issues. Distributed Memory MIMD requires message passing with associated overhead, while Shared Memory MIMD benefits from faster, more direct communication through shared memory. The two advantages & two disadvantages of Distributed Memory MIMD are:

Scalability: Distributed Memory MIMD systems can scale well with the number of processors or nodes. Since each node has its own memory, adding more nodes typically involves adding more memory and processing power, allowing for large-scale parallel computations.

Synchronization:Distributed Memory MIMD does not require sophisticated synchronization features such as monitors and semaphores. Message passing serves the dual purpose of sending the data and providing synchronization.

Load Balancing: Distributed Memory MIMD that have multiple CPUs can experience trouble balancing the load. Some may not be heavily loaded, some may be locally loaded and some may be idle. The system depends on the type of application that is used to distribute the load.

Synchronization Failures: Message passing can lead to synchronization failures in the system including deadlock. An example is BlockingSend -> BlockingReceive and BlockingReceive -> BlockingSend.

The two advantages & two disadvantages of Shared Memory MIMD are:

Ease of Programming: Shared Memory MIMD systems often provide a simpler programming model. All processors access a common memory space, so there is no need for explicit message passing. Synchronization and data sharing can be handled using synchronization primitives such as mutexes and semaphores.

Low Latency Communication: Communication between processors is typically faster because they share the same memory space. This reduces the overhead associated with data exchange compared to distributed memory systems.

Disadvantages:

Difficult Synchronization: Managing and synchronizing access to shared memory can become complex. Issues such as cache coherence, memory consistency, and synchronization can introduce overhead and potential performance bottlenecks.

Scalability Limitations: Shared Memory MIMD systems face challenges as the number of processors increases. Memory access contention and increased complexity in managing concurrent access to shared data can limit scalability.

B) We can find the number of hops between nodes 100 and 011 using the hypercube topology. Nodes are typically represented as binary numbers. The number of hops between two nodes in a hypercube is equal to the Hamming distance between their binary identifiers. The Hamming distance is the number of bit positions at which the corresponding bits differ. For nodes 100 and 011, the number of differing bits is 3 hence the number of hops between 100 and 011 is 3. The number of hops will decrease by 1 if the hypercube is rearranged into a binary balanced tree topology. In a rearranged balanced binary tree, 100 and 011 will be at the same level because there is only 1 bit difference between them. The hop goes from 100 to 001 and then to 011 or vice versa. The number of hops is only 2 for a balanced binary tree.

In a hypercube topology with 3 bits, there are 2^3 = 8 nodes. Thus, using only 3 bits is no longer sufficient and adding one more node requires expanding the number of bits. Node insertion in a balanced binary tree topology is simpler and does not necessarily require changing the bit representation of node IDs. New nodes are inserted in a way that maintains the tree's balance, which can be more straightforward compared to expanding the number of bits in a hypercube. A balanced binary tree can handle more nodes without changing the bit-width of node identifiers, while a hypercube requires expanding the number of bits to accommodate additional nodes.

Question 3
The code may result in a deadlock because both processes are attempting to MPI_Recv before they MPI_Send. If both processes are waiting to receive data from each other, neither will reach the MPI_Send call, causing the program to hang.
}

{Screenshots}

{References}