CSA Module 3 - Notes

Computer science and engineering (University College of Engineering)

CS 08.802 COMPUTER SYSTEM ARCHITECTURE

Module 3 (cont'd..)

Multiprocessors and multicomputers - multiprocessor system interconnects, cache coherence and synchronization mechanism, three generations of multicomputers, Intel Paragon system architecture**. Multivector and SIMD computers** - vector processing principles, multivector multiprocessors, SIMD computer organizations. **Scalable, multithreaded and data flow architectures** - latency hiding techniques, principles of multithreading, scalable and multithreaded architectures, data flow and hybrid architectures.

VECTOR PROCESSING PRINCIPLES

Q13: Describe about vector instruction types?

- A vector is a set of scalar data items, all of the same type, stored in memory.
- Usually, the vector elements are ordered to have a fixed addressing increment between successive elements called the stride.
- A vector processor is an ensemble of hardware resources, including vector registers, functional pipelines, processing elements, and register counters, for performing vector operations.
- Vector processing occurs when arithmetic or logical operations are applied to vectors.
- The conversion from scalar processing to vector code is called vectorization.
- Vector processing is faster and more efficient than scalar processing. It reduces software overhead in the maintenance of looping control, reduces memory-acess conflicts.etc..
- A compiler capable of vectorization is called vectorizing compiler or vectorizer.

Vector instructions types

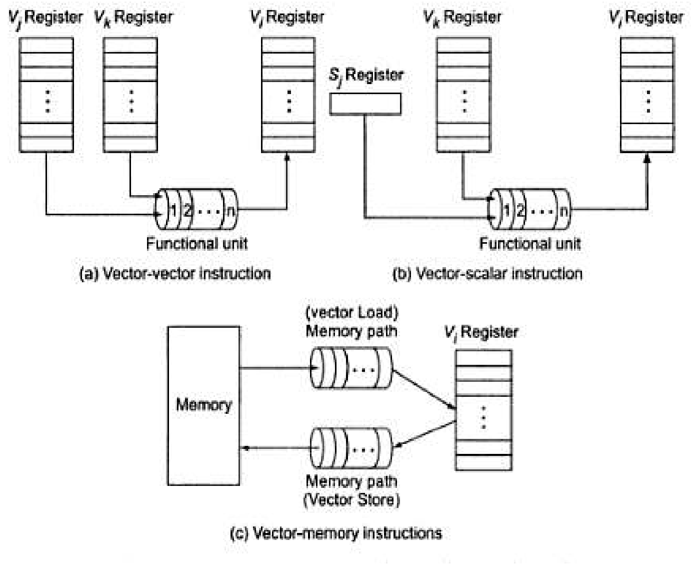
1. Vector-vector instructions :One or two vector operands are fetched form the respective vector registers, enter through a functional pipeline unit, and produce result in another vector register. These instructions are defined by following mapping:

f1: $V_j \text{ --> } V_j$ i (e.g. sin(V_1= V_2)
f2: $V_j \text{ x } V_k \text{ --> } V_i$ (e.g. $V_1 + V_2 = V_3$)

2. Vector-scalar instructions: f3: $s \text{ x } V_k \text{ --> } V_i$ .An example is as scalar product $s \text{ x } V_1 = V_2$, in which the elements of V_1 are each multiplied by a scalar s to produce vector V_2 of equal length.

3. vector-memory instructions: Store-load of vector registers

f4: M --> V (e.g. Vector Load)
f5: V --> M (e.g. Vector Store)



4. Vector reduction instructions : These corresponds to the following mapping:

f6: Vi --> s
f7: Vi x Vj --> s

Examples of f6 include finding the maximum, minimum, sum and mean value of all elements in a vector. A good example of f7 is the dot product, which performs

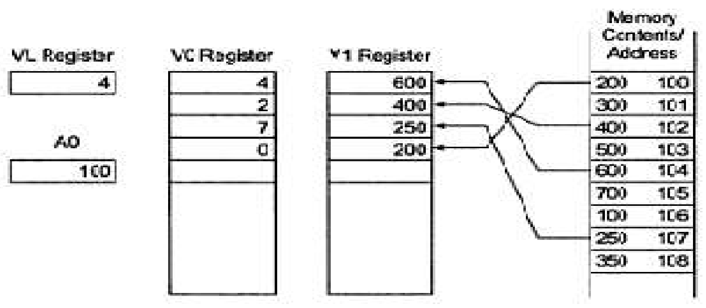
$$S= \sum_{i=1}^{n} ai \times bi$$

from two vectors A=(a_i) and B =(b_i)

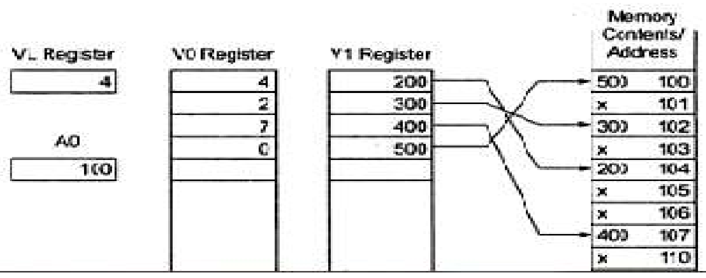
5. Gather and scatter instructions : Two instruction registers are used to gather or scatter vector elements randomly through the memory corresponding to the following mapping:

f8: M --> V_1 x V_0 (e.g. gather)
f9: V_1 x V_0 --> M (e.g. scatter)

Gather is an operation that fetches from memory the nonzero elements of a sparse vector using indices that themselves are indexed. Scatter stores into memory a vector in a sparse vector whose nonzero entries are indexed. The vector register v1 contains the data, and the vector register V0 is used a s an index to gather or scatter data from or to random memory locations .VL indicates the no: of elements being transferred. And AO contains the base address. The effective memory addresses are obtained by adding the base address to the index.



(a) Gather instruction



6. Masking instructions :The Mask vector Vm is used to compress or to expand a vector V_0 to a shorter or longer index vector (bit per index correspondence).

F10: V_0 x Vm --> V_1

Q14: Describe vector access-memory schemes?

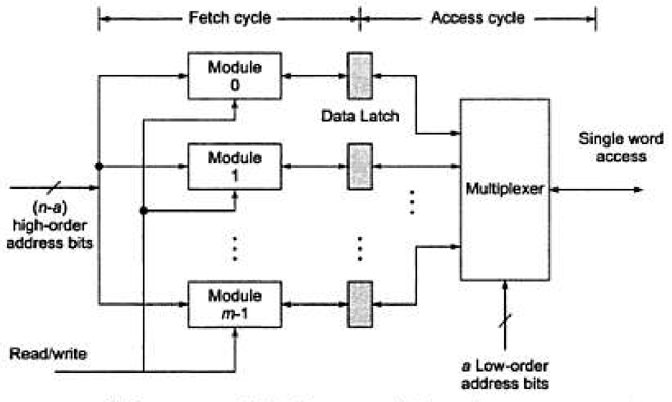
- Vector operands may have arbitrary length.
- Vector elements are not necessarily stored in contiguous memory locations.
- To access a vector a memory, one must specify its base, stride, and length.
- Since each vector register has fixed length, only a segment of the vector can be loaded into a vector register.
- Vector operands should be stored in memory to allow pipelined and parallel access.

C-Access memory organization

- The m-way low-order interleaved memory structure, allows m words to be accessed concurrently and overlapped.
- The low- order a bits select the modules, and the high-order b bits select the word within the each module, where $m= 2^a$, and a +b is the address length.
- C- access will yield the maximum throughput of m words/ memory cycle .

S-Access memory organization

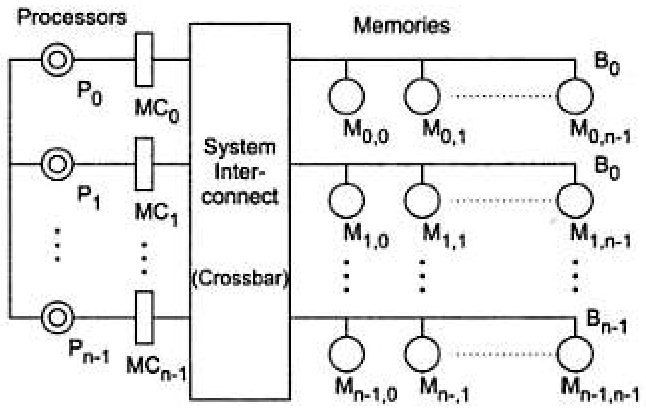
- All memory modules are accessed simultaneously in a synchronized manner. The high order (n-a) address bits select the same offset from the each module. The low order address bits are used to multiplex the m words out of buffers.



(a) S-access organization for an *m*-way interleaved memory

C/S-Access memory organization: A memory organization in which the c-access and S- access are combined is called C/S –access.

- Here n access buses are used with m interleaved memory modules attached to each bus. The m modules on each bus are m-way interleaved to allow C-access. The n buses operate in parallel to allow S –access .In each memory cycle, a most mn words are fetched if the n buses are fully used with pipelined memory accesses.



The C/S memory organization with *m* = *n*. (Courtesy of D.K. Panda, 1990)

- This scheme is suitable for use in multiprocessor configurations.

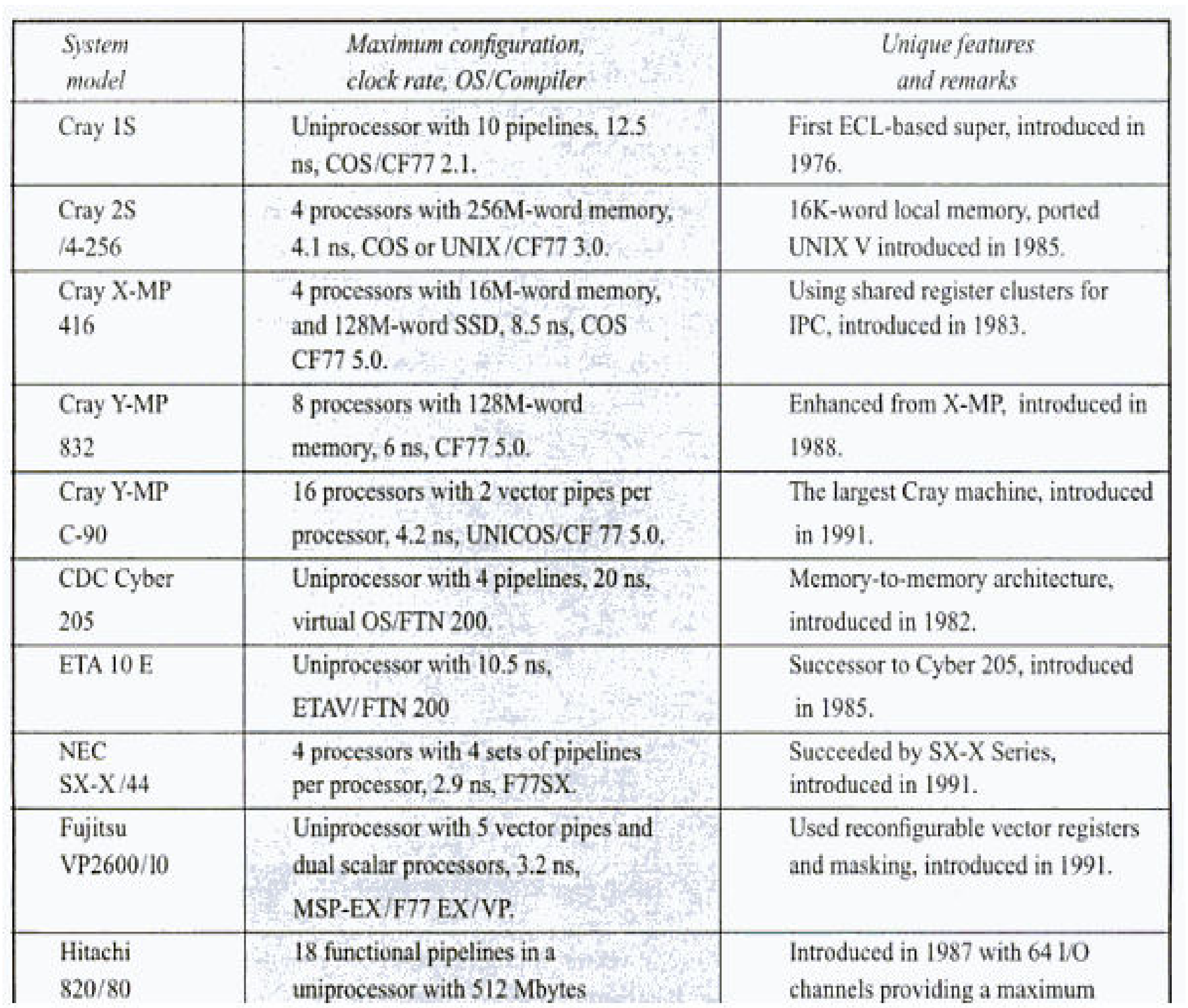
Q15: Mention about early supercomputers used for vector processing?

Here it introduces five early supercomputer families used for vector processing, including the Cray Research series, the CDC/ETA series, the fujitsu VP series, the NEC-SX series and the Hitachi 820 series. The table shows a summary of early computers.

The Cray Research series: High degree of pipelining and vector processing are the major features of these machines. Register-register based architecture is used in this series.

The cyber/ETA series: memory-memory based architecture with longer vector instructions containing memory addresses are used in this series

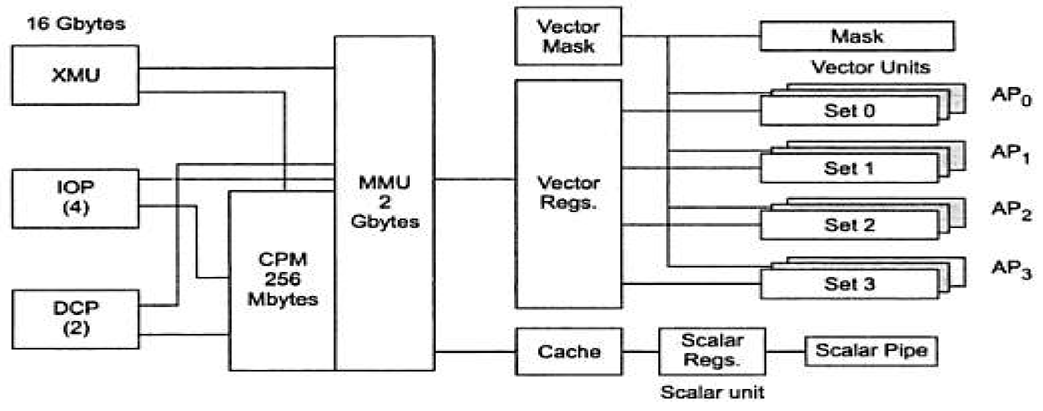
The Japanese super computers:It includes Fujitsu series and NEC_SX series machines.Shared communication registers and reconfigurable vector registers were special features in this machines.



System model	*Maximum configuration, clock rate, OS/Compiler*	*Unique features and remarks*
Cray 1S	Uniprocessor with 10 pipelines, 12.5 ns, COS/CF77 2.1.	First ECL-based super, introduced in 1976.
Cray 2S /4-256	4 processors with 256M-word memory, 4.1 ns, COS or UNIX/CF77 3.0.	16K-word local memory, ported UNIX V introduced in 1985.
Cray X-MP 416	4 processors with 16M-word memory, and 128M-word SSD, 8.5 ns, COS CF77 5.0.	Using shared register clusters for IPC, introduced in 1983.
Cray Y-MP 832	8 processors with 128M-word memory, 6 ns, CF77 5.0.	Enhanced from X-MP, introduced in 1988.
Cray Y-MP C-90	16 processors with 2 vector pipes per processor, 4.2 ns, UNICOS/CF 77 5.0.	The largest Cray machine, introduced in 1991.
CDC Cyber 205	Uniprocessor with 4 pipelines, 20 ns, virtual OS/FTN 200.	Memory-to-memory architecture, introduced in 1982.
ETA 10 E	Uniprocessor with 10.5 ns, ETAV/FTN 200	Successor to Cyber 205, introduced in 1985.
NEC SX-X /44	4 processors with 4 sets of pipelines per processor, 2.9 ns, F77SX.	Succeeded by SX-X Series, introduced in 1991.
Fujitsu VP2600/10	Uniprocessor with 5 vector pipes and dual scalar processors, 3.2 ns, MSP-EX/F77 EX/VP.	Used reconfigurable vector registers and masking, introduced in 1991.
Hitachi 820/80	18 functional pipelines in a uniprocessor with 512 Mbytes	Introduced in 1987 with 64 I/O channels providing a maximum

The NEC-SX-X 44:

- There were four arithmetic processors communicating through either the shared registers or via the shared memory.
- There were four sets of vector pipelines per processor, each set consisting of two add/ shift and two multiply/logical pipelines.
- Besides the vector unit, a high speed scalar unit employed RISC architecture with 128 registers also implemented in it.
- The main memory was 1024 memory interleaved. A maximum of four I/O processors could be configured to accommodate a 1 Gbyte/s data transfer per I/O processor.



Captions:
XMU: Extended memory unit
IOP: I/O processors (4)
DCP: Data central processors (2)
AP: Arithmetic processors (4)
MMU: Main memory unit
CPM: Data central processor memory
Each set consists of 4 pipelines for add/shift and multiply/logical vector operations

Fig. 8.5 The NEC SX-X 44 vector supercomputer architecture (Courtesy of NEC, 1991)

Relative vector/scalar performance: Let r be the vector/scalar speed ratio and f the vectorization ratio. The relative performance can be defined as

$$P = \frac{1}{(1-f) + f/r} = \frac{r}{(1-f)\,r + f}$$

This relative performance indicates the speed up performance of vector processing over scalar processing .The vectorization ratio f reflects the percentage of code in a user program which is vectorized.

If $f \to 0$, $P \to 1$ and $f \to 1$, $P \to r$.

MULTIVECTOR MULTIPROCESSORS

Q16: Describe about the design goals (performance-directed design rules) for developing multivector multiprocessors(supercomputers)?

The main architecture Design Goals are:

- Maintaining a good vector/scalar performance balance.
- Supporting scalability with an increasing number of processors.
- Increasing memory system capacity and performance
- Providing high-performance I/O and easy-access network.

Balanced vector/scalar ratio: In a supercomputer, separate hardware resources with different speeds are used to concurrent scalar and vector operations .Scalar processing is used for general purpose applications, vector processing is used for scientific and engineering applications. But both the computations should be balanced. The vector balance point is defined as the percentage of vector code in a program required to achieve equal utilization of vector and scalar hardware.

Vector/scalar performance: For maintaining good Vector/scalar performance, use high clock rate, better compiler and the optimization support .Vector performance can be enhanced with replicated functional unit pipelines in each processor.

I/O and networking performance: The I/O is defined as the transfer of data between processor/memory and peripherals or a network. I/O performance can be increased by using flexible I/O processor that could do complex processing or a simple front –end processor controlling high speed channels with most of the I/o management being done by the mainframe OS.

Memory demand: A large scale memory system must provide a low latency for scalar processing, a high bandwidth for vector processing and a large size for grand challenge problems and high throughput. For achieving this goals, an efficient memory hierarchy should be used. A typical hierarchy consists of data files or disks, extended memory in dynamic RAMs , a fast shared memory in static RAMs and a cache local memory using RAM on arrays.

Supporting scalability: Multiprocessors should be designed to support scalability, but the dominant scalability problem involves support of shared memory with an increasing number of processor and memory ports. Increased memory –access latency and interprocessor communication overhead imposes additional constraints on scalability.

Q17: Explain the supercomputers architecture's?

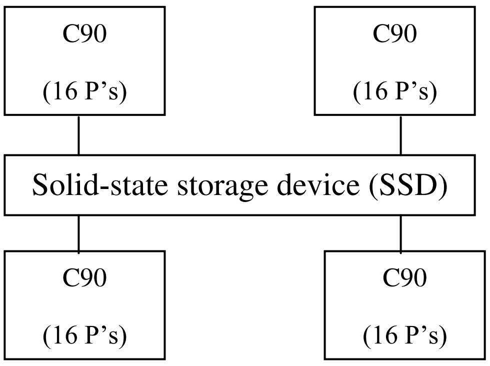
(a) Cray Y-MP 816:

- The eight CPU s of the Y-MP shared the central memory, the I/O section, the interprocessor communication section, and the real-time clock.
- The central memory was divided into 256 interleaved banks, where each bank consists of memory modules.
- A 6-ns clock period was used in this design.
- The system had inbuilt resolution hardware to minimize the delays caused by memory conflicts. To protect data, single-error correction/double – error correction logic was used in central memory and the data channels to and from the central memory.
- The CPU section consisted of 14 functional units divided into vector, scalar , address and control sections.
- Large no of address, scalar, vector, intermediate and temporary registers were used.

- The interprocessor communication section consists of the mainframe clusters of shared registers for fast synchronization purposes. Each cluster consists of shard address, shared scalar, and semaphore registers.
- The I/O section supported three types of channels with transfer rate 6Mbytes/s, 100Mbytes/s and 1Gbytes/s.

(b) The C-90 and clusters:

- The system was built with 16 CPU s, each of which is similar to Y-MP. It used shared main memory of 2 Gbytes. SSD was also available as optional secondary memory. The C-90 used the UNICOS operating system.
- Multiple C-90s could be used in a clustered configuration in order to solve large scale problems. Four C-90 clusters were connected to a group of SSDs. Each C-90 clusters was allowed to access its own shared memory. But they shared the access of SSDs. The clusters could communicate with each other through a shared semaphore unit. Only synchronization and communication information are passed through semaphore unit.
- C-90 clusters are loosely coupled.

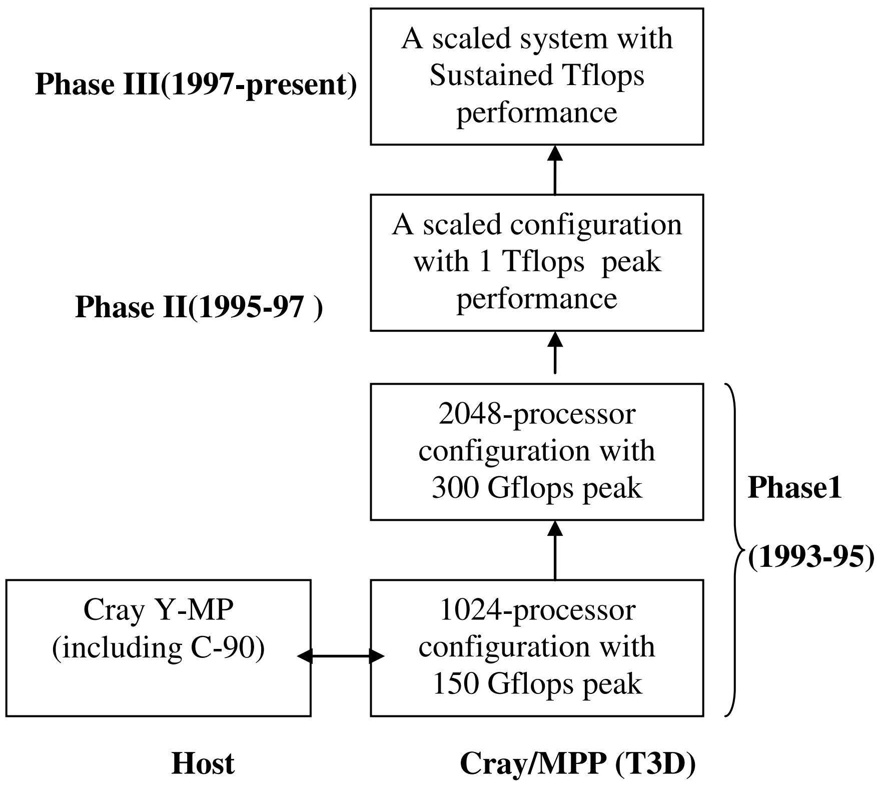


(c) The Cray/MPP system:

- Massively parallel processing (MPP) systems have the potential for solving highly parallel problems.
- It is a balanced system that matches fast processor speed with fast I/O, fast memory access, and capable software.
- Cray's first MPP was code named T3D because a three dimensional, dense torus network was used to interconnect the machine resources. The heart of Cray T3D was a scalable macroarchitecture having the special feature like;

1. The T3D was an MIMD machine that could be dynamically partitioned to emulate SIMD or multicomputer MIMID operations. The T3D network was scalable.
2. The system used a globally addressable, physically distributed memory.
3. It used a Mach-based microkernel OS.
4. Software portability was a major design goal in Cray/MPP series.
5. Its Compiler was modified with extended directives for MPP applications.

Cray/MPP development phases:



Phase III(1997-present)
A scaled system with Sustained Tflops performance
Phase II(1995-97)
A scaled configuration with 1 Tflops peak performance
2048-processor configuration with 300 Gflops peak
Phase1
(1993-95)
Cray Y-MP (including C-90)
1024-processor configuration with 150 Gflops peak
Host
Cray/MPP (T3D)

(d) Fujitsu VP2000 and VPP500: The Fujitsu VP2000 offered one or two processor configuration. The VPP 500 series offered from 7 to 222 processing elements (PEs) in a single MPP system. These two systems can be jointly used for large scale problems**.**

Fujitsu VP2000:

- The system is a dual processor. The system clock period was 3.2ns, the main memory unit was of 1 or 2 Gbytes, and the storage unit provided upto 32 Gbytes of extended memory.

- Each vector processing unit consisted of two load/store pipelines, three functional pipelines, and two mask pipelines. Two scalar units could be attached to each vector unit, making a maximum of four scalar units in the dual –processor configuration.

Fujitsu VPP500:

- This supercomputer called vector parallel processor.
- The architecture was scalable from 7 to 222 PEs, offering a high parallel MIMD multivector system. The peak performance was 333Gflops.
- Two control processors are used to coordinate the activities of PEs through a crossbar network. The data transfer unit in each PEs handles the interprocessor communication. Each PE has its own memory with 256Mbytes of RAM. Each PE had a scalar unit and vector unit operating in parallel

<u>Q18: Explain the Mainframe and Minisupercomputers?</u>

(a) **High end Mainframe supercomputers:** This class of supercomputers called near supercomputers which are mainly used for business and transaction processing. Here vector hardware is an optional feature which could be used concurrently with the scalar unit.

DEC VAX 9000:

- This system is four processor configurations.
- The system control unit utilized a crossbar switch which was designed to monitor the contents of cache memories, tracking the most up to

date cache content to maintain coherence. The cross bar had eight ports to four processors, two memory modules, and two I/O controllers.

- Each vector processor (VBOX) was equipped with an add and multiply pipeline using vector registers and mask /address generator. Vector instructions were fetched through the memory unit (MBOX), decoded in the (IBOX) and issued to the VBOX by the EOBOX. Scalar operations were directly executed in the EBOX.
- The service processor is used as an interface to control and monitoring the entire system for reliable operation and fault diagnosis.

(b) **Minisupercomputers:** Both scalar and vector processing was supported in this multiprocessors systems with shared memory and peripherals. Most of the systems were built with a graphics subsystem for visualization and performance tuning purposes.

(c) **Stardent 3000 :**The Stardent was a multiprocessor workstation .

- Here two buses were used for communication between the four CPUs , memory ,I/O, and graphics subsystems. The R-bus dedicated to data transfers from memory to the vector processor, and the S-bus handled all other transfers. The system could support a maximum of 512 Mbytes of memory.

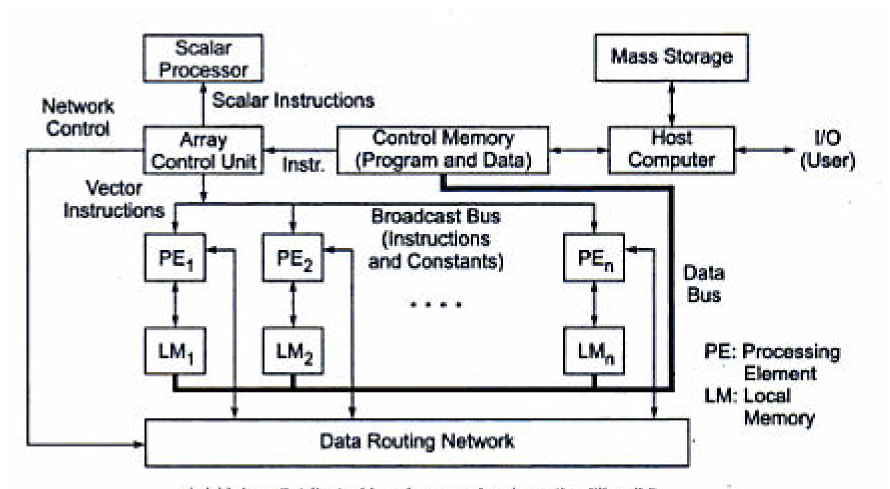
- A full graphical system(base graphics board and Graphics expansion board) consisted of two boards that were tightly coupled to the CPUs and the memory. These boards incorporated rasterizers (pixels and polygon processor) , frame buffers, Z-buffers, and additional overlay and control planes. This system was designed for numerically intensive computing with two-three dimensional rendering graphics.

SIMD COMPUTER ORGANIZATIONS

Q19: Describe the SIMD implementation models?

Two SIMD computer models are described based on the memory distribution and the addressing scheme used. The instruction set of an SIMD computer is decoded by the array control unit. The processing elements (PEs) in the SIMD array are passive ALUs executing instructions broadcast from the same array control unit .All PEs must operate in lockstep, synchronized by the same array controller.

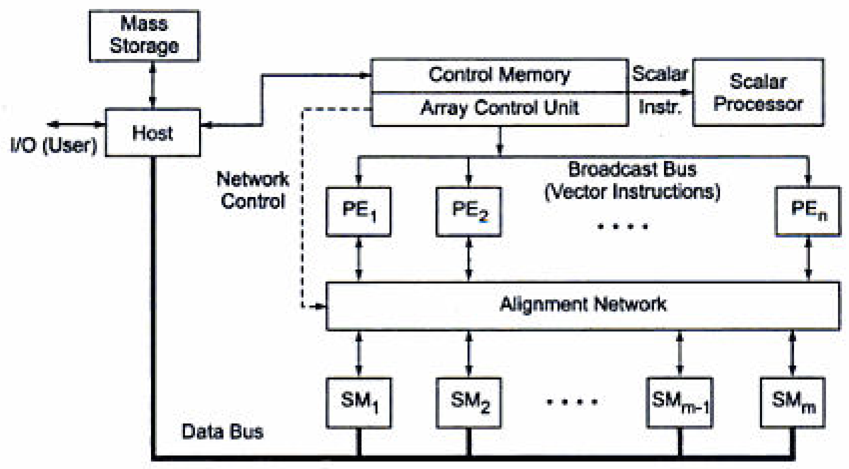
Distributed memory model:



(a) Using distributed local memories (e.g. the Illiac IV)

- A distributed–memory SIMD computer consists of an array of PEs which are controlled by the same array control unit.
- Program and data are loaded onto the control memory through the host computer.
- An instruction is sent to the control unit for decoding.
- If it is a scalar or program control operation, it will be directly executed by the scalar processor attached to the control unit. If the decoded instruction is a vector operation, it will be broadcast to all PEs for parallel execution.
- Partitioned data sets are distributed to all the local memories attached to the PEs through a vector data bus.
- The PEs are interconnected by a data-routing network which performs inter-PE data communication such as shifting, permutation and other routing operations. The data – routing network is under program control through control unit.
- The PEs are synchronized in hardware by the control unit.

Shared-memory model: Here the SIMD computer is using shared memory among the PEs. The interconnection network must be set properly to avoid access conflicts.



(b) Using shared-memory modules (e.g. the BSP)

SIMD instructions:

- SIMD computer executes vector instructions for arithmetic, logic, data-routing and masking operations over vector quantities.
- The data-routing instructions include permutations, broadcast, multicast and various rotate and shift operations. Masking operations are used to enable or disable a subset of PEs in any instruction cycle.
- All SIMD instructions must use vector operands of equal length, where n is the no of PEs.

Host and I/O:

- All I /O activities are handled by the host computer.
- A special control memory is used between the host and the array control unit. This is a staging memory for holding program and data.
- Divided data sets are distributed to the local memories or to the shared memory modules before starting the program execution. The host manages the mass storage and graphics displays of computational results.
- The scalar processor operates concurrently with the PE array under the coordination of the control unit.

Q20: Examples of SIMD computers:

(a) The CM-2 architecture:

Program execution paradigm: All programs started program exaction on a front-end, which issued microinstructions to the back-end processing array. The sequencer broke down the macroinstructions and broadcast them to all data processors in the array. Data sets and results could be exchanged between the front-end and the back-end processing array in one of the three ways: broadcasting, global combining, and scalar memory data bus .Broadcasting was carried out through the broadcast bus to all data processors at once. Global combining allowed the front-end to obtain the sum, largest, logical Or etc., of values , one from each processor. The scalar bus allowed the front-end to read or to write one 32-n=bit value at a time from or to the memories attached to the data processors.

The processing array: The processing array contained 4k to 64K bit-slices processors(PEs) ,all of which were controlled by a sequencer. The processors exchanged data among themselves in parallel through the router, NEWS grid, or a scanning mechanism. These network elements were also connected to I/O interfaces. A mass storage subsystem called the data vault was connected through the I/O for storing the data.

Processing nodes: Here each processing node contained 32-bit slice data processors. (The processor chips were paired in each node sharing a group of memory chips. It consists of two processing chips and each processing chip contained I6 processors).

Hypercube routers: Special hardware was built on each processor chip for data routing among processors. The router nodes on all processor chips were wired together to form a Boolean n cube. Each router node was connected to 12 other nodes, including its paired node.

The NEWS grid: The “NEWS” grid was based on the fact that each processor has a north, east, west and south neighbour in the various grid configurations. The flexible interconnections among the processors made it very efficient to router data on dedicated grid configurations based on the application requirements.

Scanning and spreading mechanisms: Scanning on NEWS grid combined communication and computation. The operation could simultaneously scan in every row of a grid along a particular dimension for the sum of that row, the largest or smallest, logical Or etc. Scanning operations could be expanded to cover all elements of an array.

I/O and data vault: High speed I/O channels are used for data and/or image I/O operations. Peripherals attached to the I/o channels included a data vault, which was a disk-based mass storage system for storing program files and large databases.

Major applications: The CM-2 was applied in almost all the MPP and grand challenge application including document retrieval, memory based reasoning techniques, natural language processing, VLSI circuit analysis and layout, computational fluid dynamic, signal/image/vision processing and integration, neural network simulation and modelling, dynamic programming, context-free parsing and computational geometry problems.

(b) The MasPar MP-1 architecture:

The MasPar MP-1: The MP-1 consists of four subsystems: the PE array. The array control unit (ACU), a UNIX subsystem with standard I/O, and a high speed I/o subsystem. The UNIX subsystem handled traditional serial processing. The high speed I/O working together with the PE array, handled massively parallel computing.

Array control unit: The ACU fetched and decoded MP-1 instructions ,computed address and scalar data values, issued control signals to the PE array, and monitored the status of PE array. An implemented functional unit , called a memory machine was used in parallel with the ACU. The memory machine performed PE array load and store operations, while the ACU broadcast arithmetic, logic, and routing instructions to the PEs for parallel execution.

The PE array: Each processor board had 1024 PEs and associated memory arranged as 64 PE clusters (PEC). Each PEC was composed of 16 PEs and 16 processor memories. Interprocessor communications were carried out via three mechanisms: (1) ACU-PE array communication (2) X-net nearest neighbour communication (3) Global crossbar router communications. The first mechanism supported ACU instruction/data broadcast to all PEs in the array simultaneously and performed global reductions on parallel data to cover values from the array.

X-Net mesh interconnect: The X-net interconnect directly connected each PE with its eight neighbours in the two dimensional mesh. Each PE had four connections at its diagonal corners, forming an X pattern. The connections to the PE array edges were wrapped around to form a 2-D torus.

Multistage crossbar interconnect: Here three router stages are implemented for the function of a 1024 x1024 crossbar switch. Each PE cluster shared an originating port connected to router stage S1 and a target port connected to router stage S3. Connections were established from an originating PE through stages S1, S2 and S3 and then to target PE.

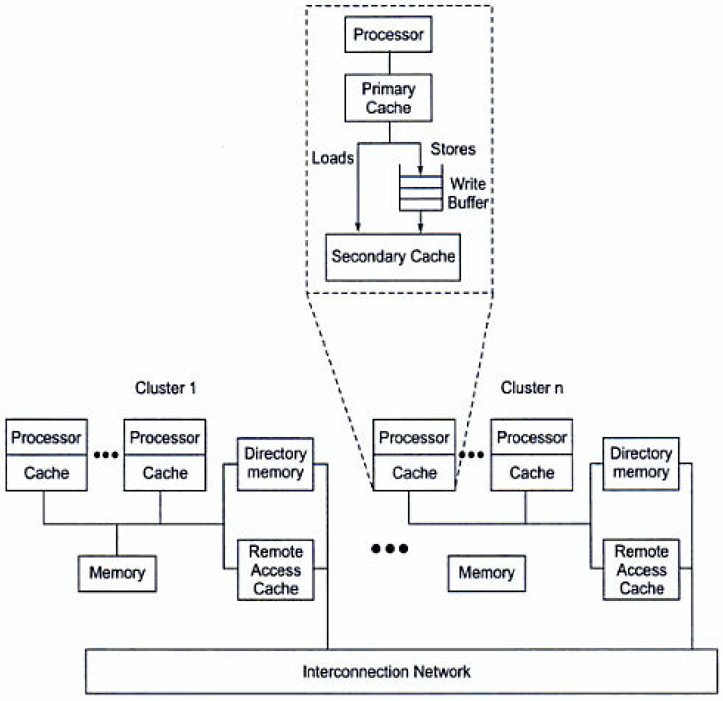
Processor elements and Memory: Each PE had forty 32bit registers available to the programmer and eight 32 bit registers for system use. The registers were bit and byte addressable. Each PE had a 4-bit integer ALU, a 1-bit logic unit, a 64-bit mantissa unit, a 16-bit exponent unit and a flag unit. Data movement was occurred through NIBBLE bus (4 bits wide) and the BIT bus(1 bit wide).

Parallel disk arrays: The PE array communicated with a parallel disk array through the high speed I/O subsystem. The parallel disk array was used to support data-parallel computation and provide file system transparency and multilevel fault tolerance.

LATENCY HIDING TECHNIQUES

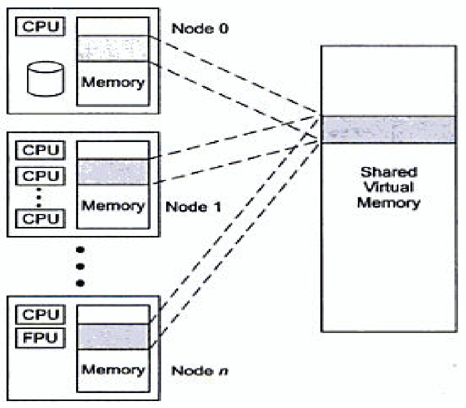
Q21: Describe SVM (Shared virtual memory)?

- Single–address-space multiprocessors/ multicomputers must use shared virtual memory.



The architecture environment : Here describes an architecture using SVM ,named as Dash Architecture (based on Stanford Dash experience) The dash architecture was a large scale, cache-coherent NUMA multiprocessor system. It consisted of multiple multiprocessor clusters connected through a scalable, low latency interconnection network. Physical memory was distributed among the processing nodes in various clusters. The distributed memory formed a global address space. Cache coherence was maintained using invalidating, distributed directory –based protocol (For each memory block, the directory keep track of remote nodes cacheing it. When a write occurred, point-to-point messages were sent to invalidate remote copies of the block. ACK messages were used to inform the originating node when an invalidation was completed). Two levels of local cache were used in processing nodes. Loads and writes were separated with use of write buffers for implementing weaker memory consistency models. The main memory was shared by all processing nodes in the same cluster. To facilitate prefetching and directory –based coherence protocol, directory memory and remote-access caches were used for each cluster. The remote-access cache was shared by all processors in the same cluster**.**

SVM concept: A global virtual address space is shared among processors residing at a large no of loosely coupled processing nodes. The idea is to implement coherent shared memory on a network of processors without physically sharing main memory,but have to share virtual memory. The system uses virtual addresses instead of physical addresses. Each virtual address space can be as large as a single node can provide and is shared by all nodes in the system. The SVM address space is organized in pages which can be accessed by any node in the system.



SVM mapping or page swapping|: Pages that are marked read-only can have copies residing in the physical memories of other processors. When a processor writes a page, it must update the page and then invalidate all copies on the other processors. A memory reference causes a page fault (if the demanded page not in memory), the memory manager retrieves the missing page from the memory of another processor. If there is a page frame available on the receiving node, new page placed there. Otherwise page replacement policies are used. The large virtual memory allows programs to be larger in code and data space then the physical memory on a single node**.**

Q22: Describe the latency hiding techniques?

In distributed shared memory machines, access to remote memory is likely to be slow compared to the ever-increasing speeds of processors and thereby reduces performance. Thus, any scalable architecture must rely on techniques to reduce/hide/tolerate remote-memory-access latencies. Latency hiding can be accomplished by

1. Using prefetching techniques ,which bring instructions or data close to the processor before they are actually needed.
2. Use of coherent caches supported by hardware to reduce cache misses.
3. Using relaxed memory consistency models by allowing buffering and pipelining of memory references.
4. Using multiple-contexts, which allows a processor to switch from one context to another when a long latency operation is encountered.

Prefetching techniques: Prefetching uses knowledge about the expected misses in a program to move the corresponding data close to the processor before it is actually needed. Prefetching can be classified based on whether it is binding or nonbinding, and whether it is controlled by hardware or software.

- **Binding prefetching**: Here the value of a later reference is bound at the time when the prefetch completes. This places restrictions on when a binding prefetch can be issued, the value will become stale if another processor modifies the same location during the interval prefetch and reference. Binding may result in a significant loss in performance due tot this limitation.

- **Nonbinding prefetching**: It also brings data close to the processor, but the data remains visible to the cache coherence protocol and is thus kept consistent until the processor actually reads the value.

- **Hardware prefetching**: It uses long cache lines (using spatial locality of reference) and instruction look ahead buffer for storing prefetched data.

- **Software prefetching**: It uses explicit prefetch instructions for prefetching. But the disadvantage includes the extra instruction overhead required to generate the prefetches, as well as the need for sophiscated software intervention.

  Benefits**:** It improves performance and reduces network traffic

  Distributed coherent caches : Here coherent caches are used in multiprocessors. So the cache coherence can be solved by using either snoopy bus or directory based protocol. Coherent caches reduce cache misses and thus reduce latency (if a cache miss occurs, the needed modified data should be load from memory or other remote cache of remote processor and increases latency).

  Benefits**:** Reduction in the no: of cycles time delay or latency) wasted due to read/write misses**.**

Relaxed memory consistency: It describes that for reducing latency, must use processor consistent memory models or relaxed consistent memory models**.**

Processor consistent memory models : Under processor consistency, writes issued by the same processor are always in program order , but writes by different processors may be observed in different orders by different processors .Two conditions related to other processors are required for ensuring processor consistency:

- Before a read is allowed to perform with respect to other processor, all previous read accesses must be performed.
- Before a write is allowed to perform with respect to other processor, all previous read or write accesses must be performed.

These conditions allow Read operations following a write operation may bypass it

Relaxed consistent memory models: Under release consistency, synchronization operations are classified into two categories: acquire (e.g. lock) and release (e.g. unlock).An acquire is a read operation that gains permission to access a set of data, while a release is write operation that gives away the permission.

- The main advantage of relaxed models is the potential for increased performance by hiding the write latency as much as possible. The main disadvantage is increased hardware complexity and a more complex programming model.
- Three conditions ensuring relaxed consistency

1. Before an ordinary read or write accesses is allowed to perform with respect to other processor, all previous acquire accesses must be performed.

2. Before release access is allowed to perform with respect to other processor, all previous ordinary read or write accesses must be performed.

3. Special accesses are processor consistent with one another

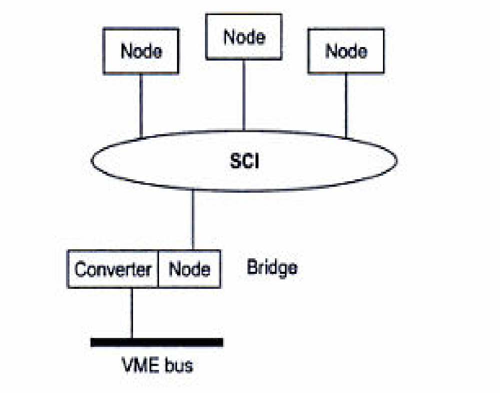
- Release consistency can be satisfied by (i) stalling the processor on an acquire access until it completes, and (ii) delaying the completion of release access until all previous memory accesses completes .

<u>Benefits</u>**:** It removes all idle time due to read or write miss latency.

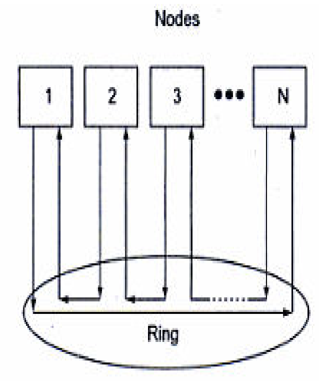
<u>Q23: Explain SCI (Scalable Coherent Interface) bus?</u>

It is a scalable coherent interconnect structure with low latency. It provides unidirectional, point-to-point connections, with two such links between each pair of nodes. The cache coherence protocols used in SCI are directory based protocols**.**

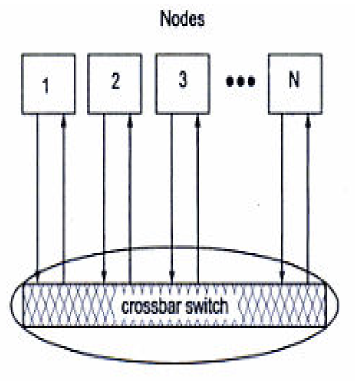
SCI Interconnect Models: SCI defines the interface between nodes and the external interconnect. The SCI interconnect can assume a ring structure or a crossbar switches . Each node has an input and an output link which are connected from or to the SCI ring or bus. The converter is used to bridge the SCI ring to the VME bus.



(a) Typical SCI configuration with bridge to other bus



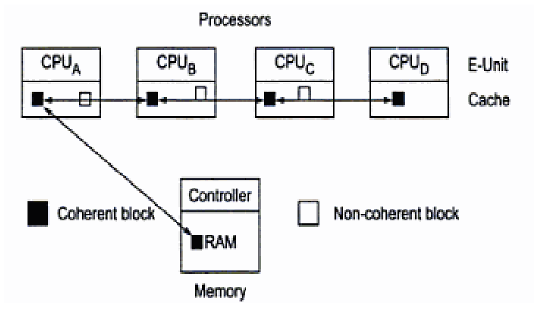
(b) A ring for point-to-point transactions



(b) A crossbar multiprocessor

Sharing –List Structures:

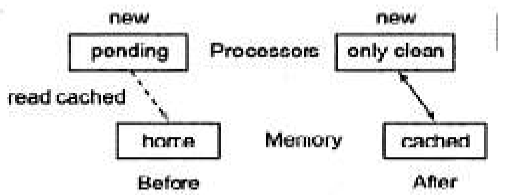
- Sharing lists are used in SCI to build chained directories for cache coherence use. Sharing lists are dynamically created, updated, and destroyed.
- Each coherently cached block is entered onto a list of processors sharing the block . Processors have the option of bypassing the coherence protocol for non coherent local cached blocks.
- For every block address, the memory and cache entries have additional tag bits which are used to identify the first processor (head) in the list and to link the previous and following nodes. Doubly linked lists are maintained between processors in the sharing list, with forward and backward pointers.



Sharing –List creation:

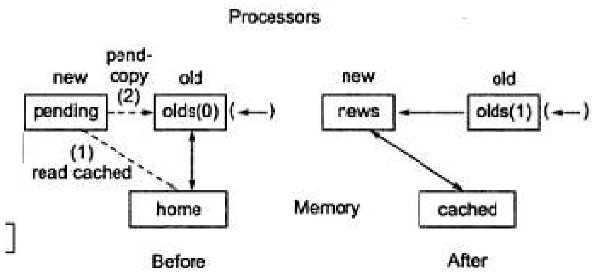
- The states of the sharing list are defined by the state of the memory and the states of the sharing list entries.
- The shared memory is in a home (uncached) or a cached (sharing list) state. The sharing list entries specify the location of the entry in a list, only entry, identify the only entry in the list, or specify the entry's cache properties such as clean, dirty, valid or stale.

- The head processor is always responsible for list management.
- The memory is initially in the home state, and all cache copies are invalid. Sharing list creation begins at the cache where an entry is changed from an invalid to a pending state. When a read cache transaction is directed from a processor to the memory controller, the memory state is changed from uncached to cached and the requested data is returned. The requester cache entry state is then changed from a pending state to an only-clean state.



(a) Creation of sharing list

Sharing –List updation:



(b) Addition of new nodes

- For subsequent memory access, the memory state is cached, and the cache head of the sharing list has dirty data.
- As shown in the figure, a new requester (Cache A) first directs its read-cache transaction to memory but receives a pointer to cache B instead of requested data.
- A second cache-cache transaction, called prepend, is directed from Cache A to Cache B. Cache B then sets its backward pointer to point to cache A and returns the requested data.
- The dashed lines correspond to transaction between the processor and memory or another processor. The solid lines are sharing-list pointers.
- After the transaction, the inserted cache A becomes the new head, and the old head, cache B is the middle as shown in the figure.

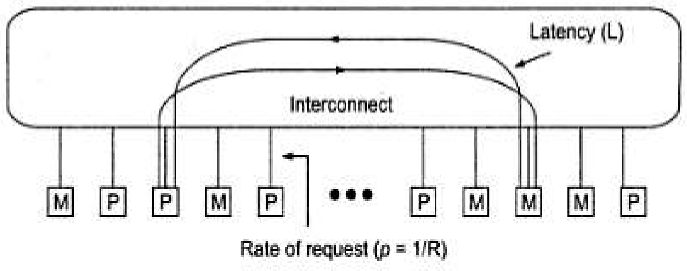
- Any Sharing-list may delete itself from the list. The head of the sharing list has the authority to remove other entries from the list to obtain an exclusive entry. Others may reenter as a new list head.

PRINCIPLES OF MULTITHREADING

Q24: Explain the Multithreaded architecture and computation model?

Multithreading demands that the processor be designed to handle multiple contexts (threads) simultaneously on a context-switching basis.

Architecture environment: A multithreaded MPP system can be modeled as a network of processors (P) and memory (M) nodes. The distributed memories form a global address space.



(a) The architecture environment. (Courtesy of Rafael Saavedra, 1992)

Four parameters are defined below to analyze the performance of the network.

(1) The latency (L): This is the communication latency on a remote memory access. The value of L includes the network delays, cache-miss penalty, and delays caused by split transactions.

(2) The number of threads (N): This is the no; of threads interleaved in each processor. A thread is represented by a context consisting of a program counter, a register set, and the required status words.

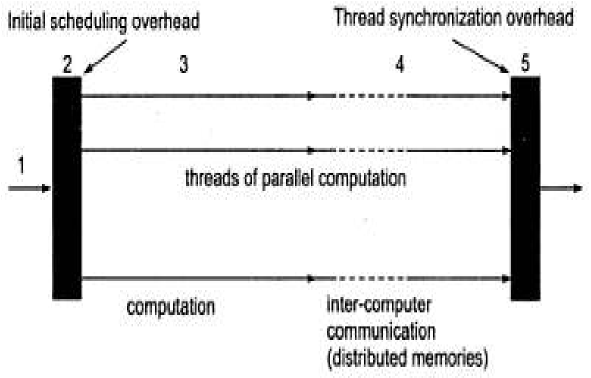
(3) The context-switching overhead: This refers to the cycles lost in performing context switching in a processor, which depends on the switching mechanism used.

(4) The interval between the switches (R) : This refers to the cycles between switches triggered by remote references. The inverse p=1/R is called the rate of requests for remote accesses. In order to increase efficiency, one approach is to reduce p. Another is to eliminate processor waiting through multithreading.

Multithreaded computation model:

The computation starts with a sequential thread (10, followed by supervisory scheduling (2) where the processors begin threads of computation (3), by interprocessor messages that update variables among the nodes when the computer has a distributed memory (4), and finally by synchronization prior to beginning the next unit of the parallel work (5).

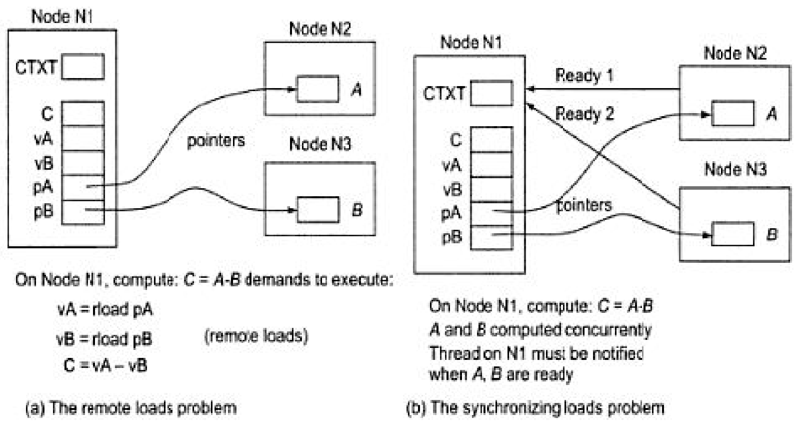
The communication overhead period (4) or message–passing overhead (send and receive calls) in multicomputers can be reduced by specialized hardware operating in parallel with computation.


Initial scheduling overhead
Thread synchronization overhead
2
3
4
5
1
threads of parallel computation
computation
inter-computer communication (distributed memories)

(b) Multithreaded computation model. (Courtesy of Gordon Bell, *Commun. ACM*, August 1992)

Q25: Describe the two problems caused by asynchrony and communication latency (problem of asynchrony)? Specify the solutions?

Massively parallel processors (MPP) operate asynchronously in a network environment. The asynchrony triggers two fundamental latency problems: remote loads and synchronizing loads.


Node N1
CTXT
C
vA
vB
pA
pB
Node N2
A
pointers
Node N3
B
On Node N1, compute: C = A-B demands to execute:
vA = rload pA
vB = rload pB
(remote loads)
C = vA – vB
(a) The remote loads problem
Node N1
CTXT
C
vA
vB
pA
pB
Ready 1
Ready 2
pointers
Node N2
A
Node N3
B
On Node N1, compute: C = A-B
A and B computed concurrently
Thread on N1 must be notified when A, B are ready
(b) The synchronizing loads problem

g. 9.12 Two common problems caused by asynchrony and communication latency in massively parallel processors (Courtesy of R.S. Nikhil, Digital Equipment Corporation, 1992)

Remote loads problem: Variable A and B are located on nodes N2 and N3 respectively. They need to be loaded into node N1 to compute the difference C=A-B. The basic computation demands the execution of two remote loads and then the subtraction. Here the two rloads cause latency.

Synchronizing loads problem: In this case, A and B are computed by concurrent processes, and only after the ready signals reach node N1 asynchronously. Here the two synchronizing loads cause latency.

Solutions:

Multithreading solutions:

- The solution is to multiplex among many threads. When one thread issues a remote-load request , the processor begins work on another thread, and so on. The cost of thread switching should be smaller than that of the latency of the remote load.
- Another concern is to make sure that messages carry continuations; Suppose, after issuing a remote load from thread T1, switch to thread T2, which also issues a remoter load request and so on. The response may not return in the same order.
- This problem can be solved by associating each remote load and response with an identifier for the appropriate thread, so that it can re-enabled on the arrival of a response. These thread identifiers are referred to as continuations on messages. A large continuation name space should be provided to name an adequate no of threads waiting for remote responses.

Distributed cacheing : Every memory location has an owner node. For eg: N1 owns B and N2 owns A. The directories are used to contain import-export lists and state whether the data is shared (for reads, many caches may hold copies) or exclusive (for writes, one cache holds the current value).

- Distributed cacheing offers a solution only for the remote-load problem and multithreading solutions offers solution for both the remote load and synchronizing loads problems**.**

<u>Q25: Explain Multiple-context processors?</u>

Multithreaded systems are constructed with multiple-context (or multithreaded) processors.

The enhanced processor model:

- A conventional single-thread processor will wait during a remote reference, so it is idle for a period of time L.
- A multithreaded processor will suspend the current context and switch to another, so after some fixed no: of cycles it will again be busy doing useful work, even though remote reference is outstanding. Only if all the contexts are suspended (blocked) will the processor be idle.
- The efficiency of the performance can be defined as

$$\text{Efficiency} = \frac{\text{busy}}{\text{busy + switching + idle}}$$

where the busy, switching and idle represent the amount of time , measured over some large interval, that the processor is in the corresponding state.

- The state of the processor is determined by the disposition of various contexts on the processor.
- During its lifetime, a context cycles through the following states: ready, running, leaving and blocked. There can be at most one context running or leaving.
- A processor is busy if there is a context in the running state; it is switching while making the transition from one context to another, i.e , when a context is leaving. Otherwise, all contexts are blocked and the processor is idle.

- A running context keeps the processor busy until it issues an operation that requires a context switch. The context then spends C cycles in the leaving state, then goes into the blocked state for L cycles, and finally reenters the ready state. Eventually the processor will choose it and the cycle will start again.

Context switching policies: Different multithreaded architectures are distinguished by the context-switching policies adopted. The four context-switching policies are

1.Switch on cache miss: This corresponds to the case where a context is preempted when it causes a cache miss. In this case, R is taken to be the average between the misses and L be the time required to satisfy the miss

2.Switch on every load: This allows switching on every load, independent of whether it will cause a miss or not, Here R is the average between loads.

3.Switch on every instruction: This allows switching on every instruction, independent of whether it is a load or not. In other words, it interleaves the instructions from different threads on a cycle-by-cycle basis.

4.Switch on block of instruction: Blocks of instructions from different threads are interleaved. This will improve the cache-hit ratio due to locality. It will also benefit single-context performance

Processor efficiencies:

- The efficiency of a single-thread machine is : $E_1 = \frac{R}{R+L} = \frac{1}{1+L/R}$

- The efficiency of a multithread machine is analyzed by two conditions

Saturation region: In this region, the processor operates with maximum utilization. The cycle of the renewal process is R +C , and the efficiency is simple

$$E_{sat} = \frac{R}{R + C} = \frac{1}{1 + C/R}$$

It observes that the efficiency in saturation is independent of the latency and also does not change with a further increase in the no: of contexts.

Linear region : When the no; of contexts is below the saturation point, there may be no ready contexts after a context switch, so the processor will be idle. The tome required to switch to a ready context, execute it until a remote reference is issued, and process the reference is equal to R + C +L. Assuming N is below the saturation point, during this time all other contexts have a turn in the processor, and thus the efficiency is simple

$$E_{lin} = \frac{NR}{R + C + L}$$

It observes that the efficiency increases linearly with the no: of contexts until the saturation point is reached and beyond that remains constant.

Q26: Explain Multidimensional architectures (examples of multithreaded system)?

The architecture of MPP or multithreaded systems has evolved from one-dimensional rings to two-dimensional mesh and then to three-dimensional torus. The figure shows the evolution and examples architectures which has adopted the same.

The Wiscosin multicube:

- It employed a snooping cache system over a grid of buses.
- Each processor was connected to a multilevel cache. The first –level cache called the processor cache, was a high-performance SRAm cache designed with the traditional

goal of minimizing memory latency. A second-level cache, referred to as the snooping cache , was a very large cache designed to minimize bus traffic.

- Each snooping cache monitors two buses, a row bus and a column bus, on order to maintain data consistency among the snooping caches. Consistency between the two cache levels was maintained by using write-through strategy. The main memory was divided among the column buses. All processors tied to he same column shared the same home memory. The row buses were used for intercolumn communication and cache coherence control.

The orthogonal multiprocessor (OMP) :

- In this architecture, n processors simultaneously access n rows or n columns of interleaved memory modules.
- The n x n memory mesh is interleaved in both dimensions. In other words, each row is n-way interleaved and so is each column of memory modules.
- There are 2n logical buses spanning in two orthogonal directions.
- The synchronized row access or column access must be performed exclusively. In fact, the row bus Ri and the column bus Ci can be the same physical bus because only one of the two will be used at a time.
- The OMP can be used for matrix algebraic computations, image processing operations, and SPMD operations.

Multidimensional extensions:

- A generalized orthogonal multiprocessor is denoted as an OMP(n, k) , where n is the dimension and k is the multiplicity.
- There are $p=k^{n-1}$ processors and $m=k^n$ memory modules in the system, where p>> n and p>>k.
- The system uses p memory buses spanning into n dimensions.

- There are k memory modules attached to each spanning bus. Each module is connected to n out of n -way switch. Thus implies that each module is shared by n out of $p=k^{n-1}$ processors.

- The processors access memory modules via buses, spanning in to three directions, called the x-access, y-access and z-access.

SCALABLE AND MULTITHREADED ARCHITECTURES

Q27: Explain the Stanford dash multiprocessor?

This multiprocessor was developed by Stanford University, in which the name Dash is an abbreviation for Directory Architecture for shared memory: It was a Scalable high performance machine with single address space, distributed coherent caches, and distributed memories.

The prototype Architecture: It consists of 64 MIPS R3000/R3010 microprocessors with 16 cluster's of 4 PEs each. The interconnection network among the 16 multiprocessors clusters using two mesh network. One mesh network was used to request remote memory, and the other was a reply mesh. This architecture was modified from the existing system by designing a pair of new boards to support the directory memory and intercluster interface.

Dash memory hierarchy: A memory location could be in one of the three states:

Uncached**-** not cached by any cluster

Shared**-** in an unmodified state in the caches of one or more clusters; or

Dirty**-**modified in a single cache of some cluster.

- The directory kept the information for each memory block, specifying the state and the clusters cacheing it.
- The Dash memory system could be logically broken into four levels of hierarchy.
- The first level was the processor cache which was designed to match the processor speed. It took only one clock to access the processor cache.
- A request that could not be serviced by the processor cache was sent to the next level containing the local cluster. The prototype allowed 30 processor clocks to access the local cluster.
- This level included the other processor cache's within the requesting processor's cluster. Otherwise, the request was sent to the home cluster level. The home level consisted of the cluster that contained the directory and physical memory for a given memory address. It took 100 processor clocks to access the directory at the home level.
- The home cluster could usually satisfy the request immediately, but if the directory entry was in a dirty state, or in a shared state when the requesting processor requested exclusive access, the fourth level had to be accessed. The remote cluster level for a memory block consisted of the clusters marked by the directory s holding a copy of the block. It took 135 processor clocks to access processor caches in remote clusters in this prototype design.

Q28: Explain the Kendall Square Research KSR-1?

The KSR-1 was a scalable multiprocessor with cache-only memory architecture (COMA). It was a size and generation scalable shared-memory multiprocessor computer**.**

The KSR-1 architecture: Here scalability was achieved by connecting 32 processors to form a ring structure. The KSR-1 used a two-level hierarchy to interconnect the processors as a ring structure. Each node comprised a primary cache, acting as a 32-Mbyte primary memory and a 64-bit superscalar processor**.**

The ALLCACHE memory: The KSR-1 eliminated the memory hierarchy found in conventional multiprocessor and the corresponding physical memory addressing overhead. For this, it offered a single –level memory called ALLCACHE memory. The architecture provided size and generation scalability in that every node was identical, and it offered an efficient environment for both arbitrary workloads and sequential to parallel processing though a large hardware-supported address space with an unlimited number of processors.

Programming model: The KSR machine provided a strict sequentially consistent programming model and dynamic management of memory through hardware migration and replication of data throughout the distributed processor memory nodes using its ALLCACHE mechanism. The hardware was designed to exploit spatial and temporal locality.

Environment and performance: It provides a commercial programming environment for transaction processing that accessed relational databases with unlimited scalability .Message passing was supported by pointer passing in the shared memory to avoid data copying and enhance performance.

Q29: Explain the Tera multiprocessor System?

The Tera multiprocessor System was implemented with VLSI circuits and packaging technology. A 400-MKz clock with maximum of 128 threads was proposed to use per processor. It provides high degree of multithreading and high degree of pipelining.

The Tera design goals:

- First it needed to be scalable for very high-speed implementations i.e,. have a short clock period and be scalable to a many processors.
- Second, it was applicable to a wide spectrum of problems.
- A third goal was ease of compiler implementation.

The sparse three-dimensional torus: The interconnection network was a three dimensional sparsely populated torus of pipelined packet-switching nodes, each of which was linked to its neighbors. Some of the nodes were also linked to resources, i.e. processors, data memory units, I/O processors, and I/O cache units. The resources were uniformly distributed throughout the network. This permitted data to be placed in memory units near the appropriate processor.

Pipelined Support: Each processor in a Tera computer could execute multiple instruction streams (threads) simultaneously. In the initial implementation, 1 or many as 128 program counters could be active at once. On every tick of the clock, the processor logic selected a ready-to-execute thread and allowed it to issue its next instruction. Since instruction interpretation was completely pipelined by the processor and by the network and memories as well, a new instruction from a different thread could be issued during each tick without interfacing with its predecessors. When an instruction finished, the thread to which it belonged became ready to execute the next instruction. As long as there were enough threads in the processor so that the average instruction latency was filled with instructions from other threads, the processor was fully utilized. Thus it was the only necessary to have enough threads to hide the expected latency: once latency was hidden, the processor would run at peak performance and additional threads would not speed the result.

Thread state: Each thread had the following states associated with it:

- One 64-bit stream status word(SSW);
- Thirty –two 64-bit general purpose registers (R0-R31);
- Eight 64-bit target registers(T0-T7);

Explicit-Dependence lookahead: The Tera architecture used a new technique called explicit-Dependence lookahead. Each instruction contained a 3-bit lookahead field that explicitly specified how many instructions from this thread would be issued before encountering that depended on the current one. Since seven was the maximum, at most 8 instructions and 24 operations could be concurrently executing from each thread.

Advantages:

- The tera used multiple contexts to hide pipeline and communication latency.
- A large no: of registers are shared between threads. The thread creation was very cheap.
- Tagged memory and registers with full/empty bits were used for synchronization
- .Due to plenty of parallelism in user programs to hide latency and plenty of compiler support, the performance was potentially very high.

Drawbacks:

- The performance must be bad in case of limited parallelism.
- The contexts (threads) demanded lots of registers and other hardware resources which in turn implied higher cost and complexity.

DATA FLOW AND HYBRID ARCHITECTURES

Q30: Explain the Data flow and hybrid architecture?

- Dataflow computers have the potential for exploiting all the parallelism available in a program. Since execution is driven only by the availability of operands at the input to the function units, there is no need for a program counter in this architecture, and its parallelism is limited by the actual data dependences in the application program.
- A dataflow graph can be used as a machine language in dataflow computer, which specify only a partial order for the executions of instructions and thus provide opportunities for parallel and pipelined execution at the level of individual instructions.

  Eg; Dataflow graph for obtaining the approximation of cosx:

  cosx =1= $1 - x^2/2! + x^4/4! - x^6/6! = 1 - x^2/2 + x^4/24 - x^6/720$

Static versus Dynamic dataflow:

- Static dataflow computers simply disallow more than one token to reside on any one arc, which is enforced by the firing rule : A node is enabled as soon as tokens are present on all input arcs and there is no token on any of its output arcs;
- The static firing rule is difficult to implement in hardware. Special feedback ACK signals are needed to secure the correct token passing between producing nodes and consuming nodes. Also, the static rule makes it very inefficient to process arrays of data.
- In dynamic dataflow computers, each token is tagged with a context descriptor, called a tagged token. The firing rule is changed to: A node is enabled as soon as tokens with matching identical tags are present at each of its input arcs. Special hardware mechanisms are needed to perform tag matching.

Evolution of dataflow computer:

Pure dataflow Machines: The MIT tagged token dataflow architecture (TTDA), the Manchester dataflow computer, and the ETL sigma -1 are all pure dataflow machines.

Explicit token store machines: These were successors to pure dataflow computers. The basic idea is to implement efficient token matching. The MIT/Motorola monsoon and ETL/ EM-4 system are examples.

Hybrid and Unified architectures; These are architectures combining positive features from Von Neumann and dataflow computers. The best examples are MIT P-RISC, the IBM Empire and the MIT/Motorola *T. P-RISC was a "RISC-ified' architecture. It provides better performance by splitting complex dataflow instructions into separate simple component instructions that could be composed by the compiler.

Q31: Two Examples of Data flow and hybrid architecture:

(a) ETL/ EM-4 : Each EMC-R node was a single-chip processor .An Omega network was used to provide interconnections among the nodes.

The Node architecture:

- The processor consists of six component units.
- The processor chip communicated with the network through a 3 x 3 crossbar switch unit.

- The processor and its memory were interfaced with a memory control unit. The memory was used to hold programs as well as tokens waiting to be fetched.
- The input buffer was used as a token store with a capacity of 32 words.
- The fetch match unit fetched tokens from the memory and performed tag-matching operations among the tokens fetched .
- Instructions were directly fetched from the memory through the memory controller.
- The heart of the processor was the execution unit, which fetched instructions until the end of a thread. Instructions with matching tokens were executed. Instructions could emit tokens or write to registers. Instructions were fetched continually using thread sequencing until a stop flag was raised to indicate the end of a thread. Then another pair of tokens was accepted. Each instruction in a thread specified the two sources for the next instruction in the thread.

(b) MIT/Motorola *T:

The prototype architecture: The *T prototype was single-address-space system. A 'brick" of 16 nodes was packaged in a 9-cube. The local network was built with 8 x 8 crossbar switching chips. A 256-node machine could be built with 16bricks. The 16 bricks were interconnected by four switching boards. No cables were used between the boards. Each board implemented a16 x 16 crossbar network.

The *T Node design: Each node was designed to be implanted with four component units. A Motorola superscalar RISC Microporocessor was modified as a data processor (dP). This dP was optimized for long threads. A synchronization coprocessor (sP) was implemented as a special functional unit (SFU) , which was optimized for simple, short threads. Both the dp and sP could handle fast loads. The dP handled incoming continuation, while the sp handled incoming messages, rload/ rstore responses and joins for messaging or synchronization purposes. The memory controller handled requests for remote memory load or store, as well as the management of node memory. The network interface unit received or transmitted messages from or to the network.