Pipelines

• Assembly line analogy
  – used in manufacturing to increase productivity
  – divide a complex operation into pieces that can be performed in parallel
  – separate ``workers'' implement successive steps along the assembly line
  – when an item finishes one step it is passed down the line to next step

• Pipelines are used in two major areas
  – instruction processing
  – arithmetic operations

• A pipelined system satisfy the following requirements:
  – repeatedly executes a basic function
  – a basic function must be divisible into independent stages that have minimal overlap
  – the complexity of the stages should be roughly similar

• The number of stages is referred to as the depth of the pipeline
Pipelines 2

- Example: floating point addition of two numbers of the form $m \cdot 2^e$
  - breakdown of this function into stages as follows:
    1. If $e_2 < e_1$ swap the operands. Find the difference in exponents. $e_d = e_2 - e_1$.
    2. Shift $m_2$ to the right by $e_d$ bits.
    3. Compute the mantissa of the sum by adding $m_1$ and $m_2$. The exponent of the sum is $e_1$.
    4. Normalize the sum.

- The extra complexity of a pipelined adder pays off when adding long sequences of numbers
  - Operations at each stage can be done on different pairs of inputs
    e.g. one stage can be comparing the exponents in one pair of operands at the same time another stage is adding the mantissas of a different pair of operands

- Important that the operands must be independent
  - e.g. the two instructions: $R2 = R0 + R1$
    $R4 = R3 + R1$
    - both instructions identify $R1$ as one of their inputs
    - potential conflict in stages two and three because stage two might need to shift the mantissa of $R1$ at the same time stage three needs to add the mantissa of $R0$ to the mantissa of $R1$. 
Pipelines 3

• The solution is to make copies of the operands, and pass the copies through the pipeline.
• Thus the CPU gives the adder copies of R0 and R1 when it starts the pipeline for the first pair, and the second stage gets these copies from the first stage along with a value of \( e_d \).

  – Another potential problem
    • one of second instruction operands is R2:
      \[
      R2 = R0 + R1 \\
      R4 = R2 + R1
      \]
    • the second instruction depends on the result of the first instruction
    • CPU cannot send the second pair of operands to the pipelined adder until the result of the first addition exits the last stage
    • such interactions lead to time periods when the pipeline stages are empty which are called **bubbles**

• Gantt chart of cases from previous example
  – two successive instructions can be overlapped
    • first stage is busy with the second instruction while second stage is busy with the first instruction
    • in each successive cycle the two instructions are passed down the \``assembly line\'' to the next stage
    • the second sum is done after 6 cycles
  – second instruction must wait for first to complete
    • second sum is not finished until after the 10th cycle
    • \``bubble\'' in the pipeline is the 4-cycle dead period in each stage caused by delaying the second instruction
Pipelined Floating Point Adder

Case 1: two independent instructions, fully overlapped. The second is computed at the end of the 6th cycle.

Case 2: the second instruction must wait until the first is done, introducing a 4-cycle bubble.
Pipelines 4

- In general a pipeline of depth $d$ can process $n$ items in $n+d$ steps when there are no bubbles
  - without a pipeline, each application of the basic function would require $d$ cycles, and they would have to be executed sequentially, for a total time of $n*d$ cycles
  - the speedup obtained by a full pipeline is:
    $$\frac{n*d}{n+d}$$

- When $n >> d$, can safely ignore the $d$ in the denominator, so the asymptotic speedup, observed for large $n$, is a factor of $d$.
  - e.g. to add 1000 pairs of numbers; at 5 cycles per addition, a machine without a pipelined adder would require 5000 cycles
  - with a 5-stage pipelined adder, the last sum will appear after $1000 + 5$ cycles, so the pipeline is $5000/1005 = 4.97$ times faster

- Providing a steady stream of independent operands that will keep a pipeline full is the distinguishing feature of a vector processor
  - can initiate such a series of operations with a single instruction
Pipelines 5

• Many possible sources of bubbles in pipelines
  – dependencies between instructions are the main cause
  – for arithmetic pipelines, *data dependencies* arise when pairs of operations share inputs and outputs

• Note that the instructions do not have to be consecutive to present dependencies
  – a compiler that checks for dependencies and reorders instructions has to "look ahead" in the code by an amount equal to the depth of the pipeline

• Instruction pipelines used to speed up the fetch-decode-execute cycle are susceptible to bubbles
  – *control dependencies* are most commonly caused by branch or loop instructions
    • if the pipeline is "looking ahead" and fetching instructions it thinks the machine will execute, but the machine branches to another location, a bubble is introduced while the fetch stage goes to get the instructions at the new location
Pipelines 6

- Pipelines have been widely used in high performance machines for many years
  - the CDC 6600 is a classic early example of a CISC pipeline, used a "scoreboard" circuit to detect data dependencies between instructions and instruction buffers
  - the Cray-1 had every data processing unit was pipelined
  - until the late 1980s pipelining was one of the attributes that separated "mainframes" and supercomputers from microprocessors
  - with VLSI technology most microprocessors now have room on chip for the complex control circuitry and instruction scheduling logic associated with pipelined data processing units

CDC 6600
First computer to be termed a "supercomputer", it was introduced in 1966. Later model CDC 6600s had a peak performance rate of 3 Megaflops. It was the first machine with extensive internal parallelism and utilized a run-time controlled multifunction processor which attempted to increase performance by reorganizing code.
Vector Processors

• A processor that can operate on entire vectors with one instruction, i.e. the operands of some instructions specify complete vectors.
  - e.g. \( C = A + B \)
    • in both scalar and vector machines this means `add the contents of A to the contents of B and put the sum in C.''
    • in a scalar machine the operands are numbers,
    • in vector processors the operands are vectors and the instruction directs the machine to compute the pairwise sum of each pair of vector elements
  - A processor register, “the vector length register”, tells the processor how many individual additions to perform when it adds the vectors

• A vectorizing compiler is a compiler that will try to recognize when loops can be transformed into single vector instructions
  - e.g. the following loop can be executed by a single instruction on a vector processor

\[
\begin{align*}
  &\text{DO } 10 \ l=1,N \\
  &A(l) = B(l) + C(l) \\
  &10 \ CONTINUE
\end{align*}
\]
Vector Processors 2

- the code is translated into an instruction that would set the vector length to N followed by a vector add instruction
- vector instructions pays off in two different ways
  - first, the machine has to fetch and decode far fewer instructions, so the control unit overhead is greatly reduced and the necessary memory bandwidth is reduced
  - second, the instruction provides the processor with a regular source of data
    - the machine knows it will have to fetch $n$ pairs of operands which can be arranged in a regular pattern in memory and the processor can tell the memory system to start sending those pairs
    - with an interleaved memory, the pairs will arrive at a rate of one per cycle, at which point they can be routed directly to a pipelined data unit for processing

- Division of vector processors based on the way the instructions access their operands
  - in memory to memory organization, operands are fetched from memory and routed directly to the functional unit; results are streamed back out to memory as the operation proceeds.
  - in register to register organization, operands are first loaded into a set of vector registers; the vector operation then proceeds by fetching the operands from the vector registers and returning the results to a vector register
Vector Processors 3

• The advantage of memory to memory machines is the ability to process very long vectors
  – register to register machines must break long vectors into fixed length segments
  – this flexibility is offset by a relatively large startup time overhead (time between the initialization of the instruction and the first result from the pipeline)
  – long startup time on a memory to memory machine is a function of memory latency, which is longer than the time it takes to access a value in an internal register
  – once the pipeline is full, however, a result is produced every cycle

\[ T = S + aN \]

• A performance model for a vector processor:

where \( s \) is the startup time, \( N \) is the length of the vector and \( a \) is an instruction dependent constant, usually 1/2, 1 or 2

• The memory-to-memory architecture
  – examples: CDC Cyber 200 family and the ETA-10
  – machines appeared in the mid 1970s after a long development cycle and with dated technology
  – they disappeared in the mid 1980s
  – one reasons for their demise was the large startup time, which was on the order of 100 processor cycles
  – very inefficient for short vector operations
  – moderate (half max) performance for 100 length vectors
Vector Processors 4

• In register to register machines the vectors have a relatively short length, 64 in the case of the Cray family, but the startup time is far less than memory to memory machines
  – these machines are much more efficient for operations involving short vectors
  – long vector operations the vector registers must loaded with each segment before the operation can continue

• Register to register machines now dominate the vector computer market
  – offerings from Cray Research Inc., include the Y-MP and the C-90 and machines from Fujitsu, Hitachi and NEC
  – clock cycles on modern vector processors range from 2.5ns (NEC SX-3) to 4.2ns (Cray C90)
  – single processor performance on LINPACK benchmarks is in the range of 1 to 2 GFLOPS

The CRAY 1

The first major successful supercomputer was the CRAY-1 in 1976. It was designed by Seymour R. Cray who left Control Data Corporation to start his own company in 1972. The CRAY 1 had a top speed of 100 megaflops. If you tried to build one yourself using PCs, it would take 200 of them all cross connected, or you could just use 33.33 Sun4s. CRAY made at least 16 CRAY 1’s. In 1976, a typical CRAY 1 cost about $700,00. You could order the machine in any color you wished.
Vector Processors 5

- **Vector Chaining** was a feature introduced in the Cray computers
  - e.g. two vector instructions:
    
    \[
    \begin{align*}
    V_2 &= V_0 \times V_1 \\
    V_4 &= V_2 + V_3
    \end{align*}
    \]

  - the output of the first instruction is one of the operands of the second instruction
  - since these are vector instructions, the first instruction will route up to 64 pairs of numbers to a pipelined multiplier
  - about midway through the execution
    - the first few elements of V2 will contain recently computed products; the products that will eventually go into the next elements of V2 are still in the multiplier pipeline; and the remainder of the operands are still in V0 and V1, waiting to be fetched and routed to the pipeline

\[\text{pipeline depth}\]

\[V_2 \leftarrow V_0 \times V_1\]

\[V_4 \leftarrow V_2 \times V_3\]
This situation is where the operands from $V_0$ and $V_1$ that are currently in the multiplier pipeline are indicated by gray cells. At this point, the system is fetching $V_0[k]$ and $V_1[k]$ to route them to the first stage of the pipeline and $V_2[j]$ is just leaving the pipeline. Vector chaining relies on the path marked with an asterisk. While $V_2[j]$ is being stored in the vector register, it is also routed directly to the pipelined adder, where it is matched with $V_3[j]$. As the figure shows, the second instruction can begin even before the first finished, and while both are executing the machine is producing two results per cycle ($V_4[i]$ and $V_2[j]$) instead of just one.
Vector Processors 6

- Without vector chaining, the peak performance of the Cray-1 would have been 80 MFLOPS
  - one full pipeline producing a result every 12.5ns, or 80,000,000 results per second
  - with three pipelines chained together, there is a very short burst of time where all three are producing results, for a theoretical peak performance of 240 MFLOPS
    - (listed as 160 MFLOPS, because it was realistic to keep only two pipelines chained together for any reasonable period of time

- Vector chaining could be implemented in a memory-to-memory vector processor, but it would require much higher memory bandwidth to do so
  - without chaining, three channels must be used to fetch two input operand streams and store one result stream; with chaining, five channels would be needed for three inputs and two outputs
  - the ability to chain operations together to double performance gave register-to-register designs another competitive edge over memory-to-memory designs
Superscalar Processors

• Architectural concepts pioneered in vector processors and mainframe computers of the 1970s started to appear in RISC processors
  – VLSI technology made more room on the chip
  – rather than increase complexity of the architecture, most designers improved the execution of their current architecture
    • on-chip caches
    • instruction pipelines

• Latest step in the evolution is the superscalar processor, a scalar processor that is capable of executing more than one instruction in each cycle
  – in many superscalar designs, the high level architecture is unchanged from earlier scalar designs, but uses instruction level parallelism for improved implementation of these architectures

• Keys to superscalar execution are:
  – an instruction fetching unit that can fetch more than one instruction at a time from cache
  – instruction decoding logic that can decide when instructions are independent and thus executed simultaneously
  – sufficient execution units to be able to process several instructions at one time
Superscalar Processors 2

- The IBM RS/6000 is a good example with three major subsystems:
  - the instruction fetch unit
    - a 2-stage pipeline;
      - in the first stage a packet of four instructions is fetched from an instruction cache
      - in the second stage instructions are routed to the integer processor and/or floating point processor
    - this unit also executes branch instructions itself so that in a tight loop there is effectively no overhead from branching since the instruction unit executes branches while the data units are computing values
  - an integer processor
    - has a four-stage pipeline
    - in addition to executing data processing instructions this unit does some preprocessing for the floating point unit
  - a floating point processor
    - has a six stage pipeline

- Advantage of a superscalar processor is that it does not rely on a vectorizing compiler to detect loops and turn them into vector instructions
  - still requires a sophisticated compiler to allocate resources and schedule operations to best take advantage of the resources of the machine
  - but in the long run the superscalar approach is more flexible and applicable to a wider range of applications than vector processing
Topology

• The pathways over which the processors, memories, and switches communicate is a major consideration in parallel systems design
  – defines the interconnection network, or topology,
  – determines how processors will share data and at what cost

• Ring vs. Fully Connected Network

• Collection of nodes that communicate via links
  – nodes can be either processors, memories, or switches
  – links will always be point-to-point data paths, i.e. not buses that are shared by several nodes
  – a link connects two neighbor nodes
  – the degree of a node is the number of its neighbors
  – the diameter of a network is the longest path between any two nodes
    • a ring of \( n \) nodes has diameter \( n/2 \)
    • a fully connected network has a fixed diameter of 1
Topology 2

- The diameter of a ring grows as more nodes are added
  - the diameter of a fully connected network remains the same
  - a ring can expand indefinitely without changing the degree, but each time a new node is added to a fully connected network a link has to be added to each existing node

- Scalability refers to the increase in the complexity of communication as more nodes are added
  - in a highly scalable topology more nodes can be added without severely increasing the amount of logic required to implement the topology and without increasing the diameter

To construct a n-dimensional cube, copy an (n-1)-dimensional cube, the connect corresponding nodes in the original and the copy. (Nodes from the original are dark.)

- Hypercubes
  - 1D
  - 2D
  - 3D
  - 4D
A scalable topology that has been used in several parallel processors is the hypercube,
- a line connecting two nodes defines a 1-dimensional ``cube.''
- a square with four nodes is a 2-dimensional cube, and a 3D cube has eight nodes
- the rule for constructing an \( n \)-dimensional cube:
  - begin with an \((n-1)\)-dimensional cube, make an identical copy, and add links from each node in the original to the corresponding node in the copy
  - doubling the number of nodes in a hypercube increases the degree by only 1 link per node, and likewise increases the diameter by only 1 path

Communication in a hypercube is based on the binary representation of node IDs.
- Node IDs are the basis for a simple algorithm for routing information in a hypercube
  - An \( n \)-dimensional cube will have \( n \)-bit node IDs
  - Sending a message from nodes A to B can be done in cycles, where on each cycle a node will either hold a message or forward it along one of its links
• **Node Symmetry** is a desirable property of interconnection networks
  – “a node symmetric network” has no distinguished node, that is, the `view` of the rest of the network is the same from any node
  – rings, fully connected networks, and hypercubes are all node symmetric
  – trees and stars, shown, are not.
    • a tree has three different types of nodes: a root, interior, and leaf nodes, each with a different degree
    • a star has a distinguished node in the center which is connected to every other node
  – when a topology is node asymmetric, a distinguished node can become a communications bottleneck

• **Tree and Star**
A more formal definition of a communication bottleneck is based on a property known as the *bisection width* or *minimum cut*

- the minimum number of links that must be cut in order to divide the topology into two independent networks
- bisection width of a tree is 1, since if either link connected to the root is removed the tree is split into two subtrees
- bandwidth is useful in defining worst-case performance of algorithms on a particular network, since it is also related to the cost of moving data from one side of the system to the other
A planar (2D) mesh is a common topology – basically a matrix of nodes, each with connections to its nearest neighbors.

Usually have "wraparound" connections – nodes at the top of the grid have an "up" link that connects to nodes at the bottom of the grid.

Multistage network systems have processors and memories on the network edges, and switching elements for interior nodes – to send information from one edge to another, the interior switches are configured to form a path that connects nodes on the edges – the size and number of interior nodes contributes to the path length for each communication – often a "setup time" involved when a message arrives at an interior node and the switch decides how to configure itself in order to pass the message.
Crossbar Switch

• Example of a multistage network

• Typically, a column of processors on the left edge and a row of memories on the bottom edge
  – the switch configures itself dynamically to connect a processor to a memory module
  – no contention, if each processor wants to communicate with a different memory
  – if two or more processors need to access the same memory, one will be blocked until the switch reconfigures itself
  – a crossbar has a short diameter - information needs to pass through only one switching element on a path from one edge to another
  – poor scalability: If there are \( n \) processors and a like number of memories there are \( n^2 \) interior switches. Adding another processor and memory means adding another \( 2n - 1 \) interior nodes.

The dark circle indicate closed switches. A processor will attempt to make at most one connection at a time, and each column can have only one connection at a time. If the processors try to connect to different memories none will be blocked.
Like the crossbar switch, configurations of the butterfly can allow each processor to connect to a different memory
  - so no requests are blocked

The butterfly is not as flexible as the crossbar
  - since combinations of requests that are nonblocking in the crossbar are blocking in the butterfly
  - e.g. if the first switch in the first column is in the straight-through configuration because processor \( P_0 \) is making a request to memory \( M_2 \), processor \( P_1 \) is constrained to communicate with memories 4 through 7 (100\(_2\) through 111\(_2\)). With a crossbar \( P_1 \) would be allowed to connect to \( M_1 \), \( M_2 \), or \( M_3 \) without blocking

An interior node is a switch that is configured to pass data straight through (top) or to the opposite side (bottom).
IBM RS/6000 SP

Premier parallel-processing computer for complex and demanding applications - everything from financial modeling, computational performance and significantly improved price/performance

fluid dynamics and numerical analysis to data mining and decision support, LAN server consolidation and internal disk mirroring and hardware and software monitoring up to 16 SP nodes can be supported by HACMP for AIX (High Availability Cluster)

Multi-Processing) - one of the industry's leading software products for critical application backup and availability.

In addition, the system can be partitioned into pools of nodes (e.g., two nodes might be pooled to work as a single node and prevents application failure. Lotus Notes server, while ten others process a parallel database). The SP system can also scale processors and memory, making access to terabytes of data possible and expansions or upgrades easier to manage.

The high availability is achieved through built-in redundancy, subsystem recovery components, and failure tolerant computations. The SP system can also scale through partitioning, providing greater flexibility and performance.

New 332MHz PowerPC 604e-based SMP Nodes - provide increased performance and significantly improved price/performance

Support for up to 128 total nodes per system (512 nodes by special request)
• The following example shows the potential of this style of computing. This code from a computer graphics application rotates and displaces a set of \((x,y)\) pairs by an angle \(\phi\) and displacement \((x_d, y_d)\):

\[
\begin{align*}
    x'_i &= x_i \cos \phi - y_i \sin \phi + x_d \\
    y'_i &= y_i \cos \phi - x_i \sin \phi + y_d
\end{align*}
\]

• A vector processor would load the \((x,y)\) pairs into two vector registers and then use vector instructions. On the RS/6000 the operations are compiled into the following loop (constants \(x_d \sin \phi\), \(\phi\), etc. are loaded into registers before the loop begins):

```
L: load  R8,x[i]
    fma   R10,R8,cos,xd
    load R9,y[l]
    fma   R11,R9,cos,yd
    fma   R12,R9,-sin,R10
    store R12,x[i]'
    fma R13,R8,sin,R11
```
The nodes are numbered so that two nodes are adjacent if and only if the binary representations of their IDs differ by one bit. For example, nodes 0110 and 0100 are immediate neighbors but 0110 and 0101 are not. An easy way to label nodes is to assign node IDs as the cube is constructed. When you copy an \((n-1)\)-dimensional cube, make sure the corresponding nodes in the two copies have the same IDs. Then extend all the IDs by one bit. Append a 0 to the IDs of nodes in the original cube, and append a 1 to the IDs of nodes in the copy. As an example the nodes in the 1D and 2D cubes in Figure 11 are labeled according to this scheme; the labeling of the 3D and 4D cubes is left for an exercise.

On cycle \(i\) the node that currently holds the message will compare bit \(i\) of its own ID with bit \(i\) of the destination ID. If the bits match, the node holds the message. If they don't match, it forwards the message along dimension \(i\).
• If you visualize only north-south links in a rectangular mesh, you can see these links turn the 2D mesh into a 3D cylinder. Now if the east-west links are added, it connects the ends of the cylinder to form a toroidal solid. Thus a mesh topology with wraparound connections is often referred to as a torus. In many systems the wraparound connections are skewed by one or more rows (or columns, or both); in this case the topology is known as a twisted torus. Note that a path that starts in the northwest corner of a twisted torus and heads continually east will visit every node exactly once before returning to the northwest corner.
A banyan network is a multistage switching network that has the same number of inputs as outputs and interior nodes that are $m \times m$ switches.

Examples of banyan networks are butterfly networks and omega networks, which are both built from $2 \times 2$ switches. The diameter of a butterfly is $\log_2 n$, where $n$ is the number of inputs and outputs, and there are $O(n \times \log_2 n)$ switches, so these networks scale more efficiently than a crossbar (Figure 15). The $2 \times 2$ switch in a butterfly can be configured in one of two states (Figure 15). One configuration connects input 0 to output 0 and input 1 to output 1. The other configuration flips the outputs, so input 0 connects to output 1 and input 1 connects to output 0. The switching network uses the binary representation of the destination address in order to construct a path from input to output. The switch at stage in the network uses bit to determine how to configure itself: if the bit is 0, the request should go through the top output, and if it is 1 it should go through the bottom output.
Pipelines

- **Assembly line analogy**
  - used in manufacturing to increase productivity
  - divide a complex operation into pieces that can be performed in parallel
  - separate "workers" implement successive steps along the assembly line
  - when an item finishes one step it is passed down the line to next step

- **Pipelines are used in two major areas**
  - instruction processing
  - arithmetic operations

- **A pipelined system satisfy the following requirements:**
  - repeatedly executes a basic function
  - a basic function must be divisible into independent stages that have minimal overlap
  - the complexity of the stages should be roughly similar

- **The number of stages is referred to as the depth of the pipeline**
Pipelines 2

• Example: floating point addition of two numbers of the form $m \times 2^e$
  – breakdown of this function into stages as follows:
    1. If $e_2 < e_1$ swap the operands. Find the difference in exponents.
       $e_d = e_2 - e_1$.
    2. Shift $m_2$ to the right by $e_d$ bits.
    3. Compute the mantissa of the sum by adding $m_1$ and $m_2$. The exponent of the sum is $e_1$.
    4. Normalize the sum.

• The extra complexity of a pipelined adder pays off when adding long sequences of numbers
  – Operations at each stage can be done on different pairs of inputs
    e.g. one stage can be comparing the exponents in one pair of operands at the same time another stage is adding the mantissas of a different pair of operands

• Important that the operands must be independent
  – e.g. the two instructions: $R2 = R0 + R1$
    $R4 = R3 + R1$
    • both instructions identify $R1$ as one of their inputs
    • potential conflict in stages two and three because stage two might need to shift the mantissa of $R1$ at the same time stage three needs to add the mantissa of $R0$ to the mantissa of $R1$. 
Pipelines 3

- The solution is to make copies of the operands, and pass the copies through the pipeline.
- Thus the CPU gives the adder copies of R0 and R1 when it starts the pipeline for the first pair, and the second stage gets these copies from the first stage along with a value of $e_d$.
  
  - Another potential problem
    - one of second instruction operands is R2:
      
      $\begin{align*}
      R2 &= R0 + R1 \\
      R4 &= R2 + R1
      \end{align*}$
      
      - the second instruction depends on the result of the first instruction
      - CPU cannot send the second pair of operands to the pipelined adder until the result of the first addition exits the last stage
      - such interactions lead to time periods when the pipeline stages are empty which are called **bubbles**

- Gantt chart of cases from previous example
  
  - two successive instructions can be overlapped
    - first stage is busy with the second instruction while second stage is busy with the first instruction
    - in each successive cycle the two instructions are passed down the "assembly line" to the next stage
    - the second sum is done after 6 cycles
  
  - second instruction must wait for first to complete
    - second sum is not finished until after the 10th cycle
    - "bubble" in the pipeline is the 4-cycle dead period in each stage caused by delaying the second instruction
Pipelined Floating Point Adder

Case 1: two independent instructions, fully overlapped. The second is computed at the end of the 6th cycle.

Stage
(1) 
(2) 
(3) 
(4) 
(5) 

Cycle
$\text{Cycle } t_0 \ t_1 \ t_2 \ t_3 \ t_4 \ t_5 \ t_6$

Case 2: the second instruction must wait until the first is done, introducing a 4-cycle bubble.

Stage
(1) bubble 
(2) 
(3) 
(4) 
(5) 

Cycle
$\text{Cycle } t_0 \ t_1 \ t_2 \ t_3 \ t_4 \ t_5 \ t_6 \ t_7 \ t_8 \ t_9 \ t_{10}$
Pipelines 4

• In general a pipeline of depth \( d \) can process \( n \) items in \( n+d \) steps when there are no bubbles
  – without a pipeline, each application of the basic function would require \( d \) cycles, and they would have to be executed sequentially, for a total time of \( n*d \) cycles
  – the speedup obtained by a full pipeline is:

\[
\frac{n*d}{n+d}
\]

• When \( n >> d \), can safely ignore the \( d \) in the denominator, so the asymptotic speedup, observed for large \( n \), is a factor of \( d \).
  – e.g. to add 1000 pairs of numbers; at 5 cycles per addition, a machine without a pipelined adder would require 5000 cycles
  – with a 5-stage pipelined adder, the last sum will appear after \( 1000 + 5 \) cycles, so the pipeline is \( 5000/1005 = 4.97 \) times faster

• Providing a steady stream of independent operands that will keep a pipeline full is the distinguishing feature of a vector processor
  – can initiate such a series of operations with a single instruction
Pipelines 5

- Many possible sources of bubbles in pipelines
  - dependencies between instructions are the main cause
  - for arithmetic pipelines, *data dependencies* arise when pairs of operations share inputs and outputs

- Note that the instructions do not have to be consecutive to present dependencies
  - a compiler that checks for dependencies and reorders instructions has to "look ahead" in the code by an amount equal to the depth of the pipeline

- Instruction pipelines used to speed up the fetch-decode-execute cycle are susceptible to bubbles
  - *control dependencies* are most commonly caused by branch or loop instructions
    - if the pipeline is "looking ahead" and fetching instructions it thinks the machine will execute, but the machine branches to another location, a bubble is introduced while the fetch stage goes to get the instructions at the new location
Pipelines 6

• Pipelines have been widely used in high performance machines for many years
  – the CDC 6600 is a classic early example of a CISC pipeline, used a "scoreboard" circuit to detect data dependencies between instructions and instruction buffers
  – the Cray-1 had every data processing unit was pipelined
  – until the late 1980s pipelining was one of the attributes that separated "mainframes" and supercomputers from microprocessors
  – with VLSI technology most microprocessors now have room on chip for the complex control circuitry and instruction scheduling logic associated with pipelined data processing units

CDC 6600
First computer to be termed a "supercomputer", it was introduced in 1966. Later model CDC 6600s had a peak performance rate of 3 Megaflops. It was the first machine with extensive internal parallelism and utilized a run-time controlled multifunction processor which attempted to increase performance by reorganizing code.
Vector Processors

- A processor that can operate on entire vectors with one instruction, i.e. the operands of some instructions specify complete vectors.
  - e.g. \( C = A + B \)
    - in both scalar and vector machines this means 
      \`
      \text{``add the contents of } \ A \text{ to the contents of } \ B \text{ and put the sum in } \ C.\text{''}
      \`
    - in a scalar machine the operands are numbers,
    - in vector processors the operands are vectors and the instruction directs the machine to compute the pairwise sum of each pair of vector elements
  - A processor register, “the vector length register”, tells the processor how many individual additions to perform when it adds the vectors

- A vectorizing compiler is a compiler that will try to recognize when loops can be transformed into single vector instructions
  - e.g. the following loop can be executed by a single instruction on a vector processor

\[
\begin{align*}
\text{DO 10 } & \text{I=1,N} \\
A(\text{I}) & = B(\text{I}) + C(\text{I}) \\
10 & \text{CONTINUE}
\end{align*}
\]
Vector Processors 2

- the code is translated into an instruction that would set the vector length to N followed by a vector add instruction

- vector instructions pays off in two different ways
  - first, the machine has to fetch and decode far fewer instructions, so the control unit overhead is greatly reduced and the necessary memory bandwidth is reduced
  - second, the instruction provides the processor with a regular source of data
    - the machine knows it will have to fetch $n$ pairs of operands which can be arranged in a regular pattern in memory and the processor can tell the memory system to start sending those pairs
    - with an interleaved memory, the pairs will arrive at a rate of one per cycle, at which point they can be routed directly to a pipelined data unit for processing

- Division of vector processors based on the way the instructions access their operands
  - in memory to memory organization, operands are fetched from memory and routed directly to the functional unit; results are streamed back out to memory as the operation proceeds.
  - in register to register organization, operands are first loaded into a set of vector registers; the vector operation then proceeds by fetching the operands from the vector registers and returning the results to a vector register
Vector Processors 3

• The advantage of memory to memory machines is the ability to process very long vectors
  – register to register machines must break long vectors into fixed length segments
  – this flexibility is offset by a relatively large startup time overhead (time between the initialization of the instruction and the first result from the pipeline)
  – long startup time on a memory to memory machine is a function of memory latency, which is longer than the time it takes to access a value in an internal register
  – once the pipeline is full, however, a result is produced every cycle

\[ T = S + aN \]

• A performance model for a vector processor:

where \( s \) is the startup time, \( N \) is the length of the vector and \( a \) is an instruction dependent constant, usually \( 1/2, 1 \) or \( 2 \)

• The memory-to-memory architecture
  – examples: CDC Cyber 200 family and the ETA-10
  – machines appeared in the mid 1970s after a long development cycle and with dated technology
  – they disappeared in the mid 1980s
  – one reasons for their demise was the large startup time, which was on the order of 100 processor cycles
  – very inefficient for short vector operations
  – moderate (half max) performance for 100 length vectors
Vector Processors 4

- In register to register machines the vectors have a relatively short length, 64 in the case of the Cray family, but the startup time is far less than memory to memory machines
  - these machines are much more efficient for operations involving short vectors
  - long vector operations the vector registers must loaded with each segment before the operation can continue

- Register to register machines now dominate the vector computer market
  - offerings from Cray Research Inc., include the Y-MP and the C-90 and machines from Fujitsu, Hitachi and NEC
  - clock cycles on modern vector processors range from 2.5ns (NEC SX-3) to 4.2ns (Cray C90)
  - single processor performance on LINPACK benchmarks is in the range of 1 to 2 GFLOPS

The CRAY 1

The first major successful supercomputer was the CRAY-1 in 1976. It was designed by Seymour R. Cray who left Control Data Corporation to start his own company in 1972. The CRAY 1 had a top speed of 100 megaflops. If you tried to build one yourself using PCs, it would take 200 of them all cross connected, or you could just use 33.33 Sun4s. CRAY made at least 16 CRAY 1’s. In 1976, a typical CRAY 1 cost about $700,00.
Vector Processors 5

- **Vector Chaining** was a feature introduced in the Cray computers
  - e.g. two vector instructions:
    \[
    \begin{align*}
    V_2 &= V_0 \times V_1 \\
    V_4 &= V_2 + V_3
    \end{align*}
    \]
  - the output of the first instruction is one of the operands of the second instruction
  - since these are vector instructions, the first instruction will route up to 64 pairs of numbers to a pipelined multiplier
  - about midway through the execution
    - the first few elements of \(V_2\) will contain recently computed products; the products that will eventually go into the next elements of \(V_2\) are still in the multiplier pipeline; and the remainder of the operands are still in \(V_0\) and \(V_1\), waiting to be fetched and routed to the pipeline
This situation is where the operands from V0 and V1 that are currently in the multiplier pipeline are indicated by gray cells. At this point, the system is fetching V0[k] and V1[k] to route them to the first stage of the pipeline and V2[j] is just leaving the pipeline. Vector chaining relies on the path marked with an asterisk. While V2[j] is being stored in the vector register, it is also routed directly to the pipelined adder, where it is matched with V3[j]. As the figure shows, the second instruction can begin even before the first finished, and while both are executing the machine is producing two results per cycle (V4[i] and V2[j]) instead of just one.
Vector Processors 6

• Without vector chaining, the peak performance of the Cray-1 would have been 80 MFLOPS
  – one full pipeline producing a result every 12.5ns, or 80,000,000 results per second
  – with three pipelines chained together, there is a very short burst of time where all three are producing results, for a theoretical peak performance of 240 MFLOPS
    • (listed as 160 MFLOPS, because it was realistic to keep only two pipelines chained together for any reasonable period of time

• Vector chaining could be implemented in a memory-to-memory vector processor, but it would require much higher memory bandwidth to do so
  – without chaining, three channels must be used to fetch two input operand streams and store one result stream; with chaining, five channels would be needed for three inputs and two outputs
  – the ability to chain operations together to double performance gave register-to-register designs another competitive edge over memory-to-memory designs
Superscalar Processors

- Architectural concepts pioneered in vector processors and mainframe computers of the 1970s started to appear in RISC processors
  - VLSI technology made more room on the chip
  - rather than increase complexity of the architecture, most designers improved the execution of their current architecture
    - on-chip caches
    - instruction pipelines

- Latest step in the evolution is the *superscalar processor*, a scalar processor that is capable of executing more than one instruction in each cycle
  - in many superscalar designs, the high level architecture is unchanged from earlier scalar designs, but uses instruction level parallelism for improved implementation of these architectures

- Keys to superscalar execution are:
  - an instruction fetching unit that can fetch more than one instruction at a time from cache
  - instruction decoding logic that can decide when instructions are independent and thus executed simultaneously
  - sufficient execution units to be able to process several instructions at one time
Superscalar Processors 2

- The IBM RS/6000 is a good example with three major subsystems:
  - the instruction fetch unit
    - a 2-stage pipeline;
      - in the first stage a packet of four instructions is fetched from an instruction cache
      - in the second stage instructions are routed to the integer processor and/or floating point processor
    - this unit also executes branch instructions itself so that in a tight loop there is effectively no overhead from branching since the instruction unit executes branches while the data units are computing values
  - an integer processor
    - has a four-stage pipeline
    - in addition to executing data processing instructions this unit does some preprocessing for the floating point unit
  - a floating point processor
    - has a six stage pipeline

- Advantage of a superscalar processor is that it does not rely on a vectorizing compiler to detect loops and turn them into vector instructions
  - still requires a sophisticated compiler to allocate resources and schedule operations to best take advantage of the resources of the machine
  - but in the long run the superscalar approach is more flexible and applicable to a wider range of applications than vector processing
Topology

- The pathways over which the processors, memories, and switches communicate is a major consideration in parallel systems design
  - defines the *interconnection network*, or *topology*,
  - determines how processors will share data and at what cost

- Ring vs. Fully Connected Network

- Collection of *nodes* that communicate via *links*
  - nodes can be either processors, memories, or switches
  - links will always be point-to-point data paths, i.e. not buses that are shared by several nodes
  - a link connects two *neighbor* nodes
  - the *degree* of a node is the number of its neighbors
  - the *diameter* of a network is the longest path between any two nodes
    - a ring of *n* nodes has diameter *n/2*
    - a fully connected network has a fixed diameter of 1
Topology 2

• The diameter of a ring grows as more nodes are added
  – the diameter of a fully connected network remains the same
  – a ring can expand indefinitely without changing the degree, but each time a new node is added to a fully connected network a link has to be added to each existing node

• Scalability refers to the increase in the complexity of communication as more nodes are added
  – in a highly scalable topology more nodes can be added without severely increasing the amount of logic required to implement the topology and without increasing the diameter

To construct a n-dimensional cube, copy an (n-1)-dimensional cube, the connect corresponding nodes in the original and the copy. (Nodes from the original are dark.)

• Hypercubes

<table>
<thead>
<tr>
<th>Dimension</th>
<th>1D</th>
<th>2D</th>
<th>3D</th>
<th>4D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>00</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A scalable topology that has been used in several parallel processors is the hypercube,
- a line connecting two nodes defines a 1-dimensional "cube."
- a square with four nodes is a 2-dimensional cube, and a 3D cube has eight nodes
- the rule for constructing an $n$-dimensional cube:
  - begin with an $(n-1)$-dimensional cube, make an identical copy, and add links from each node in the original to the corresponding node in the copy
  - doubling the number of nodes in a hypercube increases the degree by only 1 link per node, and likewise increases the diameter by only 1 path

Communication in a hypercube is based on the binary representation of node IDs.
- Node IDs are the basis for a simple algorithm for routing information in a hypercube
  - An $n$-dimensional cube will have $n$-bit node IDs
  - Sending a message from nodes A to B can be done in cycles, where on each cycle a node will either hold a message or forward it along one of its links
• **Node Symmetry** is a desirable property of interconnection networks
  – “a node symmetric network” has no distinguished node, that is, the “view” of the rest of the network is the same from any node
  – rings, fully connected networks, and hypercubes are all node symmetric
  – trees and stars, shown, are not.
    • a tree has three different types of nodes: a root, interior, and leaf nodes, each with a different degree
    • a star has a distinguished node in the center which is connected to every other node
  – when a topology is node asymmetric, a distinguished node can become a communications bottleneck

• Tree and Star
A more formal definition of a communication bottleneck is based on a property known as the *bisection width* or *minimum cut*

- the minimum number of links that must be cut in order to divide the topology into two independent networks
- bisection width of a tree is 1, since if either link connected to the root is removed the tree is split into two subtrees
- bandwidth is useful in defining worst-case performance of algorithms on a particular network, since it is also related to the cost of moving data from one side of the system to the other
A planar (2D) mesh is a common topology – basically a matrix of nodes, each with connections to its nearest neighbors. Usually have "wraparound" connections – nodes at the top of the grid have an "up" link that connect to nodes at the bottom of the grid.

Multistage network systems have processors and memories on the network edges, and switching elements for interior nodes – to send information from one edge to another, the interior switches are configured to form a path that connects nodes on the edges – the size and number of interior nodes contributes to the path length for each communication – often a "setup time" involved when a message arrives at an interior node and the switch decides how to configure itself in order to pass the message.
Crossbar Switch

- Example of a multistage network

- Typically, a column of processors on the left edge and a row of memories on the bottom edge
  - the switch configures itself dynamically to connect a processor to a memory module
  - no contention, if each processor wants to communicate with a different memory
  - if two or more processors need to access the same memory, one will be blocked until the switch reconfigures itself
  - a crossbar has a short diameter - information needs to pass through only one switching element on a path from one edge to another
  - poor scalability: If there are $n$ processors and a like number of memories there are $n^2$ interior switches. Adding another processor and memory means adding another $2n - 1$ interior nodes.

The dark circle indicate closed switches. A processor will attempt to make at most one connection at a time, and each column can have only one connection at a time. If the processors try to connect to different memories none will be blocked.
• Like the crossbar switch, configurations of the butterfly can allow each processor to connect to a different memory
  – so no requests are blocked

• The butterfly is not as flexible as the crossbar
  – since combinations of requests that are nonblocking in the crossbar are blocking in the butterfly
  – e.g. if the first switch in the first column is in the straight-through configuration because processor $P_0$ is making a request to memory $M_2$, processor $P_1$ is constrained to communicate with memories 4 through 7 ($100_2$ through $111_2$). With a crossbar $P_1$ would be allowed to connect to $M_1$, $M_2$, or $M_3$ without blocking

---

An interior node is a switch that is configured to pass data straight through (top) or to the opposite side (bottom).
IBM RS/6000 SP

Premier parallel-processing computer for complex and demanding applications - everything from financial modeling, computational performance and fluid dynamics and numerical analysis to data mining and decision support, LAN server consolidation.

The RS/6000 SP simultaneously brings the power of dozens of RISC processors to a computing problem, enhancing overall computational performance and throughput many times over that possible with traditional serial computing techniques. The basic processor architecture of the Cray supercomputers has changed little since the Cray-1 was introduced in 1976. There are eight 332MHz PowerPC 604e-based SMP Nodes - provide increased computational performance and significantly improved price/performance.

New 332MHz PowerPC 604e-based SMP Nodes - provide increased performance and significantly improved price/performance
Support for up to 128 total nodes per system (512 nodes by special request)

The high availability is achieved through built-in redundancy, subsystem recovery components, and internal disk mirroring and hardware and software monitoring up to 16 SP nodes can be supported by HACMP for AIX (High Availability Cluster Multi-Processing) - one of the industry's leading software products for critical application backup and availability. If a node failure or other error occurs, the system can execute a recovery script that transfers the work to another node and prevents application failure.

Lotus Notes server, while ten others process a parallel database. The SP system can also scale processors and memory, making access to terabytes of data possible and expansions or upgrades easier to manage.

In addition, the system can be partitioned into pools of nodes (e.g., two nodes might be pooled to work as a single node) andprevent application failure. The SP system can also scale processors and memory, making access to terabytes of data possible and expansions or upgrades easier to manage.

Support for up to 128 total nodes per system (512 nodes by special request)
Superscalar Processors 3

• The following example shows the potential of this style of computing. This code from a computer graphics application rotates and displaces a set of \((x,y)\) pairs by an angle \(\phi\) and displacement \((x_d,y_d)\):

\[
\begin{align*}
x'_i &= x_i \cos \phi - y_i \sin \phi + x_d \\
y'_i &= y_i \cos \phi - x_i \sin \phi + y_d
\end{align*}
\]

• A vector processor would load the \((x,y)\) pairs into two vector registers and then use vector instructions. On the RS/6000 the operations are compiled into the following loop (constants \(x_d, \sin \phi, \), etc. are loaded into registers before the loop begins):

```
L: load R8,x[i]
   fma R10,R8,cos,xd
load R9,y[l]
   fma R11,R9,cos,yd
   fma R12,R9,-sin,R10
   store R12,x[i]'
   fma R13,R8,sin,R11
```
• The nodes are numbered so that two nodes are adjacent if and only if the binary representations of their IDs differ by one bit. For example, nodes 0110 and 0100 are immediate neighbors but 0110 and 0101 are not. An easy way to label nodes is to assign node IDs as the cube is constructed. When you copy an \((n-1)\)-dimensional cube, make sure the corresponding nodes in the two copies have the same IDs. Then extend all the IDs by one bit. Append a 0 to the IDs of nodes in the original cube, and append a 1 to the IDs of nodes in the copy. As an example the nodes in the 1D and 2D cubes in Figure 11 are labeled according to this scheme; the labeling of the 3D and 4D cubes is left for an exercise.

• On cycle \(i\) the node that currently holds the message will compare bit \(i\) of its own ID with bit \(i\) of the destination ID. If the bits match, the node holds the message. If they don't match, it forwards the message along dimension \(i\).
• If you visualize only north-south links in a rectangular mesh, you can see these links turn the 2D mesh into a 3D cylinder. Now if the east-west links are added, it connects the ends of the cylinder to form a toroidal solid. Thus a mesh topology with wraparound connections is often referred to as a torus. In many systems the wraparound connections are skewed by one or more rows (or columns, or both); in this case the topology is known as a twisted torus. Note that a path that starts in the northwest corner of a twisted torus and heads continually east will visit every node exactly once before returning to the northwest corner.
A banyan network is a multistage switching network that has the same number of inputs as outputs and interior nodes that are $m \times m$ switches.

Examples of banyan networks are butterfly networks and omega networks, which are both built from $2 \times 2$ switches. The diameter of a butterfly is $\log_2 n$, where $n$ is the number of inputs and outputs, and there are $O(n \times \log_2 n)$ switches, so these networks scale more efficiently than a crossbar (Figure 15). The $2 \times 2$ switch in a butterfly can be configured in one of two states (Figure 15). One configuration connects input 0 to output 0 and input 1 to output 1. The other configuration flips the outputs, so input 0 connects to output 1 and input 1 connects to output 0. The switching network uses the binary representation of the destination address in order to construct a path from input to output. The switch at stage in the network uses bit $i$ to determine how to configure itself: if the bit is 0, the request should go through the top output, and if it is 1 it should go through the bottom output. For