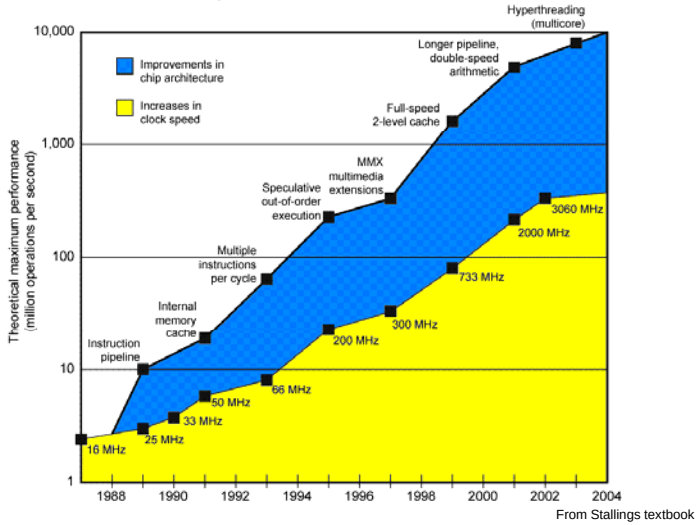
# Parallel Architectures

COMP375 Computer Architecture and Organization

## Ever Faster

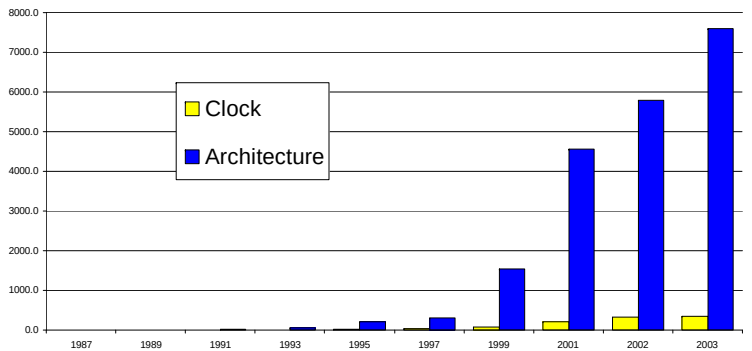
- There is a continuing drive to create faster computers.
- Some large engineering problems require massive computing resources.
- Increases in clock speed alone will not produce a sufficient increase in computer performance.
- Parallelism provides an ability to get more done in the same amount of time.

## Intel Microprocessor Performance

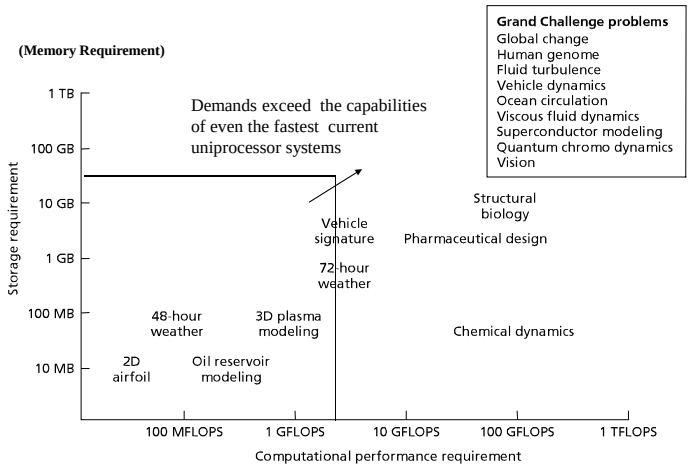



From Stallings textbook

## Intel Performance



**Large Computing Challenges**




from http://meseec.ce.rit.edu/eecc756-spring2002/756-3-8-2005.ppt

# Parallelism

- The key to making a computer fast is to do as much in parallel as possible.
- Processors in modern home PCs do a lot in parallel
  - Superscalar execution
  - Multiple ALUs
  - Hyperthreading
  - Pipelining

# Processes and Threads

- All modern operating systems support multiple processes and threads.
- Processes have their own memory space. Each program is a process.
- Threads are a flow of control through a process. Each process has at least one thread.
- Single processor systems support multiple processes and threads by sequentially sharing the processor.

# Hyperthreading

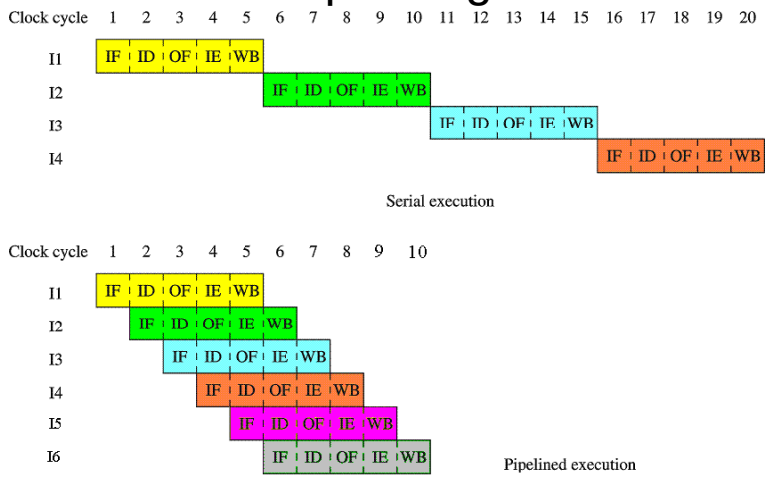
- Hyperthreading allows the apparently parallel execution of two threads.
- Each thread has its own set of registers, but they share a single CPU.
- The CPU alternates executing instructions from each thread.
- Pipelining is improved because adjacent instructions come from different threads and therefore do not have data hazards.

## Hyperthreading Performance

- Also called Multiple Instruction Issue processors
- Overall throughput is increased
- Individual threads do not run as fast as they would without hyperthreading

Thread 1
Thread 2
Thread 1
Thread 2
Thread 1
Thread 2
Thread 1
Thread 2

## Pipelining




## Hyperthreading

1. Runs best in single threaded applications
2. Replaces pipelining
3. Duplicates the user register set
4. Replaces dual-core processors.

## How Visible is Parallelism?

- Superscalar
  - Programmer never notices
- Multiple Threads
  - Programmer must create multiple threads in the program.
- Multiple processes
  - Different programs
    - Programmer never notices
  - Parts of the same program
    - Programmer must divide the work among different processes.

## Flynn's Parallel Classification

- **SISD** – Single Instruction Single Data
  - – standard uniprocessors
- **SIMD** – Single Instruction Multiple Data
  - – vector and array processors
- **MISD** - Multiple Instruction Single Data
  - – systolic processors
- **MIMD** – Multiple Instruction Multiple Data

## MIMD Computers

- Shared Memory
  - – Attached processors
  - – SMP
    - Multiple processor systems
    - Multiple instruction issue processors
    - Multi-Core processors
  - – NUMA
- Separate Memory
  - – Clusters
  - – Message passing multiprocessors
    - Hypercube
    - Mesh

## Single Instruction Multiple Data

- Many high performance computing programs perform calculations on large data sets. SIMD computers can be used to perform matrix operations.
- The Intel Pentium MMX instructions perform SIMD operations on 8 one byte integers.

## Vector Processors

- A vector is a one dimensional array.
- Vector processors are SIMD machines that have instructions to perform operations on vectors.
- A vector process might have vector registers with 64 doubles in each register.
- A single instruction could add two vectors.

## Vector Example

```
/*  traditional vector addition */
double a[64], b[64], c[64];
for (i = 0; i < 64; i++)
    c[i] = a[i] + b[i];
```

/* vector processor in one instruction*/

$c_i = a_i + b_i$

## Non-contiguous Vectors

- A simple vector is a one dimensional array with each element in successive addresses
- In a two dimensional array, the rows can be considered simple vectors. The columns are vectors, but the are not stored in consecutive addresses.
- The distance between elements of a vector is called the “stride”.

## Vector Stride

- Matrix multiplication

$$\begin{bmatrix} a11, a12, a13 \\ a21, a22, a23 \end{bmatrix} * \begin{bmatrix} b11, b12 \\ b21, b22 \\ b31, b32 \end{bmatrix}$$

| | | | | | |
|---|---|---|---|---|---|
| $a11$ | $a12$ | $a13$ | $a21$ | $a22$ | $a23$ |

| | | | | | |
|---|---|---|---|---|---|
| $b11$ | $b12$ | $b21$ | $b22$ | $b31$ | $b32$ |

## Cray-1 Vector Processor

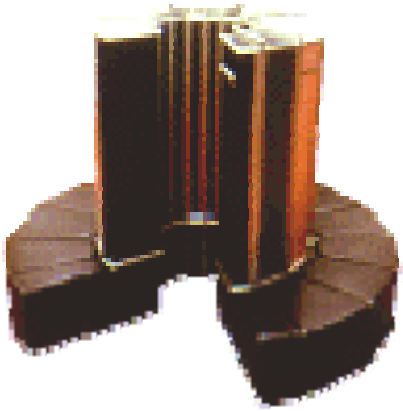
Sold to Los Alamos National Laboratory in1976 for $8.8 million

80 MHz clock

160 megaflops

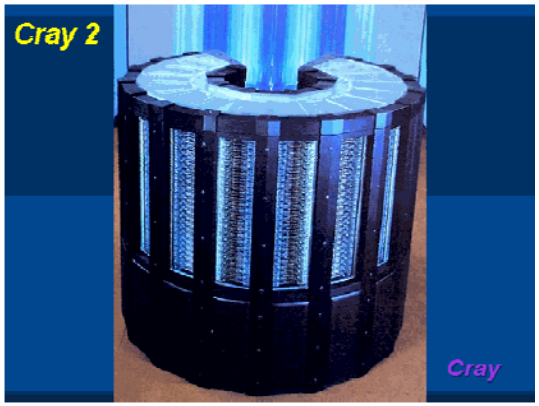
8 megabyte (1 M words)

Freon cooled



## Cray-2

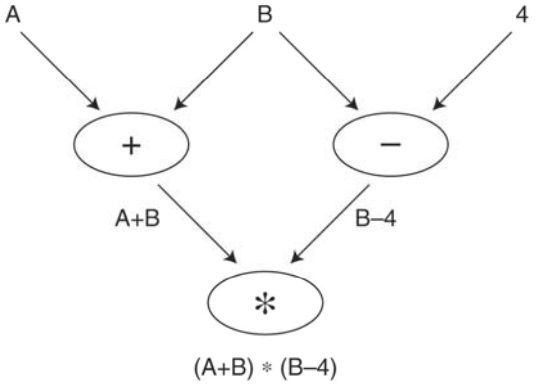
Built in 1985 with ten times the performance of the Cray-1




## Intel calls some of their new Pentium instructions SIMD because

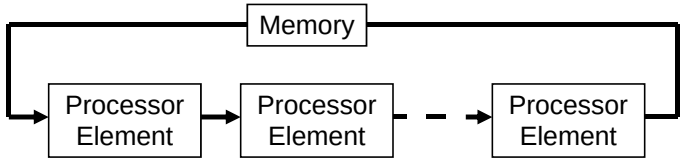
1. They operate on multiple bytes
2. There is a separate processor for them
3. Multiple processing units operate on one piece of data
4. They are executed in the GPU

## Data Flow Architecture




Data flow graph computing N =(A+B) * (B-4)

## MISD Systolic Architectures




- A computed value is passed from one processor to the next.
- Each processor may perform a different operation.
- Minimizes memory accesses.

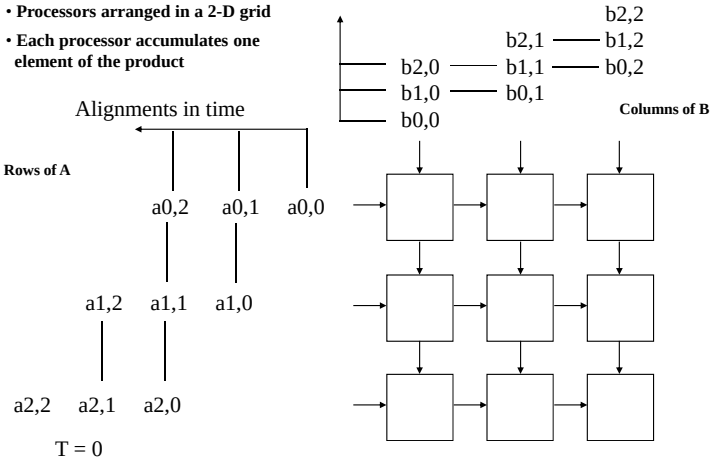
# Systolic Architectures

- Systolic arrays meant to be special-purpose processors or coprocessors and were very fine-grained
- Processors implement a limited and very simple computation, usually called cells
- Communication is very fast, granularity meant to be very fine (a small number of computational operations per communication)
- Very fast clock rates due to regular, synchronous structure
- Data moved through pipelined computational units in a regular and rhythmic fashion
- Warp and iWarp were examples of systolic arrays

from www-csag.ucsd.edu/teaching/cse160s05/lectures/Lecture15.pdf

## Systolic Array Example: 3x3 Systolic Array Matrix Multiplication

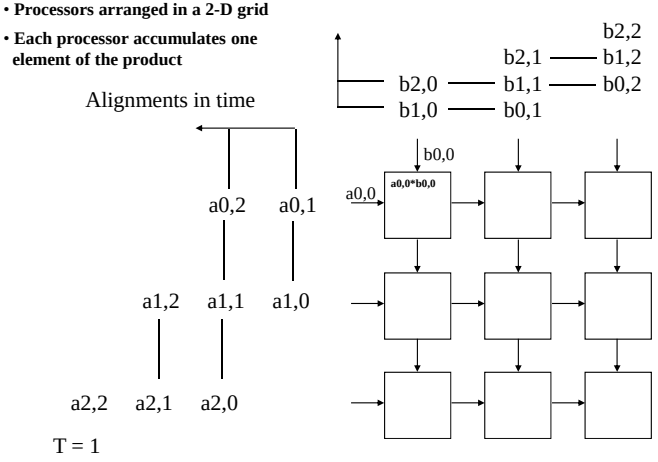
- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**




**Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/**

## Systolic Array Example: 3x3 Systolic Array Matrix Multiplication

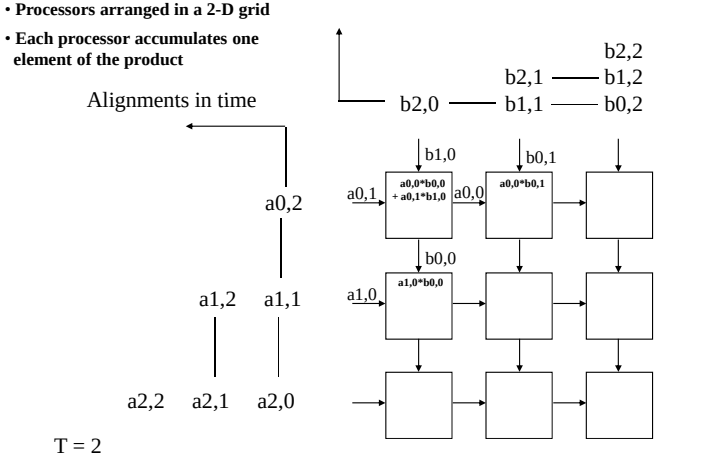
- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**




**Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/**

## Systolic Array Example: 3x3 Systolic Array Matrix Multiplication

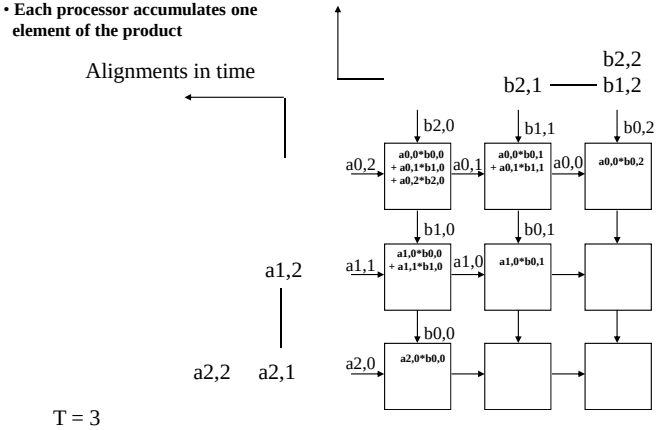
- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**




**Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/**

## Systolic Array Example: 3x3 Systolic Array Matrix Multiplication

- Processors arranged in a 2-D grid
- Each processor accumulates one element of the product

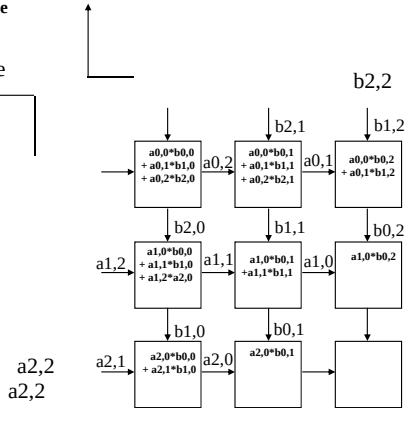



T = 3

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

## Systolic Array Example: 3x3 Systolic Array Matrix Multiplication

- Processors arranged in a 2-D grid
- Each processor accumulates one element of the product

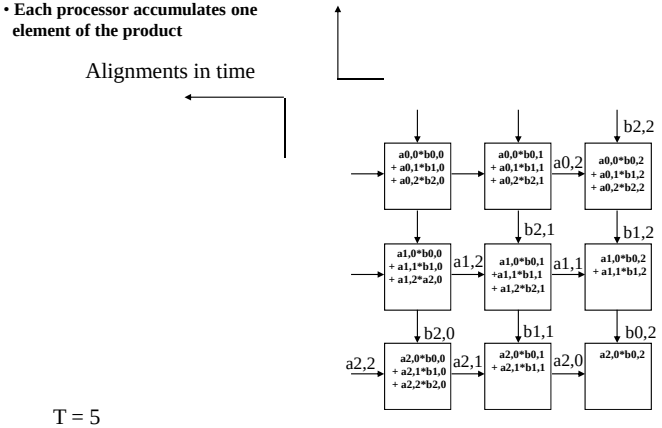



T = 4

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

## Systolic Array Example: 3x3 Systolic Array Matrix Multiplication

- Processors arranged in a 2-D grid
- Each processor accumulates one element of the product

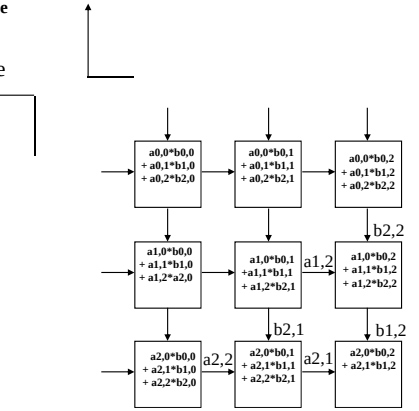



T = 5

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

## Systolic Array Example: 3x3 Systolic Array Matrix Multiplication

- Processors arranged in a 2-D grid
- Each processor accumulates one element of the product



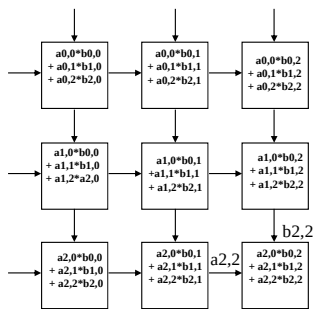

T = 6

Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/

**Systolic Array Example:**
**3x3 Systolic Array Matrix Multiplication**

- **Processors arranged in a 2-D grid**
- **Each processor accumulates one element of the product**

Alignments in time




Done

T = 7

**Example source: http://www.cs.hmc.edu/courses/2001/spring/cs156/**

## Systolic Architectures

- Never caught on for General-purpose computation
  - Regular structures are HARD to achieve
  - Real efficiency requires local communication, but flexibility of global communication is often critical
- Used mainly in structured settings
  - High speed multipliers
  - Structured Array Computations

from www-csag.ucsd.edu/teaching/cse160s05/lectures/Lecture15.pdf

## Multiple Instruction Multiple Data

- Multiple independent processors can execute different parts of a program to different programs.
- The processors can be different or identical.
- Different architectures allow the processors to access:
  - the same memory
  - separate memory
  - local memory with access to all memory

## Specialized Processors

- Some architectures have a general purpose processor with additional specialized processors
  - I/O processors
    - I/O controllers with DMA
    - Graphics processor
  - Floating point processors
    - Intel 80387 for Intel 80386
  - Vector processors

## Separate Memory Systems

- Each processor has its own memory that none of the other processors can access.
- Each processor has unhindered access to its memory.
- No cache coherence problems.
- Scalable to very large numbers of processors
- Communication is done by passing messages over special I/O channels

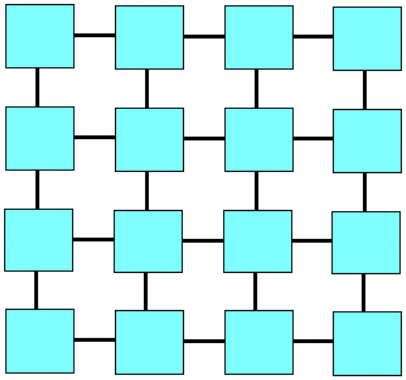
## Topologies

- The CPUs can be interconnected in different ways.
- Goals of an interconnection method are:
  - Minimize the number of connections per CPU
  - Minimize the average distance between nodes
  - Minimize the diameter or maximum distance between any two nodes
  - Maximize the throughput
  - Simple routing of messages

## Grid or Mesh

- Each node has 2, 3 or 4 connections
- Routing is simple and many routes are available.
- The diameter is

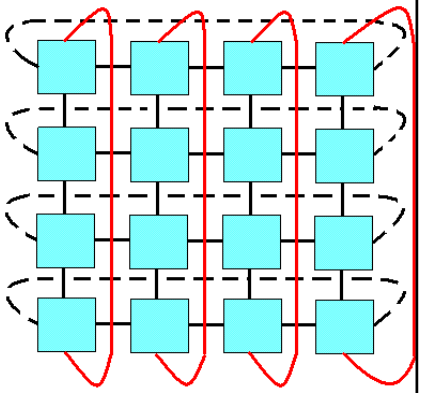
$$2\sqrt{N} - 2$$



## Torus

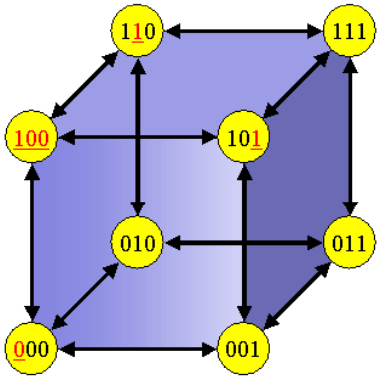
- Similar to a mesh with connected end nodes.
- Many similar routes available.
- The diameter is

$$\sqrt{N}$$



## Hypercube

- Each node $\log_2 N$ connections
- Routing can be done by changing one bit of the address at a time.
- The diameter is $\log_2 N$




## NUMA systems

- In **N**on**U**niform **M**emory **A**ccess systems each node has local memory but can also access the memory of other nodes.
- Most memory access are to local memory.
- The memory of other nodes can be accessed, but the access is slower than local memory.
- Cache coherence algorithms are used when shared memory is accesses.
- More scalable than SMP.

## NUMA system