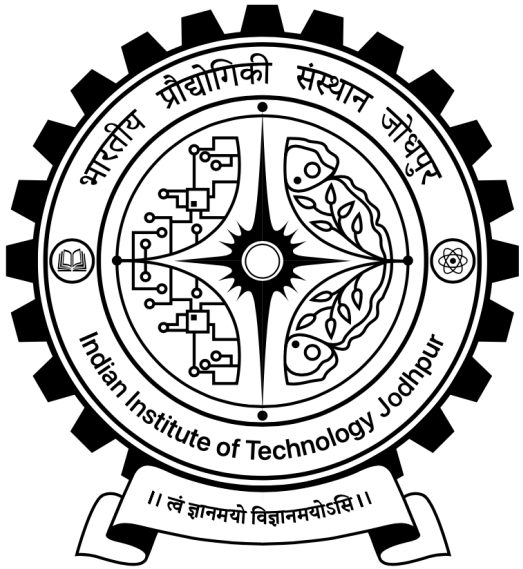
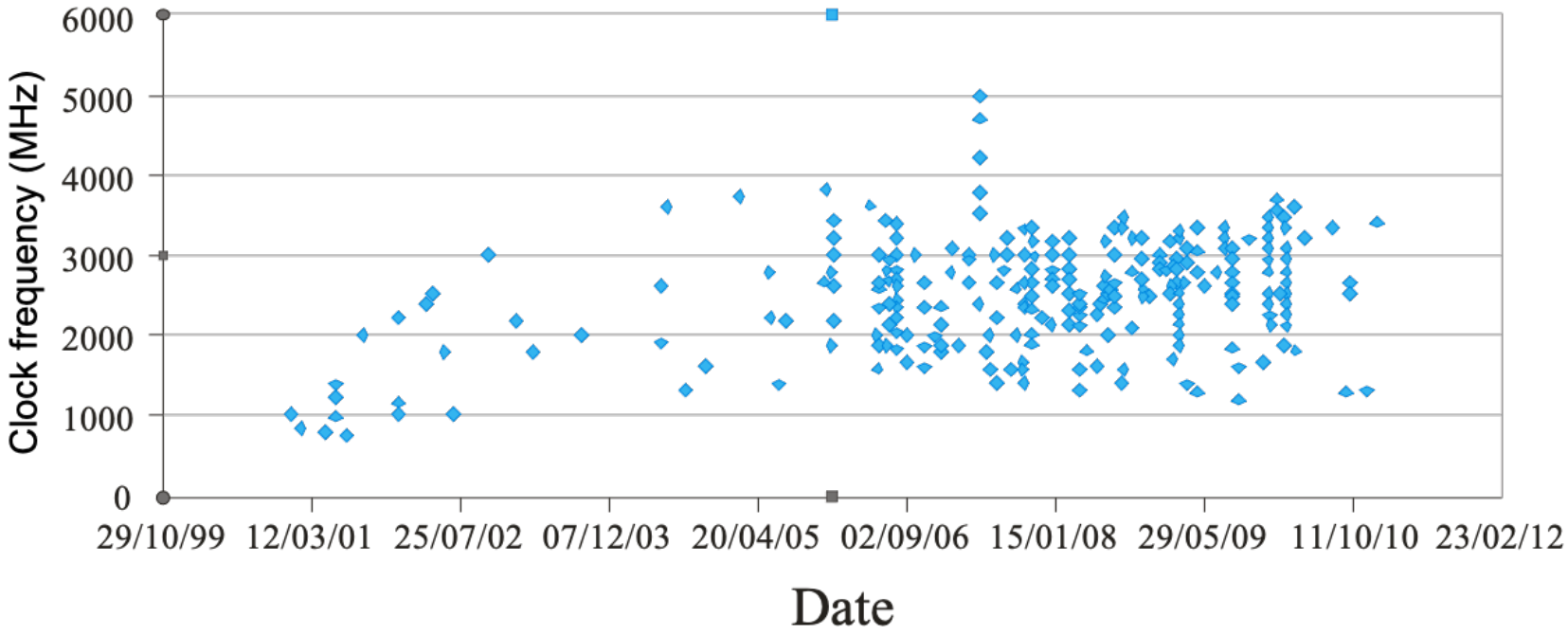



# Computer Architecture
# CSL3020

Deepak Mishra

http://home.iitj.ac.in/~dmishra/
Department of Computer Science and Engineering
Indian Institute of Technology Jodhpur

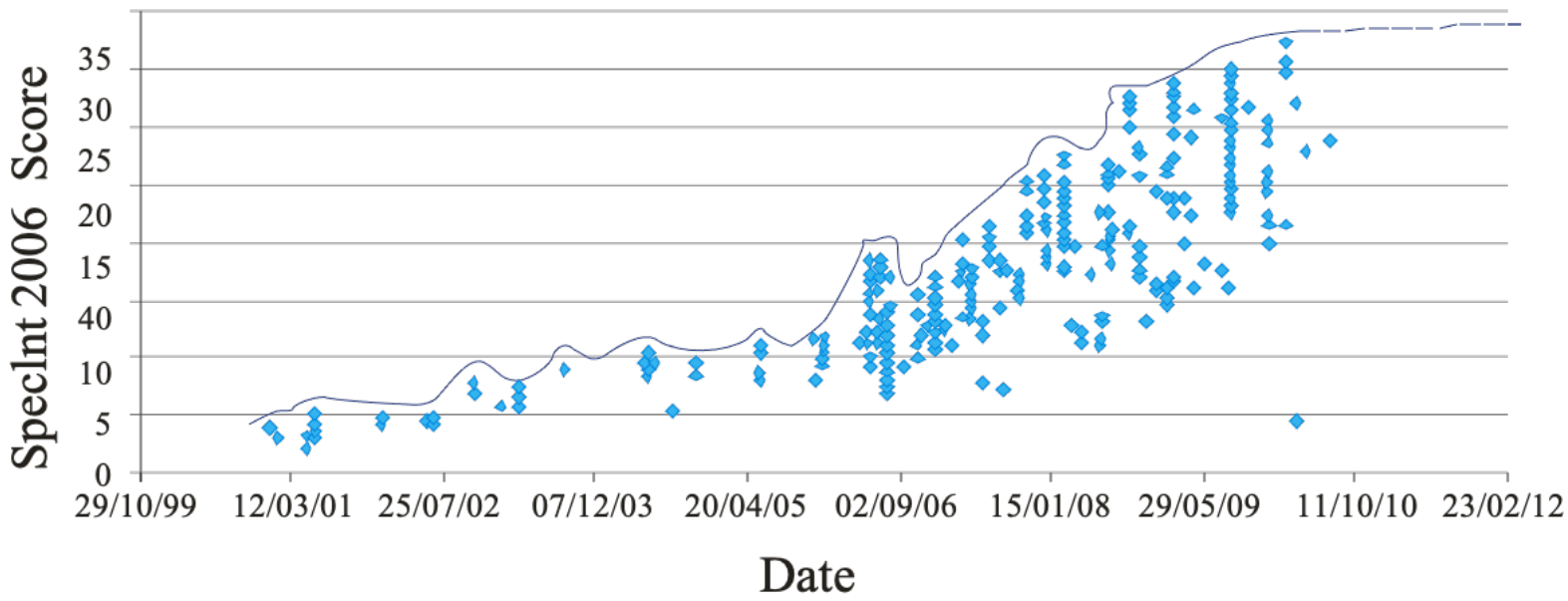
# Multicore Architecture




Clock frequency saturation.

Source: Basic Computer Architecture by SR Sarangi

# Multicore Architecture




Performance saturation.

Source: Basic Computer Architecture by SR Sarangi

# Multicore Architecture

*Multiprocessing:* It refers to multiple processors working in parallel.

- This is a generic definition, and it can refer to multiple processors in the same chip, or processors across different chips.

# Multicore Architecture

*Multiprocessing:* It refers to multiple processors working in parallel.

- This is a generic definition, and it can refer to multiple processors in the same chip, or processors across different chips.
- A multicore processor is a specific type of multiprocessor that contains all of its constituent processors in the same chip. Each such processor is known as a **core**.

# Amdahl's law

*Amdahl's law:* A rule stating that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used.

# Amdahl's law

*Amdahl's law:* A rule stating that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used.

In ideal scenario we may expect a speedup of $P$ for $P$ parallel processors.

# Amdahl's law

*Amdahl's law:* A rule stating that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used.

In ideal scenario we may expect a speedup of $P$ for $P$ parallel processors.

According to Amdahl's law the achieved speedup would be

$$S = \frac{1}{(1 - f_{par}) + \frac{f_{par}}{P}}$$

where $f_{par}$ is the parallel portion of the program.

# Amdahl's law

*Amdahl's law:* A rule stating that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used.

# Amdahl's law

*Amdahl's law:* A rule stating that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used.

In ideal scenario we may expect a speed-up of $P$ for $P$ parallel processors.

# Amdahl's law

*Amdahl's law:* A rule stating that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used.

In ideal scenario we may expect a speed-up of $P$ for $P$ parallel processors.

According to Amdahl's law the achieved speed-up would be

$$S = \frac{1}{(1 - f_{par}) + \frac{f_{par}}{P}}$$

where $f_{par}$ is the parallel portion of the program.

# Amdahl's law

Example: Suppose you want to perform two sums – one is a sum of 10 scalar variables, and one is a matrix sum of a pair of two-dimensional arrays, with dimensions 10 by 10. What speed-up do you get with 10 versus 100 processors? Next, calculate the speed-ups assuming the matrices grow to 100 by 100.

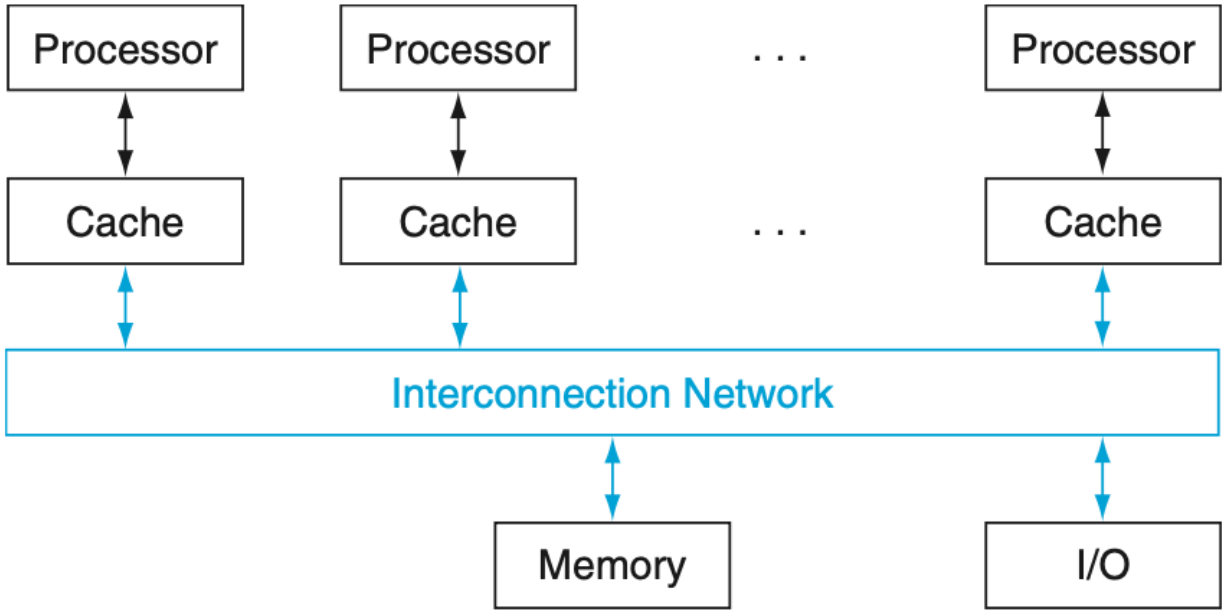
# Amdahl's law

Example: Suppose you want to perform two sums – one is a sum of 10 scalar variables, and one is a matrix sum of a pair of two-dimensional arrays, with dimensions 10 by 10. What speed-up do you get with 10 versus 100 processors? Next, calculate the speed-ups assuming the matrices grow to 100 by 100.

Ans. 5.5, 10

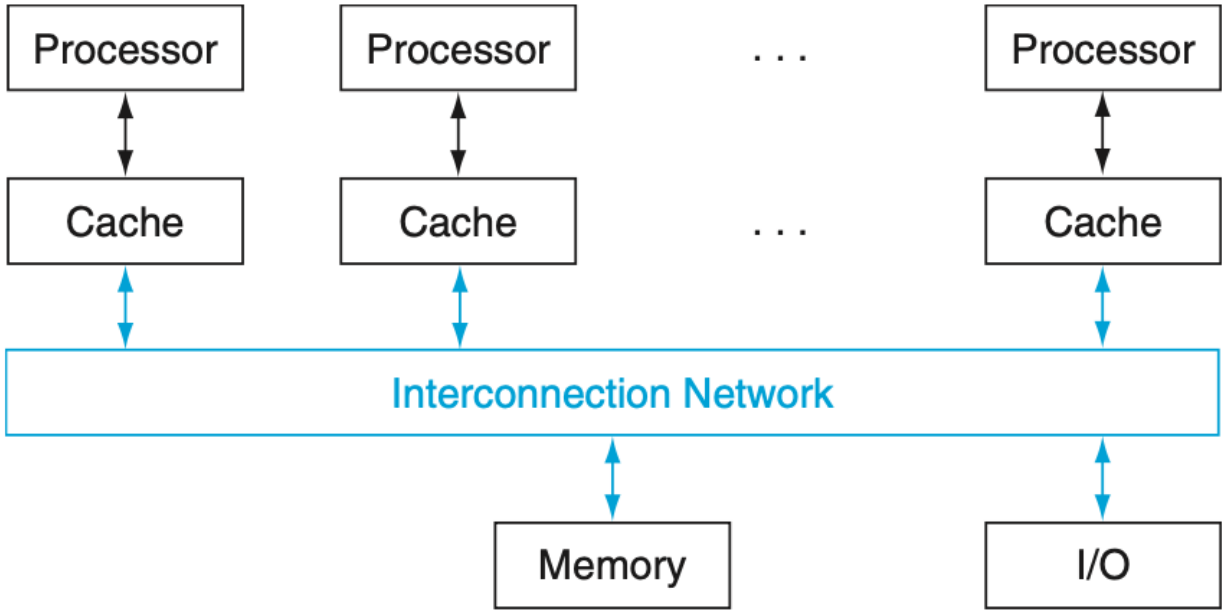
# Shared Memory Multiprocessors

A shared memory multiprocessor (SMP) is one that offers the programmer a single physical address space across all processors.



# Shared Memory Multiprocessors

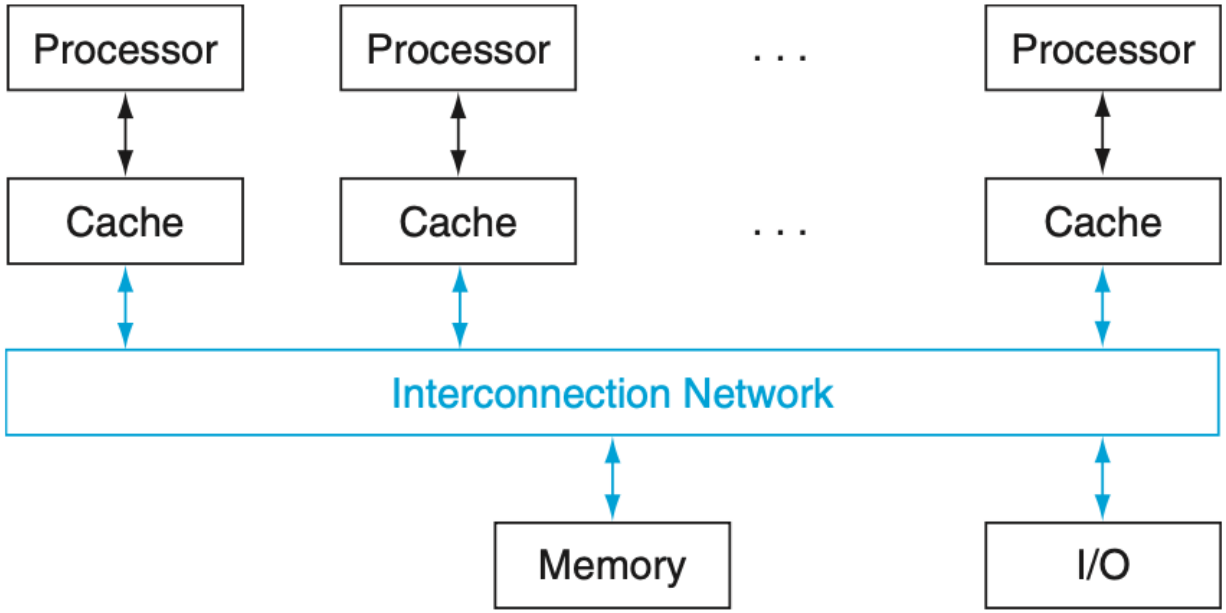
A shared memory multiprocessor (SMP) is one that offers the programmer a single physical address space across all processors.




Processors communicate through shared variables in memory.

# Shared Memory Multiprocessors

A shared memory multiprocessor (SMP) is one that offers the programmer a single physical address space across all processors.

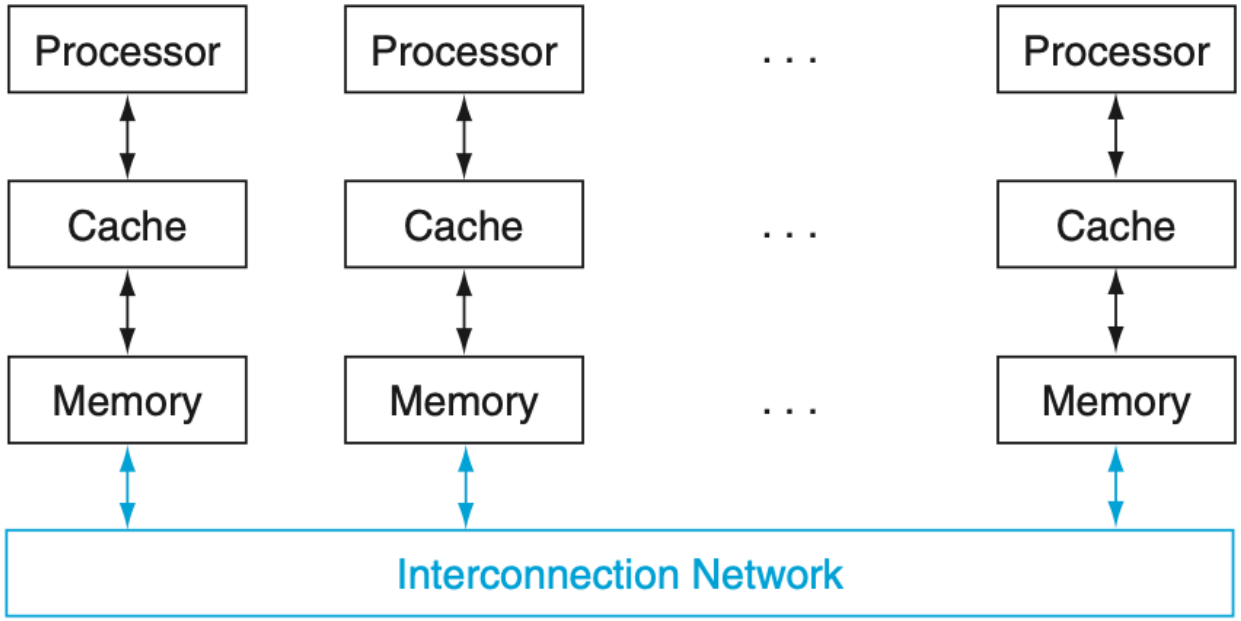



Processors communicate through shared variables in memory.

Uniform Memory Access (UMA) vs. Nonuniform Memory Access (NUMA)

# Message-Passing Multiprocessors

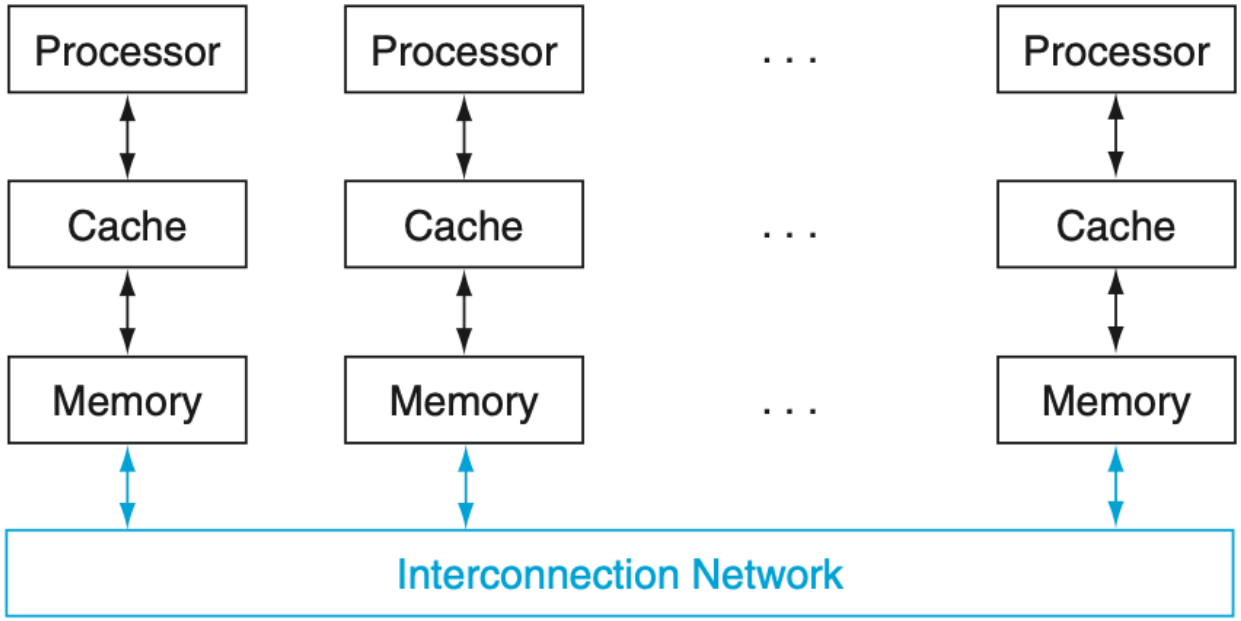
Each processor has its own private physical address space.




This configuration relies on explicit message passing, in which communication between multiple processors happens by explicitly sending and receiving information.

# Message-Passing Multiprocessors

Each processor has its own private physical address space.




This configuration relies on explicit message passing, in which communication between multiple processors happens by explicitly sending and receiving information.

High-performance computing (HPC) clusters commonly use message passing.

# Message-Passing Multiprocessors

Example: Suppose we want to sum 100,000 numbers in a message-passing multi processor with 100 processors, each with multiple private memories.

# Message-Passing Multiprocessors

Example: Suppose we want to sum 100,000 numbers in a message-passing multi processor with 100 processors, each with multiple private memories.

In the first step, The processor containing the 100,000 numbers sends the subsets to each of the 100 processor-memory nodes.

Example: Suppose we want to sum 100,000 numbers in a message-passing multi processor with 100 processors, each with multiple private memories.

In the first step, The processor containing the 100,000 numbers sends the subsets to each of the 100 processor-memory nodes.

The next step is to get the sum of each subset.

```
sum = 0;
for (i = 0; i<1000; i = i + 1) /* loop over each array */
 sum = sum + AN[i];  /* sum the local arrays */
```

The last step is the reduction that adds these 100 partial sums.

```
limit = 100; half = 100;/* 100 processors */
repeat
  half = (half+1)/2; /* send vs. receive dividing line*/
  if (Pn >= half && Pn < limit) send(Pn - half, sum);
  if (Pn < (limit/2)) sum = sum + receive();
  limit = half; /* upper limit of senders */
until (half == 1); /* exit with final sum */
```

# Multiple Instruction and Data Streams

- **SISD:** A standard uniprocessor.
- **SIMD:** Single Instruction, Multiple Data. Vector processor.
- **MISD:** Multiple Instruction, Single Data. Rarely used.
- **MIMD:** Multiple Instruction, Multiple Data. Generally used in Single Program Multiple Data (SPMD) form.

# Vector Processors

- A vector instruction operates on arrays of data.

Source: Basic Computer Architecture by SR Sarangi

# Vector Processors

- A vector instruction operates on arrays of data.
- There are vector instructions to add or multiply two arrays of data, and produce an array as output.

Source: Basic Computer Architecture by SR Sarangi

# Vector Processors

- A vector instruction operates on arrays of data.
- There are vector instructions to add or multiply two arrays of data, and produce an array as output.
- These operations form the core of many scientific programs, high intensity graphics, and data anaytics applications.

Source: Basic Computer Architecture by SR Sarangi

# Vector Processors

- A vector instruction operates on arrays of data.
- There are vector instructions to add or multiply two arrays of data, and produce an array as output.
- These operations form the core of many scientific programs, high intensity graphics, and data anaytics applications.
- MMX, SSE1, SSE2, SSE3, SSE4, and AVX instruction sets for x86 processors.

Source: Basic Computer Architecture by SR Sarangi

# Vector Processors

MIPS code for $Y = a \times X + Y$

```
l.d      $f0,a($sp)      ;load scalar a
lv       $v1,0($s0)      ;load vector x
mulvs.d  $v2,$v1,$f0     ;vector-scalar multiply
lv       $v3,0($s1)      ;load vector y
addv.d   $v4,$v2,$v3     ;add y to product
sv       $v4,0($s1)      ;store the result
```