MULTIPROCESSORS

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

Overview

- This lecture
  - Flynn’s taxonomy
  - Vector processing
  - Performance of parallel processing
  - Communication in multiprocessors

Flynn's Taxonomy

- Data vs. instruction streams

Data Stream	Instruction Stream: Single	Instruction Stream: Multiple
Single	Single-Instruction, Single Data (SISD) uniprocessors	Multiple-Instruction, Single Data (MISD) systolic arrays
Multiple	Single-Instruction, Multiple Data (SIMD) vector processors	Multiple-Instruction, Multiple Data (MIMD) multicores

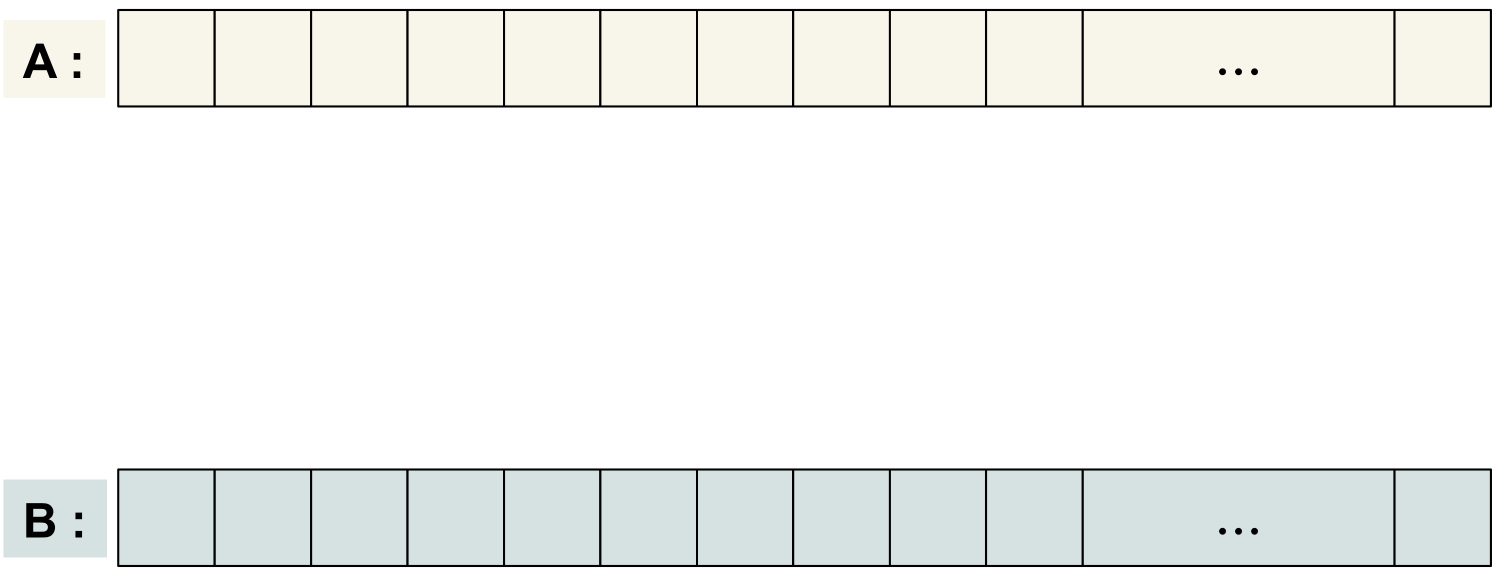
Data Level Parallelism

- Due to executing the same code on a large number of objects
  - Common in scientific computing
- DLP architectures
  - Vector processors—e.g., Cray machines
  - SIMD extensions—e.g., Intel MMX
  - Graphics processing unit—e.g., NVIDIA
- Improve throughput rather than latency
  - Not good for non-parallel workloads

Vector Processing

- Scalar vs. vector processor

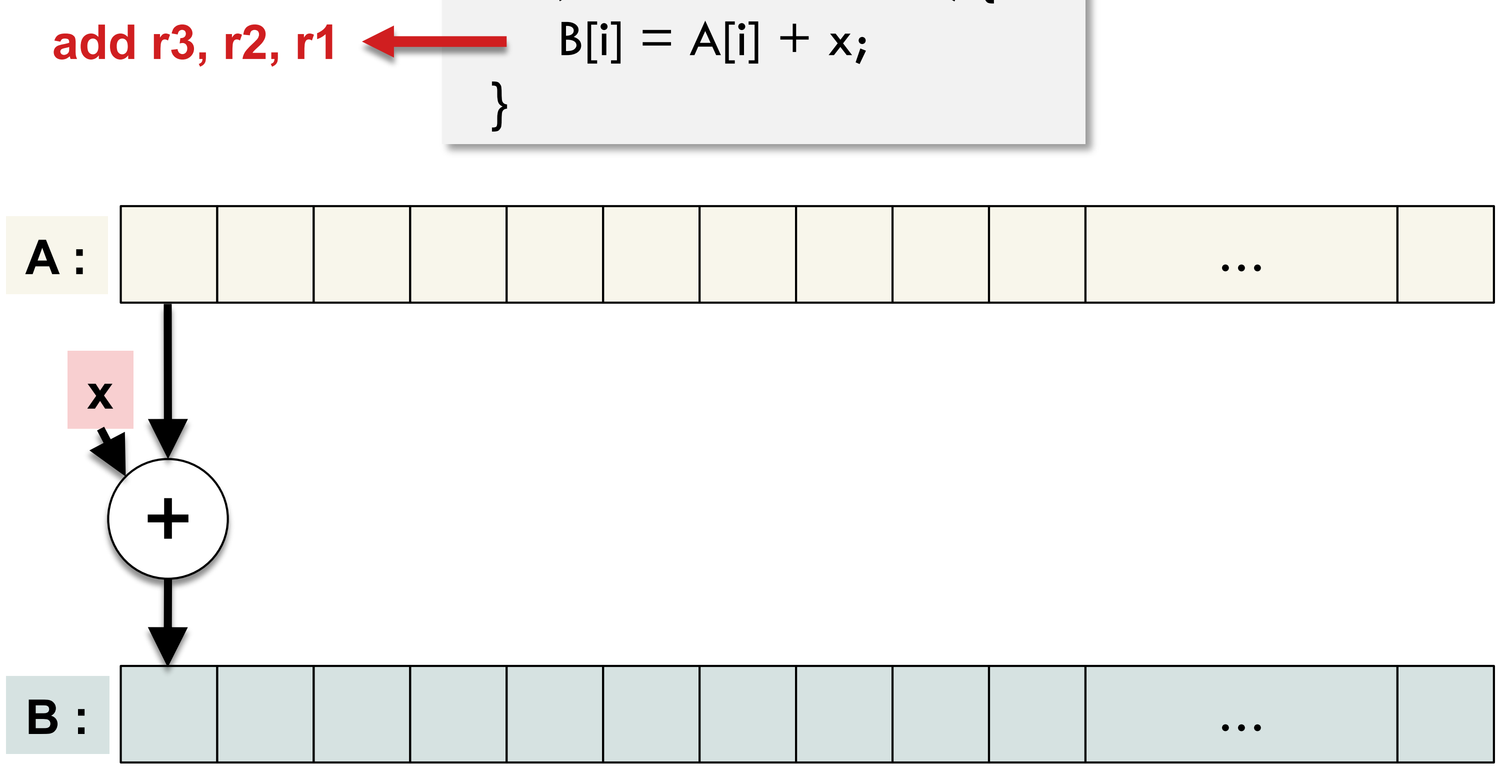
```
for(i=0; i<1000; ++i) {
    B[i] = A[i] + x;
}
```



Vector Processing

- Scalar vs. vector processor

```
for(i=0; i<1000; ++i) {
    B[i] = A[i] + x;
}
```

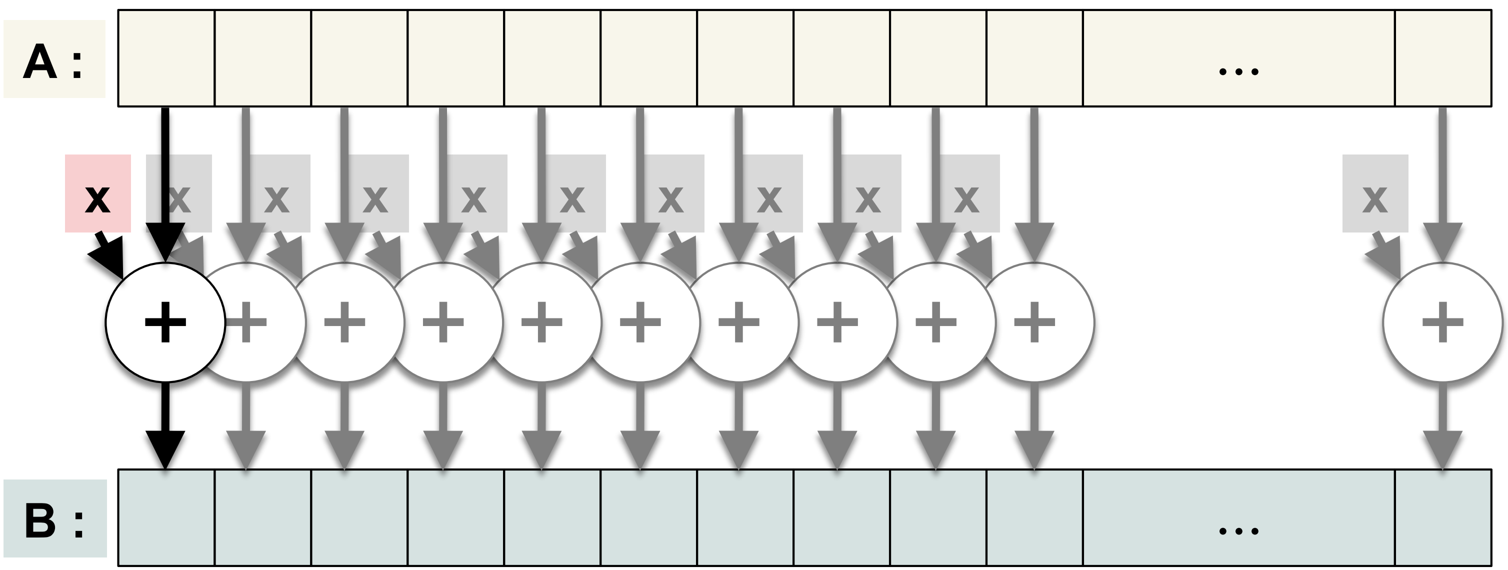


Vector Processing

- Scalar vs. vector processor

```
for(i=0; i<1000; ++i) {
    B[i] = A[i] + x;
}
```

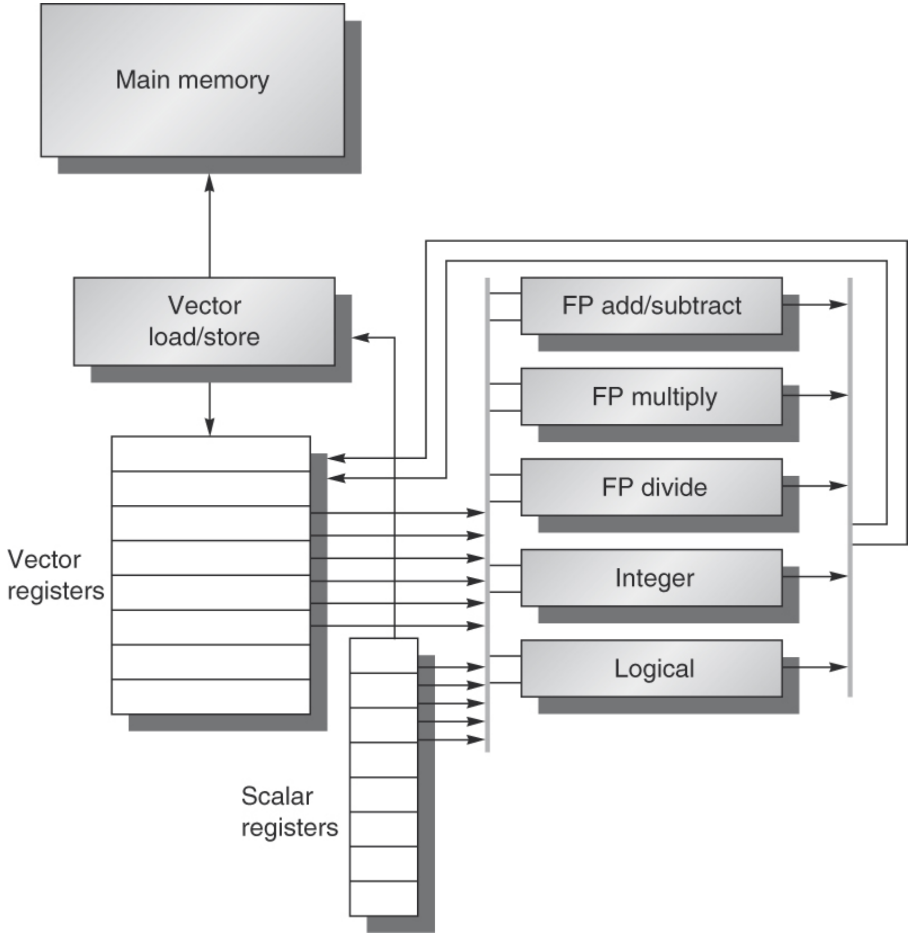
→ **vadd v3, v2, v1**



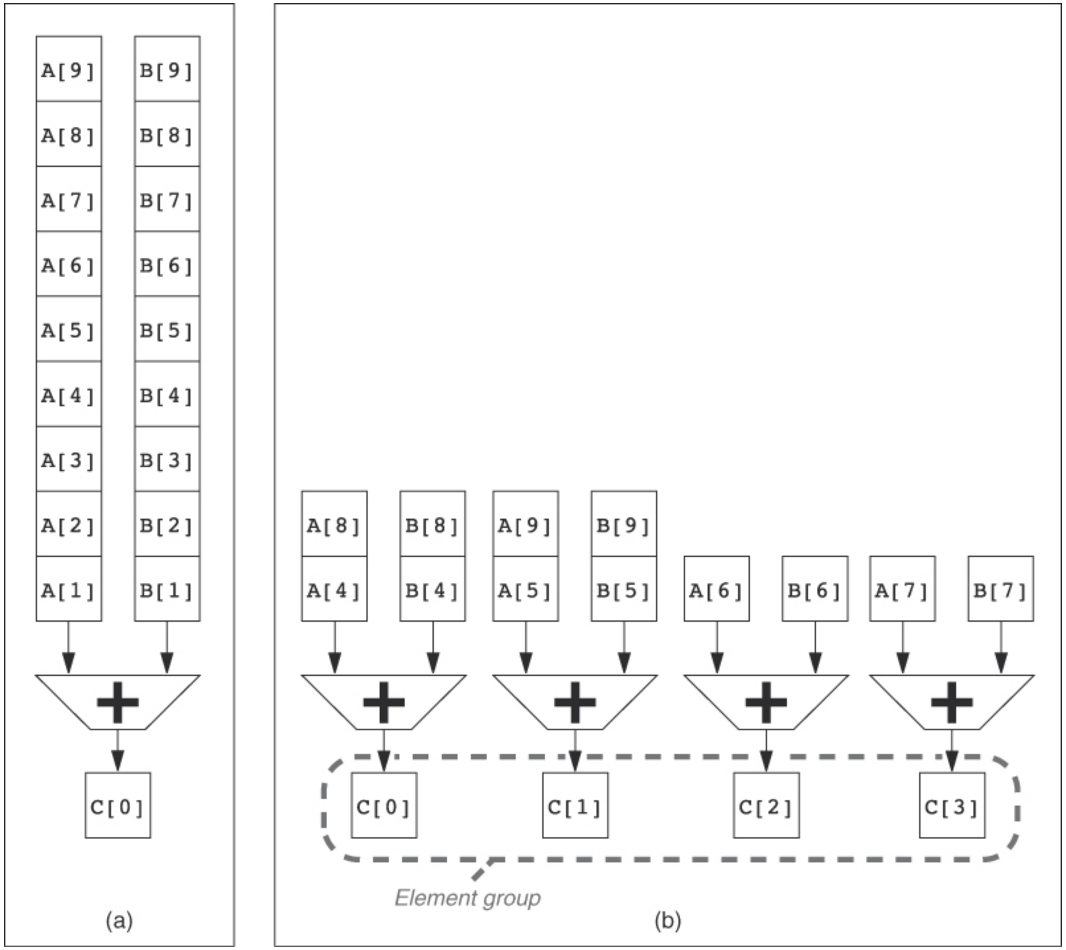
Vector Processor

- A scalar processor—e.g., MIPS
  - Scalar register file
  - Scalar functional units
- Vector register file
  - 2D register array
  - Each register is an array of registers
  - The number of elements per register determines the max vector length
- Vector functional units
  - Single opcode activates multiple units
  - Integer, floating point, load and stores

Basic Vector Processor Architecture



Parallel vs. Pipeline Units

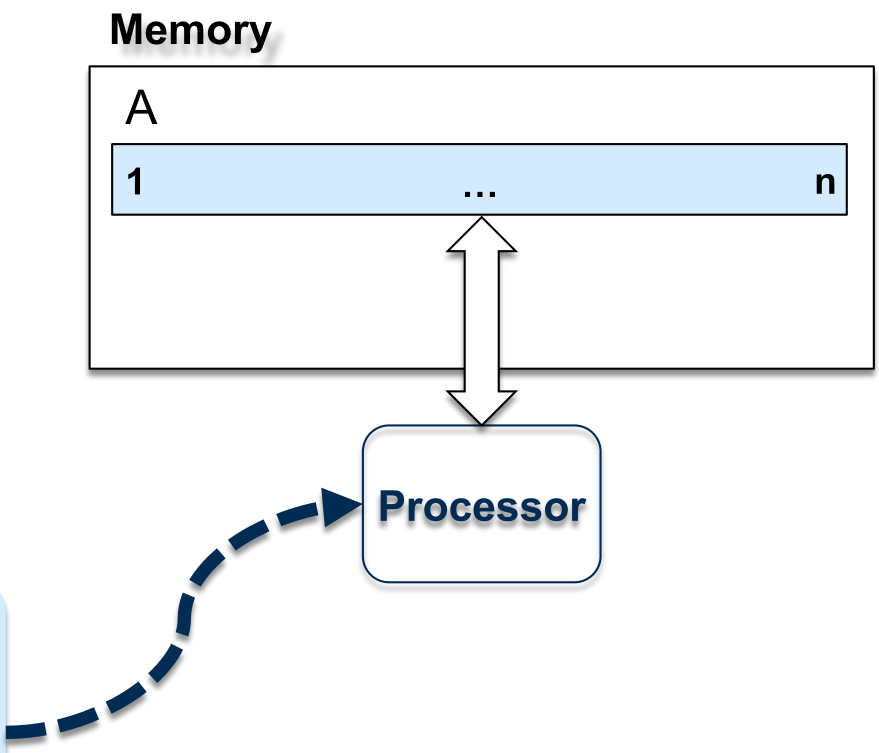


Example Code I

- A sequential application runs as a single thread

Kernel Function:

```
void kern (int start, int end) {
   int i;
   for(i=start; i<=end; ++i) {
      A[i] = A[i] * A[i] + 5;
   }
}
```



Single Thread

```
main() {
   ...
   kern (1, n);
   ...
}
```

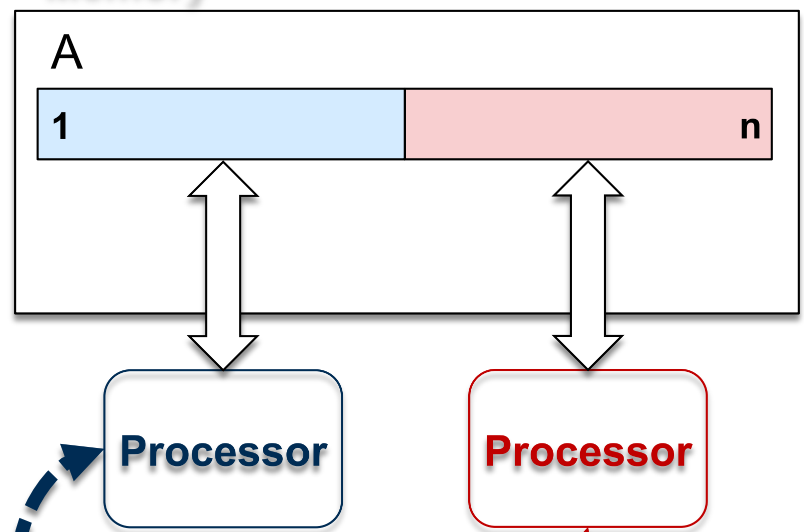
Example Code I

- Two threads operating on separate partitions

Kernel Function:

```
void kern (int start, int end) {
   int i;
   for(i=start; i<=end; ++i) {
      A[i] = A[i] * A[i] + 5;
   }
}
```

Memory



Thread 0

```
main() {
   ...
   kern (1, n/2);
   ...
}
```

Thread 1

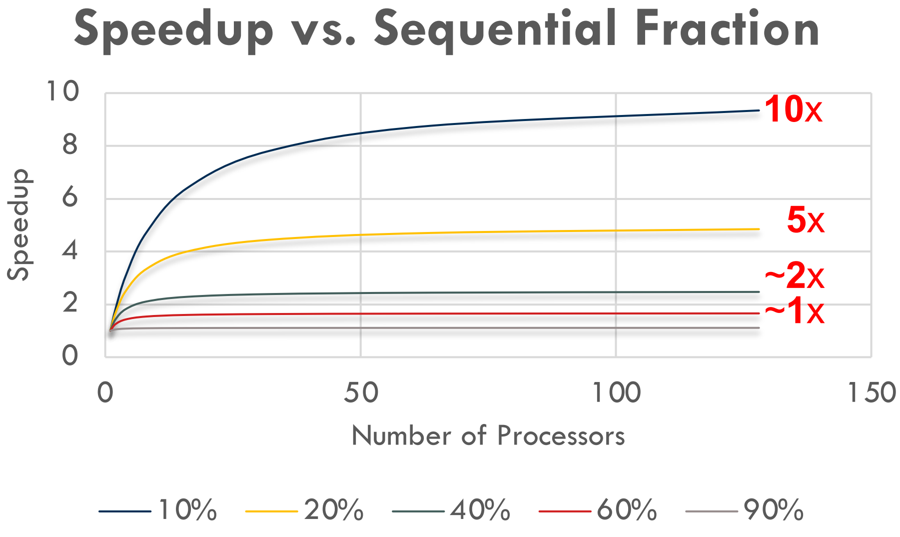
```
kern (n/2+1, n);
```

Performance of Parallel Processing

- Recall: Amdahl's law for theoretical speedup
  - Overall speedup is limited to the fraction of the program that can be executed in parallel

$$speedup = \frac{1}{f + \frac{1-f}{n}}$$

f: sequential fraction

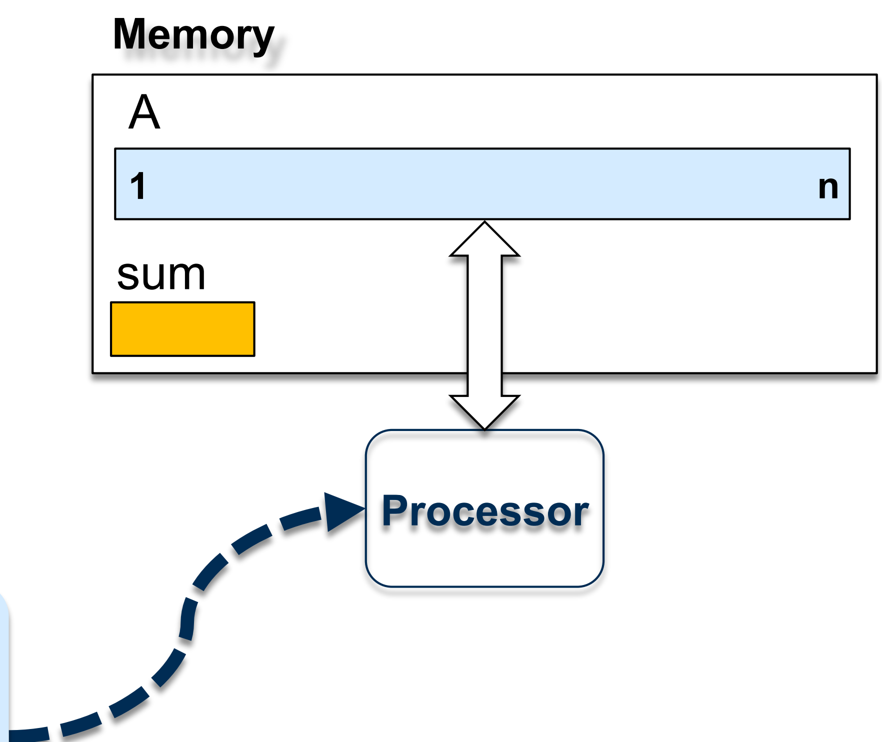


Example Code II

- A single location is updated every time

Kernel Function:

```
void kern (int start, int end) {
   int i;
   for(i=start; i<=end; ++i) {
      sum = sum * A[i];
   }
}
```



Thread 0

```
main() {
   ...
   kern (1, n);
   ...
}
```

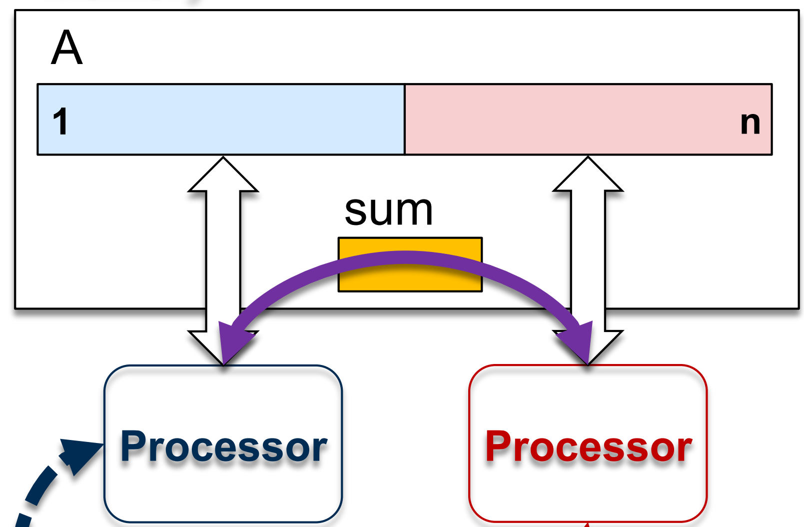
Example Code II

- Two threads operating on separate partitions

Kernel Function:

```
void kern (int start, int end) {
   int i;
   for(i=start; i<=end; ++i) {
      sum = sum * A[i];
   }
}
```

Memory



Thread 0

```
main() {
   …
   kern (1, n/2);
   …
}
```

Thread 1

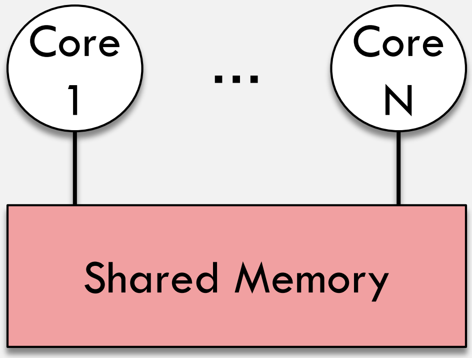
```
kern (n/2+1, n);
```

Communication in Multiprocessors

- How multiple processor cores communicate?

Shared Memory

- Multiple threads employ shared memory
- Easy for programmers (loads and stores)



Message Passing

- Explicit communication through interconnection network
- Simple hardware