

# DATA LEVEL PARALLELISM

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

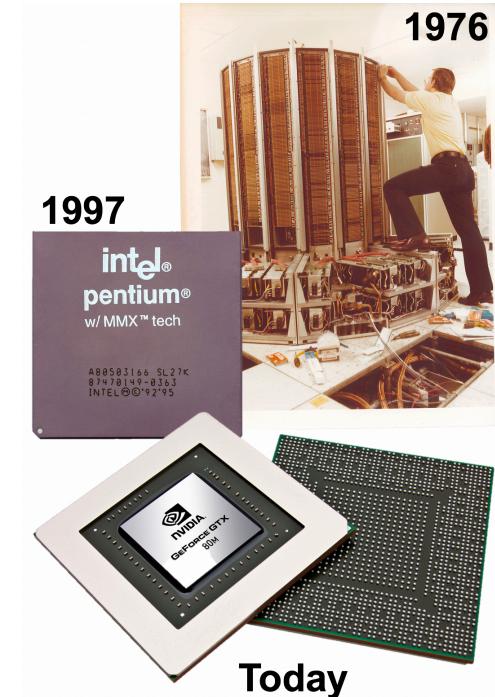
University of Utah

# Overview

- ILP: instruction level parallelism
  - ▣ Out of order execution (all in hardware)
  - ▣ IPC hardly achieves more than 2
- Other forms of parallelism
  - ▣ DLP: data level parallelism
    - Vector processors, SIMD, and GPUs
  - ▣ TLP: thread level parallelism
    - Multiprocessors, and hardware multithreading
  - ▣ RLP: request level parallelism
    - Datacenters

# Data Level Parallelism

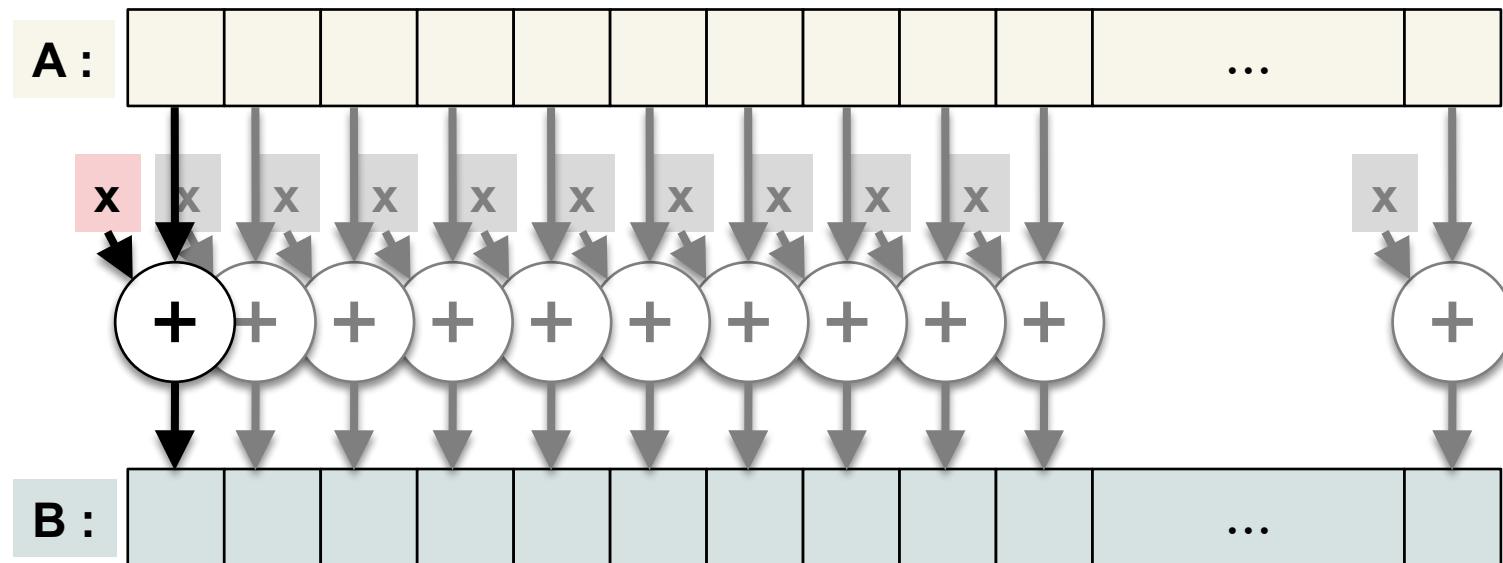
- Due to executing the same code on a large number of objects
  - ▣ Common in scientific computing
- DLP architectures
  - ▣ Vector processors—e.g., Cray machines
  - ▣ SIMD extensions—e.g., Intel MMX
  - ▣ Graphics processing unit—e.g., NVIDIA
- Improve throughput rather than latency
  - ▣ Not good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



# Vector Processing

## □ Scalar vs. vector processor

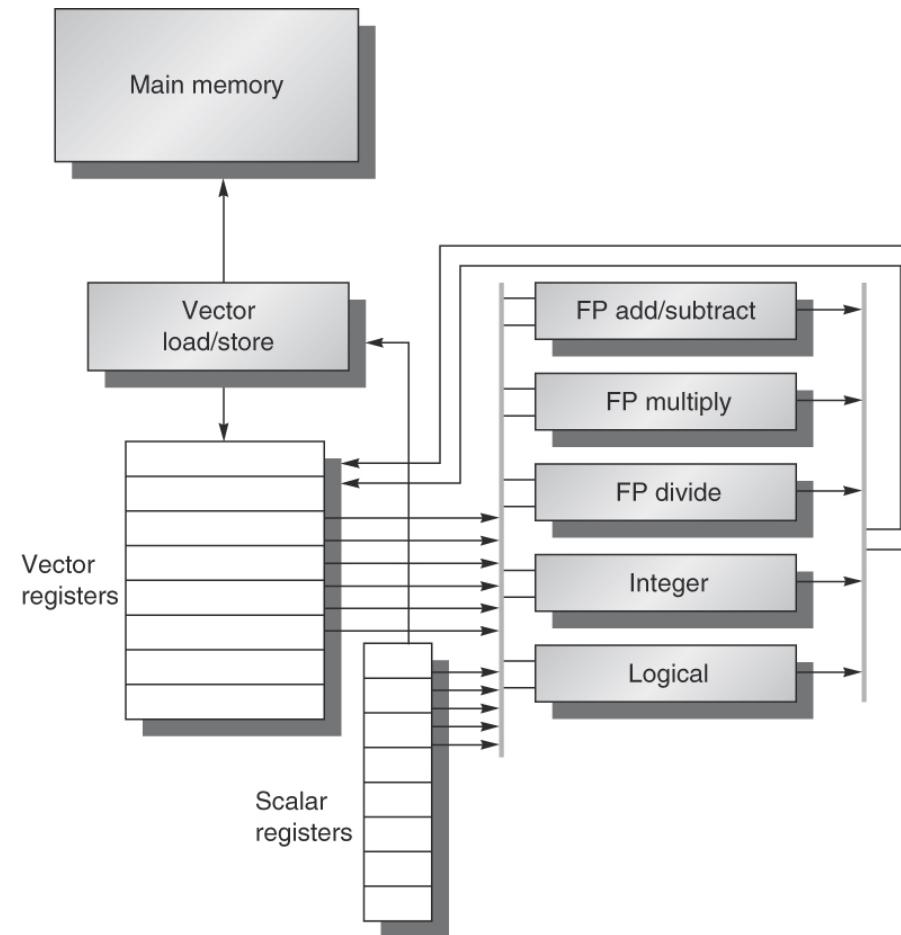
```
for(i=0; i<1000; ++i) {  
    add r3, r2, r1 ← B[i] = A[i] + x;  
}  
} → vadd v3, v2, v1
```



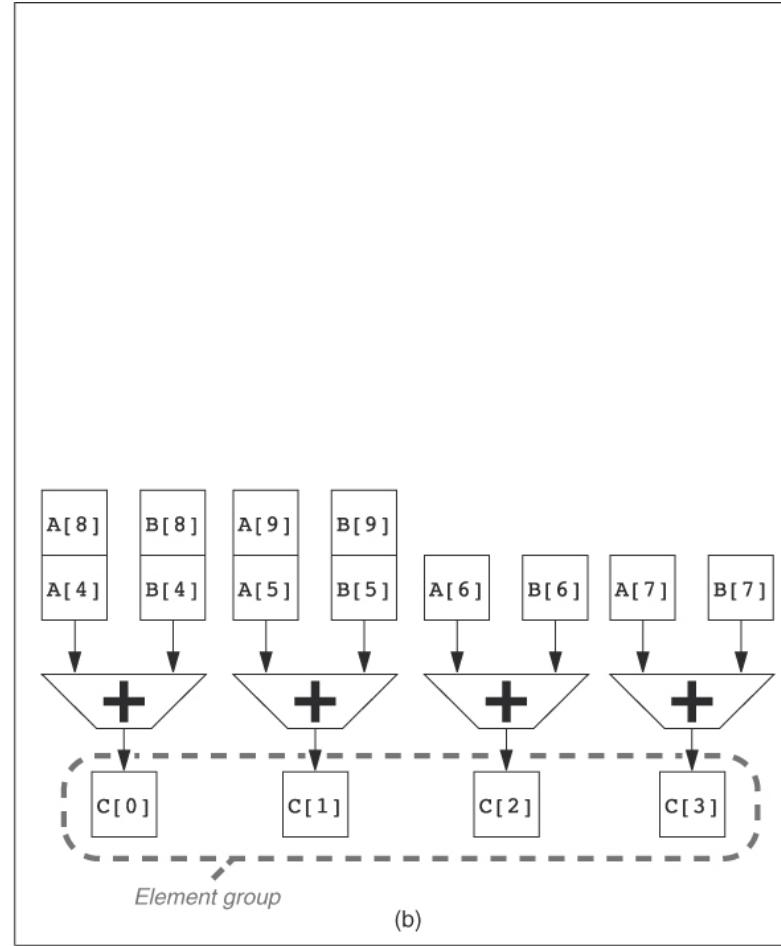
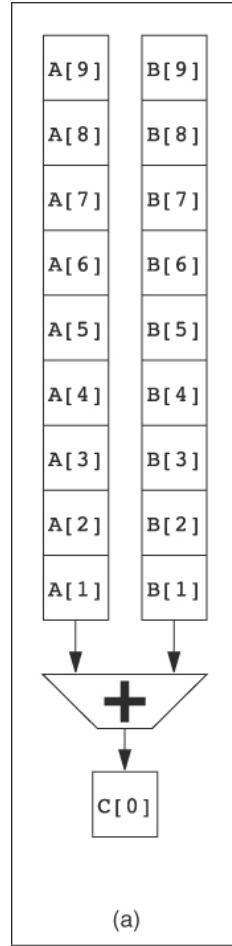
# Vector Processor

- 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



# Vector Instruction Set Architecture

---

- Single instruction defines multiple operations
  - ▣ Lower instruction fetch/decode/issue cost
- Operations are executed in parallel
  - ▣ Naturally no dependency among data elements
  - ▣ Simple hardware
- Predictable memory access pattern
  - ▣ Improve performance via prefetching
  - ▣ Simple memory scheduling policy
  - ▣ Multi banking may be used for improving bandwidth

# Vector Operation Length

---

- Fixed in hardware
  - ▣ Common in narrow SIMD
  - ▣ Not efficient for wide SIMD
- Variable length
  - ▣ Determined by a vector length register (VLR)
  - ▣ MVL is the maximum VL
  - ▣ How to process vectors wider than MVL?

# Conditional Execution

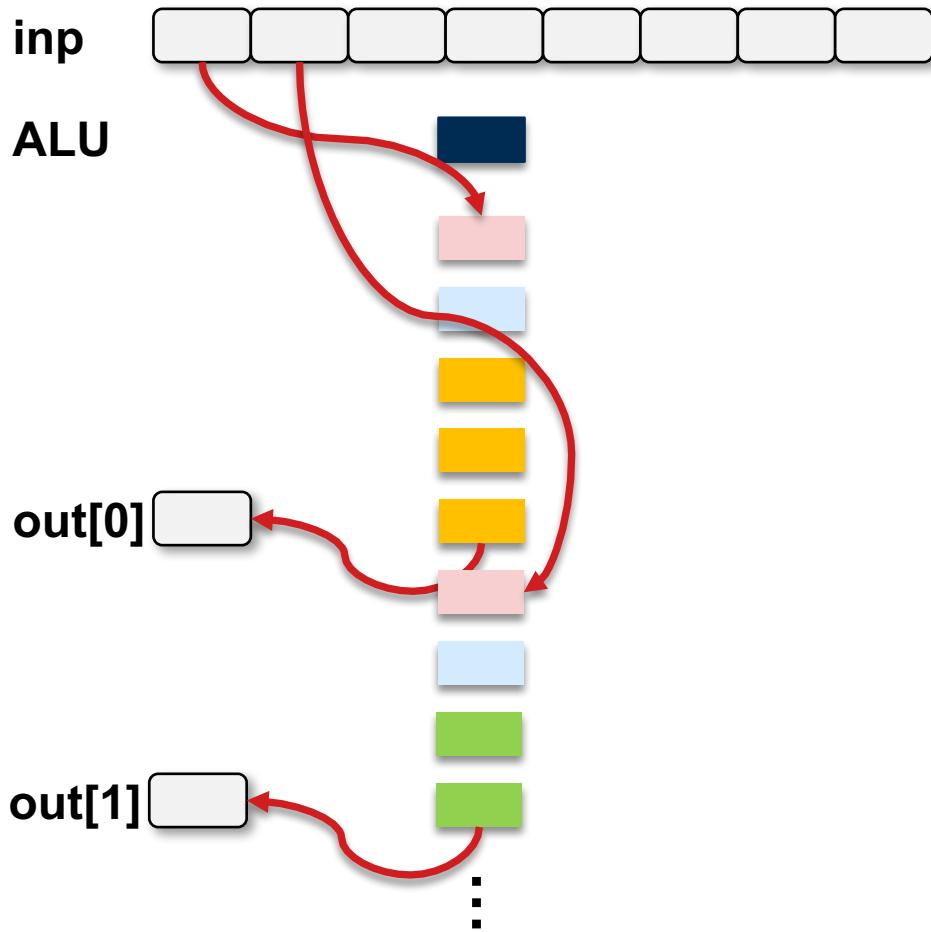
- Question: how to handle branches?
- Solution: by predication
  - ▣ Use masks, flag vectors with single-bit elements
  - ▣ Determine the flag values based on vector compare
  - ▣ Use flag registers as control mask for the next vector operations

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



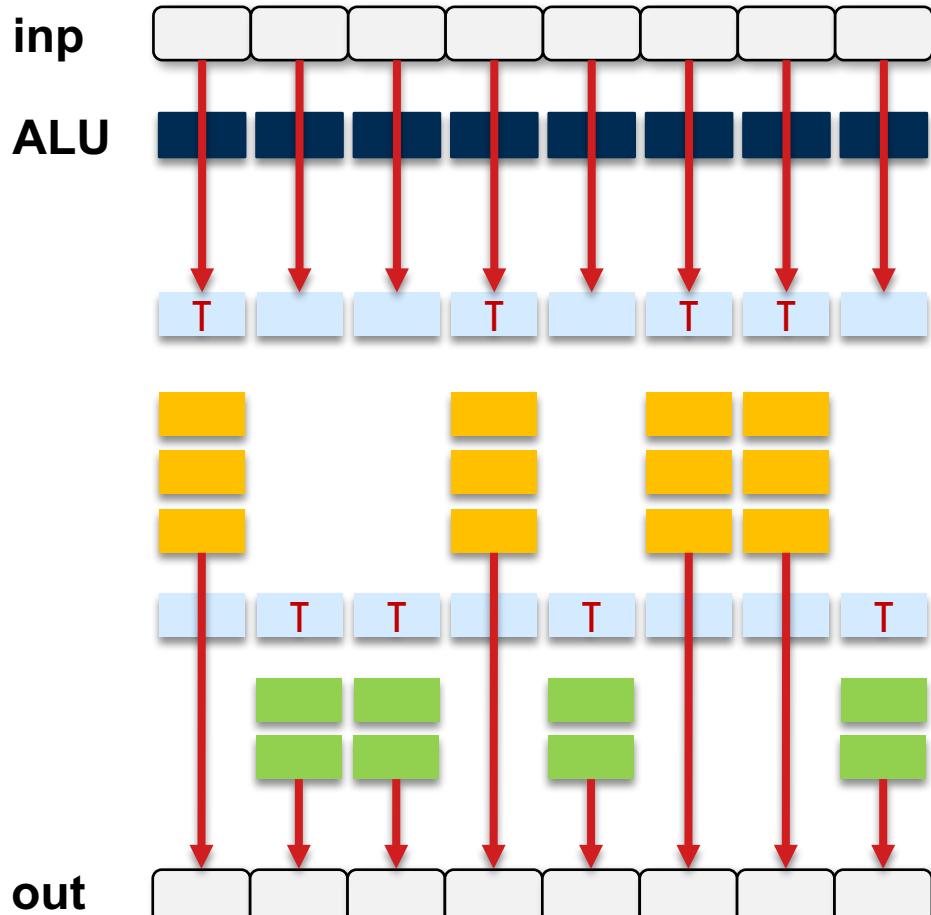
```
vld V1, Ra  
vld V2, Rb  
vcmp.neq.vv MO, V1, V2  
vsub.vv V3, V2, V1, MO  
vst V3, Ra
```

# Branches in Scalar Processors



```
for (i = 0; i < 8; ++i) {  
    if (inp[i] > 0) {  
        y = inp[i] * inp[i];  
        y = y + 2 * inp[i];  
        out[i] = y + 3;  
    } else {  
        y = 4 * inp[i];  
        out[i] = y + 1;  
    }  
}
```

# Branches in Vector Processors



```
if (inp[i] > 0) {  
    y = inp[i] * inp[i];  
    y = y + 2 * inp[i];  
    out[i] = y + 3;  
} else {  
    y = 4 * inp[i];  
    out[i] = y + 1;  
}
```