

CSE 160

Lecture 10

Instruction level parallelism (ILP)
Vectorization

Announcements

- Quiz on Friday
- Signup for Friday labs sessions in APM

Particle simulation

- Two routines contain the work to be parallelized according to **openmp #pragmas**
- Anything not inside **OMP parallel for** should run serially
- Synchronization?

```
void SimulateParticles(){  
    for( int step = 0; step < nsteps; step++ ) {  
        apply_forces(particles,n);  
  
        if (imbal)  
            imbal_particles(particles,n);  
  
        move_particles(particles,n);  
  
        VelNorms(particles,n,uMax,vMax,uL2,vL2);  
    }  
}
```

Code for computing the force

- OpenMP parallelization of the outermost loop

```
void apply_forces( particle_t* particles, int n){  
#pragma omp parallel for shared(particles,n)  
for( int i = 0; i < n; i++ ) {  
    particles[i].ax = particles[i].ay = 0;  
    if ((particles[i].vx  $\neq$  0) || (particles[i].vy  $\neq$  0)) // for imbalanced  
        for (int j = 0; j < n; j++ ) { // case  
            if (i==j)  
                continue;  
            particles[i].{ax,ay} += coef * {dx,dy};  
        }  
    }  
}
```

Today's lecture

- Pipelining
- Instruction level parallelism (ILP)
- Vectorization, SSE, vector instructions

What makes a processor run faster?

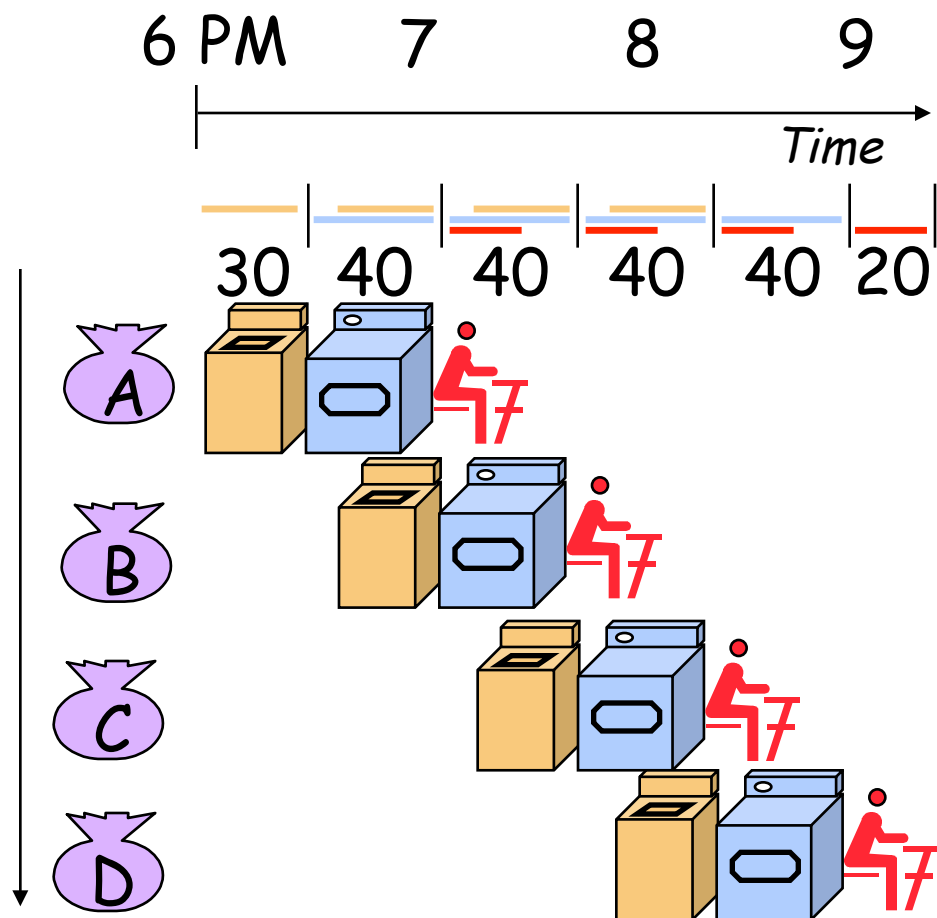
- Registers and cache
- Pipelining
- Instruction level parallelism
- Vectorization, SSE

Pipelining

- Assembly line processing - an auto plant

Dave Patterson's Laundry example: 4 people doing laundry

wash (30 min) + dry (40 min) + fold (20 min) = 90 min



- Sequential execution takes $4 * 90\text{min} = 6 \text{ hours}$
- Pipelined execution takes $30 + 4 * 40 + 20 = 3.5 \text{ hours}$
- Bandwidth = loads/hour
- Pipelining helps bandwidth but not latency (90 min)
- Bandwidth limited by slowest pipeline stage
- Potential speedup = Number pipe stages

Instruction level parallelism

- Execute more than one instruction at a time

$x = y / z;$

$a = b + c;$

- Dynamic techniques

- Allow stalled instructions to proceed

- Reorder instructions

$x = y / z;$

$a = x + c;$

$t = c - q;$



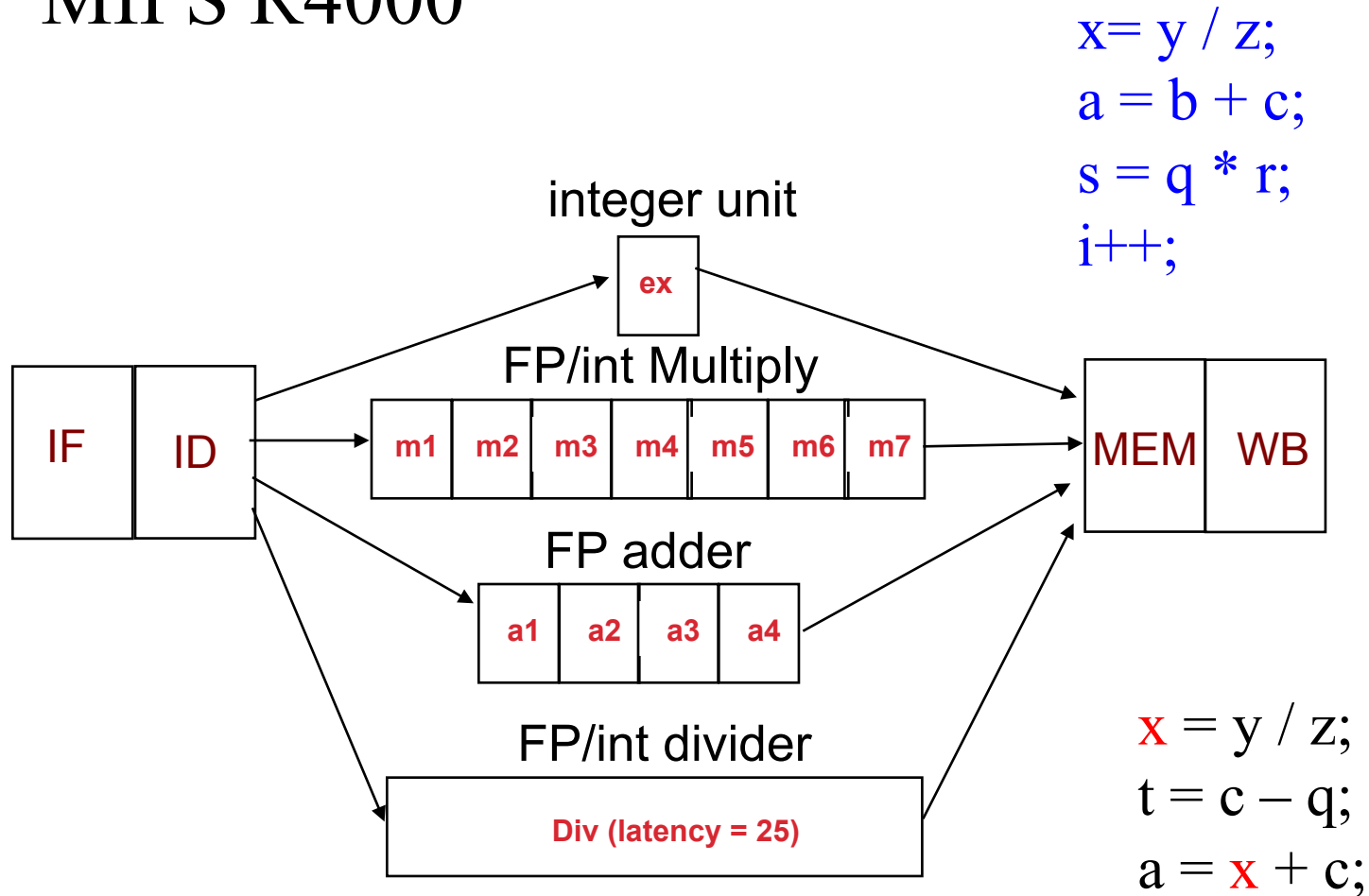
$x = y / z;$

$t = c - q;$

$a = x + c;$

Multi-execution pipeline

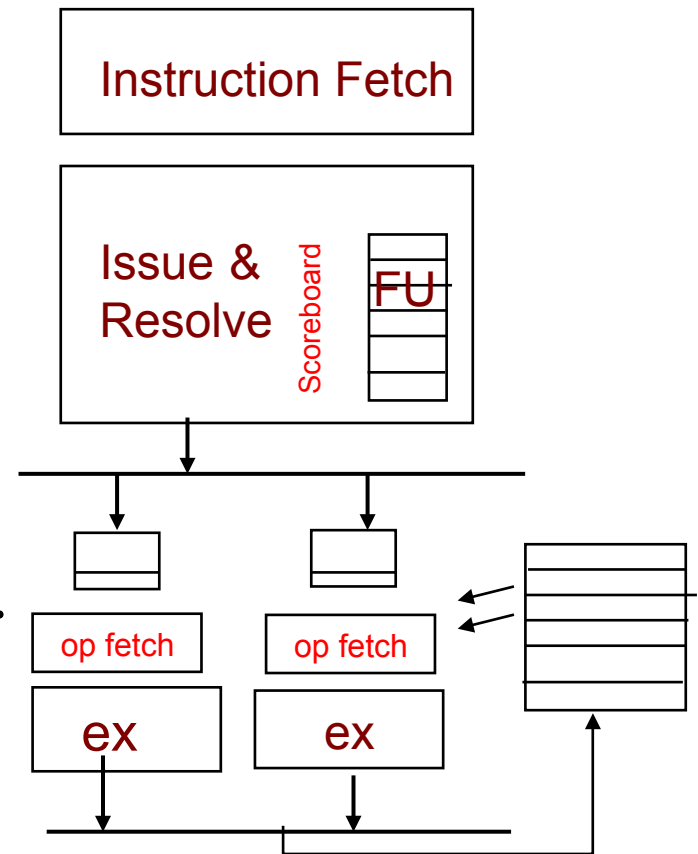
- MIPS R4000



David Culler

Scoreboarding, Tomasulo's algorithm

- Hardware data structures keep track of
 - When instructions complete
 - Which instructions depend on the results
 - When it's safe to write a reg.
- Deals with data hazards
 - **WAR** (Write after read)
 - **RAW** (Read after write)
 - **WAW** (Write after write)



$a = b * c$
 $x = y - b$
 $q = a / y$
 $y = x + b$

What makes a processor run faster?

- Registers and cache
- Pipelining
- Instruction level parallelism
- **Vectorization, SSE**

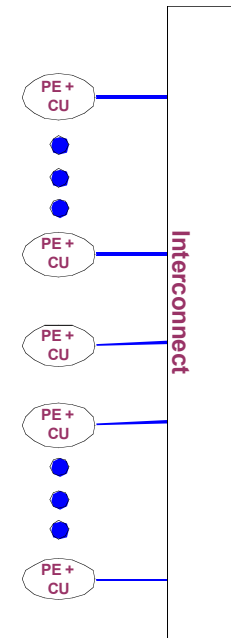
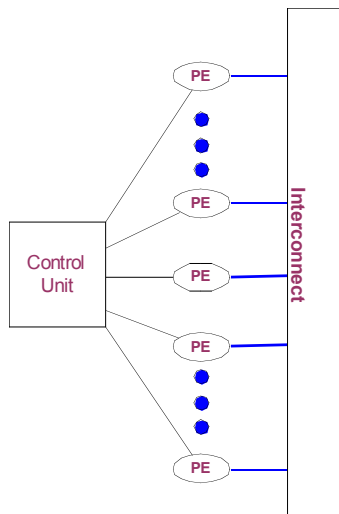
Control Mechanism

Flynn's classification (1966)

How do the processors issue instructions?

SIMD: Single Instruction, Multiple Data

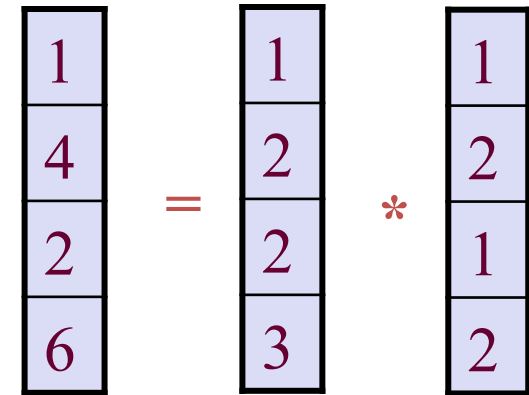
Execute a global instruction stream in lock-step



MIMD: Multiple Instruction, Multiple Data
Clusters and servers processors execute
instruction streams independently

SIMD (Single Instruction Multiple Data)

- Operate on regular arrays of data
- Two landmark SIMD designs
 - ILIAC IV (1960s)
 - Connection Machine 1 and 2 (1980s)
- Vector computer: Cray-1 (1976)
- Intel and others support SIMD for multimedia and graphics
 - SSE
 - Streaming SIMD extensions, AltiVec
 - Operations defined on vectors
- GPUs, Cell Broadband Engine (Sony Playstation)
- Reduced performance on data dependent or irregular computations



```
forall i = 0:N-1  
    p[i] = a[i] * b[i]
```

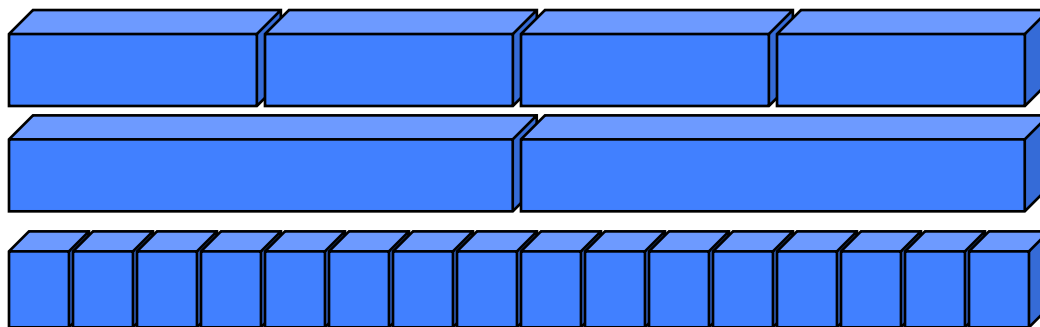
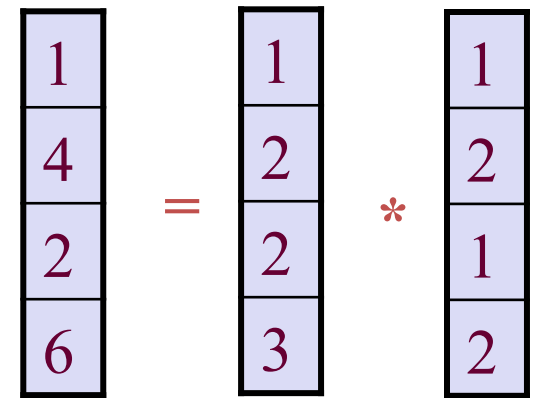
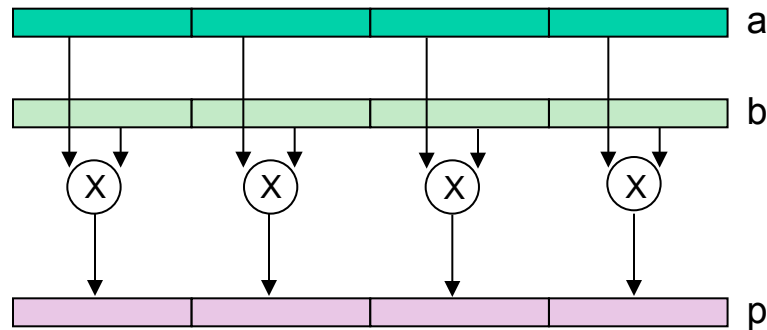
```
forall i = 0 : n-1  
    x[i] = y[i] + z [ K[i] ]  
end forall
```

```
forall i = 0 : n-1  
    if ( x[i] < 0) then  
        y[i] = x[i]  
    else  
        y[i] = √x[i]  
    end if  
end forall
```

Streaming SIMD Extensions

- SSE (SSE4 on Intel Nehalem), AltiVec
- Short vectors: 128 bits (AVX: 256 bits)
- en.wikipedia.org/wiki/Streaming_SIMD_Extensions

for $i = 0:N-1$ { $p[i] = a[i] * b[i];$ }



4 floats

2 doubles

16 bytes

How do we use the SSE instructions?

- Low level: assembly language or libraries
- Higher level: a vectorizing compiler
- Bang (sse3): g++ -O3 automatically vectorizes

```
g++ --std=c++11 -O3 -ftree-vectorizer-verbose=2  
float b[N], c[N];  
for (int i=0; i<N; i++)  
    b[i] += b[i]*b[i] + c[i]*c[i];
```

7: LOOP VECTORIZED.

vec.cpp:6: note: vectorized 1 loops in function..

- Performance

Single precision: With vectorization : 1.9 sec.

Without vectorization : 3.2 sec.

Double precision: With vectorization: 3.6 sec.

Without vectorization : 3.3 sec.

<http://gcc.gnu.org/projects/tree-ssa/vectorization.html>

How does the vectorizer work?

- Transformed code

```
for (i = 0; i < 1024; i+=4)
    a[i:i+3] = b[i:i+3] + c[i:i+3];
```

- Vector instructions

```
for (i = 0; i < 1024; i+=4){
    vB = vec_ld( &b[i] );
    vC = vec_ld( &c[i] );
    vA = vec_add( vB, vC );
    vec_st( vA, &a[i] );
}
```

What prevents vectorization

- Data dependencies

```
for (int i = 1; i < N; i++)
```

```
    b[i] = b[i-1] + 2;
```

```
b[1] = b[0] + 2;
```

```
b[2] = b[1] + 2;
```

```
b[3] = b[2] + 2;
```

not vectorized, possible dependence between data-refs `b[D.30220_12]` and `b[i_42]`

- Inner loops only

```
for(int j=0; j< reps; j++)
```

```
    for (int i=0; i<N; i++)
```

```
        a[i] = b[i] + c[i];
```

What prevents vectorization

- Interrupted flow out of the loop

```
for (i=0; i<n; i++) {  
    a[i] = b[i] + c[i];  
    maxval = (a[i] > maxval ? a[i] : maxval);  
    if (maxval > 1000.0) break;  
}
```

Loop not vectorized/parallelized: multiple exits

- This loop will vectorize

```
for (i=0; i<n; i++) {  
    a[i] = b[i] + c[i];  
    maxval = (a[i] > maxval ? a[i] : maxval);  
}
```

In class exercises

Questions

1. ~~Iteration to thread mapping~~
2. Removing data dependencies
3. Dependence analysis
4. Time constrained scaling
5. Tree Summation
6. Performance

2. Removing data dependencies

- B initially: 0 1 2 3 4 5 6 7
- B on 1 thread: 7 7 7 7 11 12 13 14
- How can we split into 2 loops so that each loop parallelizes, the result is correct?

#pragma omp for shared (N,B)

for i = 0 to N-1

B[i] += B[N-1-i];

B[0] += B[7], B[1] += B[6], B[2] += B[5]

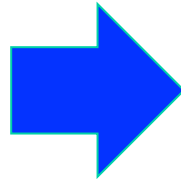
B[3] += B[4], B[4] += B[2], B[5] += B[2]

B[7] += B[0]

Splitting a loop

- For iterations $i=N/2+1$ to N , $B[N-i]$ reference newly computed data
- All others reference “old” data
- B initially: 0 1 2 3 4 5 6 7
- Correct result: 7 7 7 7 11 12 13 14

for $i = 0$ to $N-1$
 $B[i] += B[N-i];$



for $i = 0$ to $N/2-1$
 $B[i] += B[N-1-i];$
for $i = N/2+1$ to $N-1$
 $B[i] += B[N-1-i];$

In \$PUB/Examples/OpenMP/Assign
Compile with `omp=1` on “make” line

3. Loop Dependence Analysis

- Which loop(s) can we correctly parallelize with OpenMP?

1. for $i = 1$ to $N-1$
 $A[i] = A[i] + B[i-1];$

2. for $i = 0$ to $N-2$
 $A[i+1] = A[i] + 1;$

3. for $i = 0$ to $N-1$ step 2
 $A[i] = A[i-1] + A[i];$

4. for $i = 0$ to $N-2$ {
 $A[i] = B[i];$
 $C[i] = A[i] + B[i];$
 $E[i] = C[i+1];$
}

4. Time constrained scaling

- Sum N numbers on P processors
- Let $N \gg P$
- Determine the largest problem that can be solved in time $T=10^4$ time units on 512 processors
- Let time to perform one addition = 1 time unit
- Let β = time to add a value inside a critical section

Performance model

- Local additions: $N/P - 1$
- Reduction: $\beta (\lg P)$
- Since $N \gg P$
$$T(N,P) \sim (N/P) + \beta (\lg P)$$
- Determine the largest problem that can be solved in time $T = 10^4$ time units on $P = 512$ processors, $\beta = 1000$ time units
- Constraint: $T(512,N) \leq 10^4$
$$\Rightarrow (N/512) + 1000 (\lg 512)$$
$$= (N/512) + 1000 * (9) \leq 10^4$$
$$\Rightarrow N \leq 5 \times 10^5 \text{ (approximately)}$$

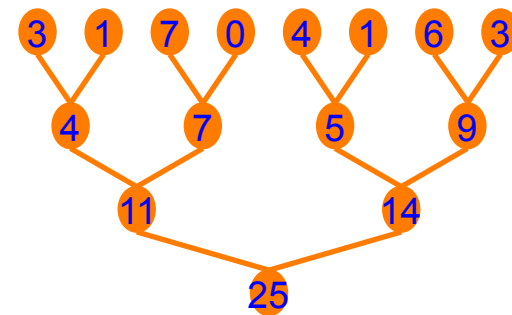
5. Tree Summation

- Input: an array $x[]$, length $N \gg P$
- Output: Sum of the elements of $x[]$
- Goal: Compute the sum in $\lg P$ time

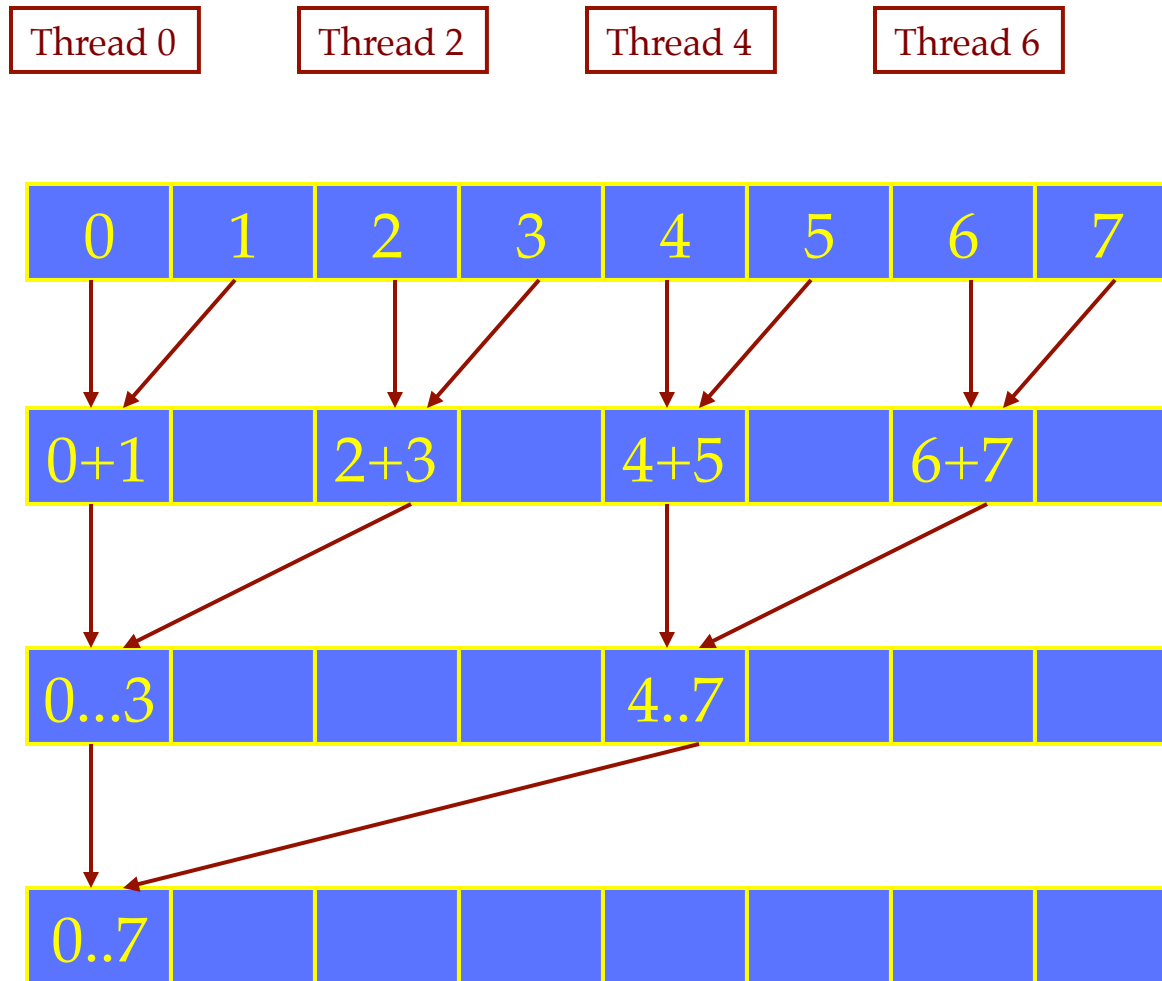
```
sum = 0;  
for i=0 to N-1  
    sum += x[i]
```

- Assume P is a power of 2, $K = \lg P$
- Starter code
for $m = 0$ to $K-1$ {

}
}



Visualizing the Summation



6. Performance

- You observe the following running times for a parallel program running a fixed workload N
- Assume that the only losses are due to serial sections
- What is the speedup and efficiency on 8 processors?
- What will the running time be on 4 processors?
- What is the maximum possible speedup on an infinite number of processors?
- What fraction of the total running time on 1 processor corresponds to the serial section?
- What fraction of the total running time on 2 processors corresponds to the serial section?

NT	Time
1	10000
2	6000
8	3000