

Fundamentals of Parallelism on Intel® Architecture

Week 3
Multithreading
with OpenMP



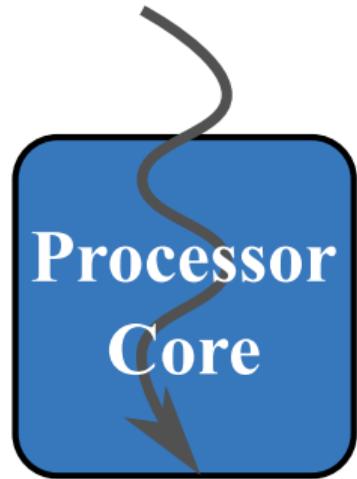
May 2017

§1. Cores, Processes and Threads



Serial Processor, Serial Code

Instruction
Stream



Computing Platforms

Intel Xeon
Processor



Current: Broadwell
Upcoming: Skylake

Intel Xeon Phi
Coprocessor, 1st generation



Knights Corner (KNC)

Intel Xeon Phi
Processor, 2nd generation*



* socket and coprocessor versions

Knights Landing (KNL)

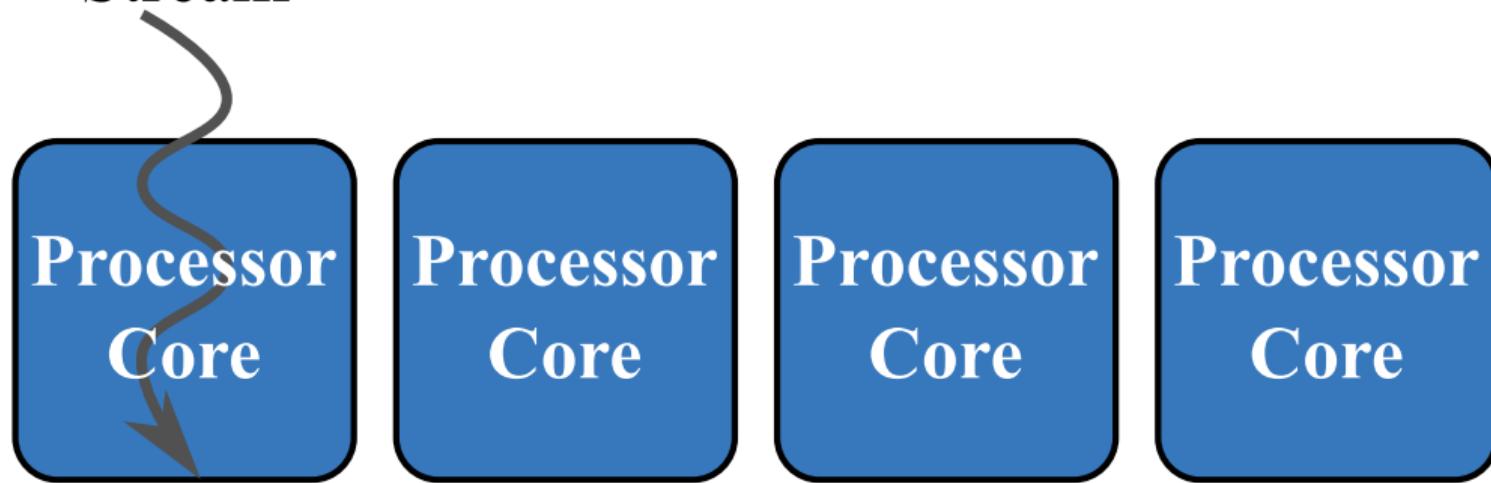
Multi-Core Architecture

Intel Many Integrated Core (MIC) Architecture

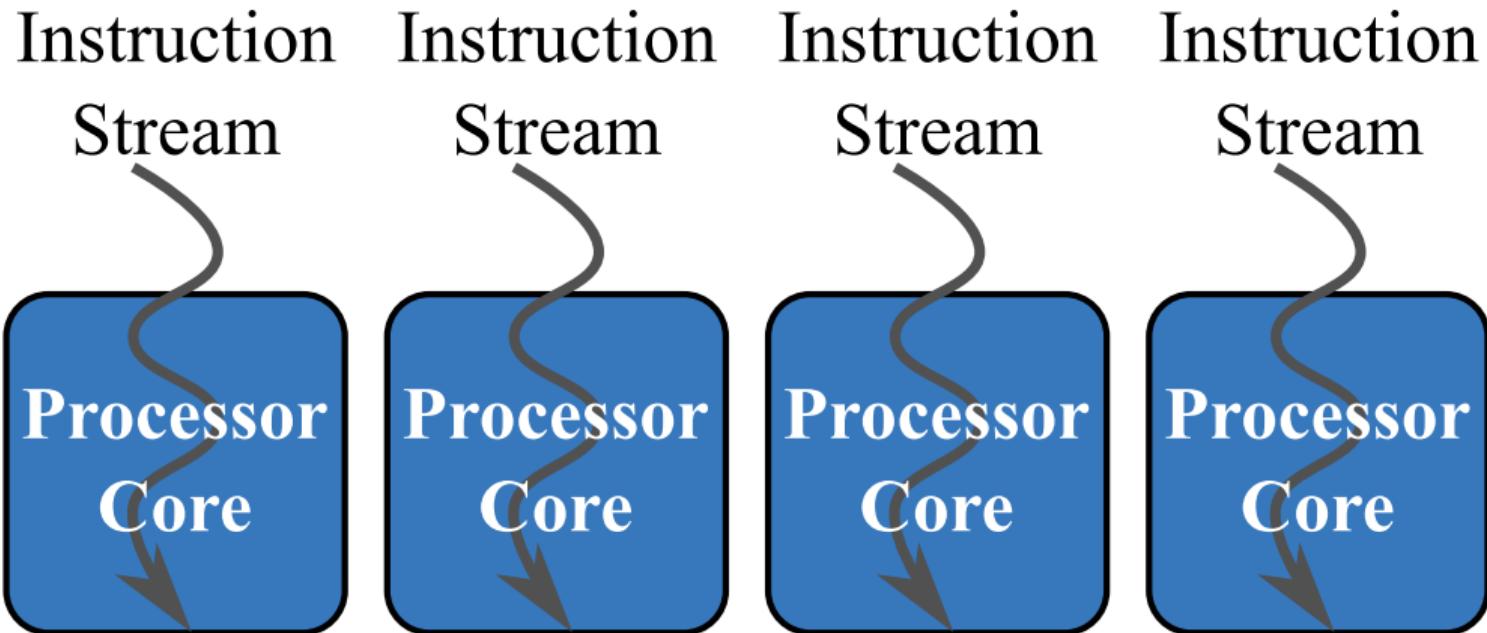


Parallel Processor, Serial Code

Instruction
Stream

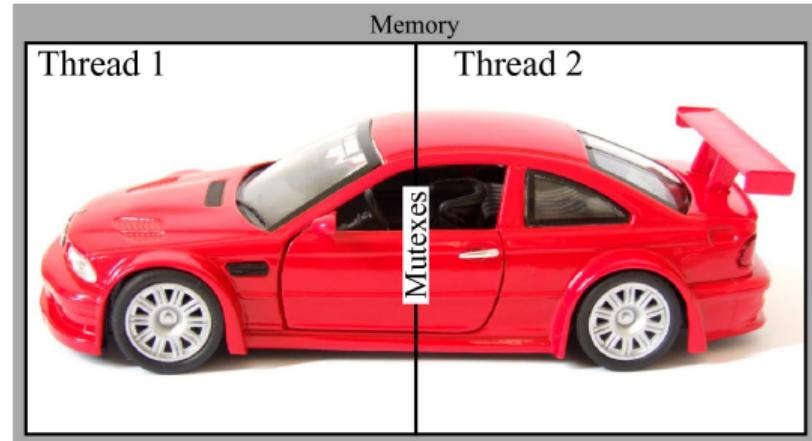
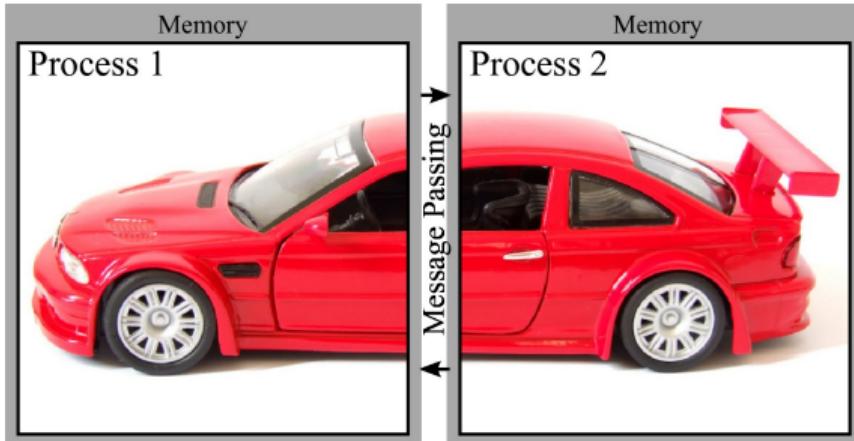


Parallel Processor, Parallel Code



Threads versus Processes

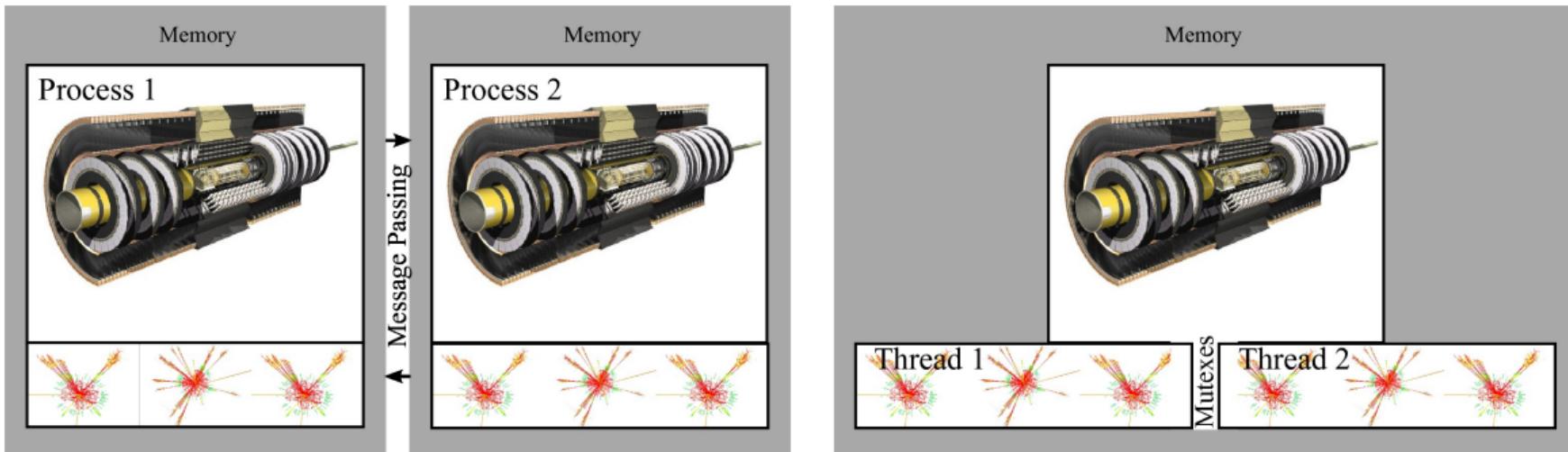
Option 1: Partitioning data set between threads/processes



Examples: computational fluid dynamics (CFD), image processing.

Threads versus Processes

Option 2: Sharing data set between threads/processes



Examples: particle transport simulation, machine learning (inference).

§2. Creating Threads



Threading Frameworks

| Framework | Functionality |
|---------------|--|
| C++11 Threads | Asynchronous functions; only C++ |
| POSIX Threads | Fork/join; C/C++/Fortran; Linux |
| Cilk Plus | Async tasks, loops, reducers, load balance; C/C++ |
| TBB | Trees of tasks, complex patterns; only C++ |
| OpenMP | Tasks, loops, reduction, load balancing, affinity, nesting, C/C++/Fortran (+SIMD, offload) |



"Hello World" OpenMP Program

```
1 #include <omp.h>
2 #include <stdio.h>
3
4 int main(){
5     // This code is executed by 1 thread
6     const int nt=omp_get_max_threads();
7     printf("OpenMP with %d threads\n", nt);
8
9 #pragma omp parallel
10    { // This code is executed in parallel
11        // by multiple threads
12        printf("Hello World from thread %d\n",
13              omp_get_thread_num());
14    }
15 }
```

- ▷ OpenMP = “Open Multi-Processing” = computing-oriented framework for shared-memory programming
- ▷ Threads – streams of instructions that share memory address space
- ▷ Distribute threads across CPU cores for parallel speedup



Compiling the "Hello World" OpenMP Program

```
vega@lyra% icpc -fopenmp hello_omp.cc
vega@lyra% export OMP_NUM_THREADS=5
vega@lyra% ./a.out
OpenMP with 5 threads
Hello World from thread 0
Hello World from thread 3
Hello World from thread 1
Hello World from thread 2
Hello World from thread 4
```

`OMP_NUM_THREADS` controls number of OpenMP threads (default: logical CPU count)



§3. Variable Sharing



Control of Variable Sharing

Method 1: using clauses in pragma omp parallel (C, C++, Fortran):

```
1 int A, B; // Variables declared at the beginning of a function
2 #pragma omp parallel private(A) shared(B)
3 {
4     // Each thread has its own copy of A, but B is shared
5 }
```

Method 2: using scoping (only C and C++):

```
1 int B; // Variable declared outside of parallel scope - shared by default
2 #pragma omp parallel
3 {
4     int A; // Variable declared inside the parallel scope - always private
5     // Each thread has its own copy of A, but B is shared
6 }
```

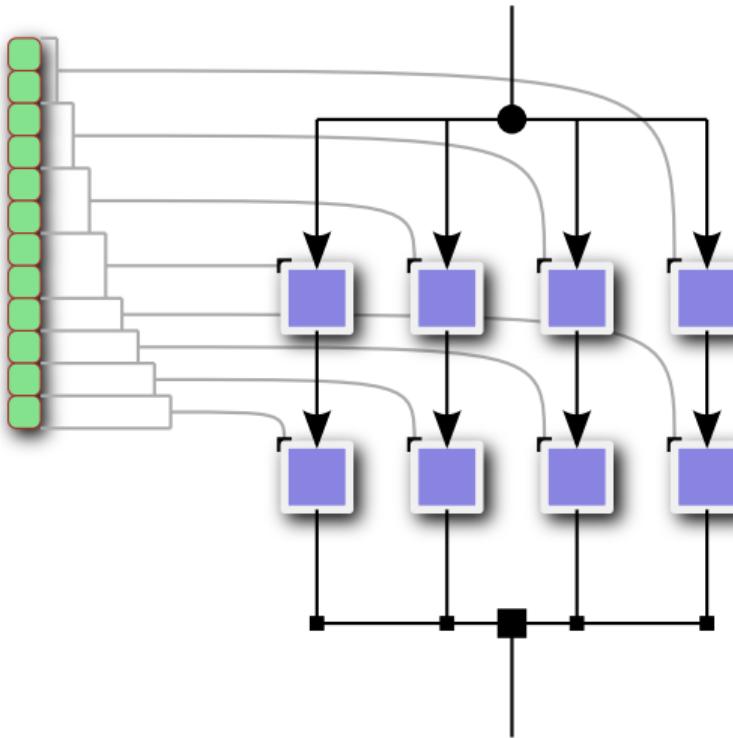


§4. Parallel Loops



Loop-Centric Parallelism: For-Loops in OpenMP

- ▷ Simultaneously launch multiple threads
- ▷ Scheduler assigns loop iterations to threads
- ▷ Each thread processes one iteration at a time



Parallelizing a for-loop.

Loop-Centric Parallelism: For-Loops in OpenMP

The OpenMP library will distribute the iterations of the loop following the `#pragma omp parallel for` across threads.

```
1 #pragma omp parallel for
2 for (int i = 0; i < n; i++) {
3     printf("Iteration %d is processed by thread %d\n",
4            i, omp_get_thread_num());
5     // ... iterations will be distributed across available threads...
6 }
```



Loop-Centric Parallelism: For-Loops in OpenMP

```
1 #pragma omp parallel
2 {
3     // Code placed here will be executed by all threads.
4
5     // Alternative way to specify private variables:
6     // declare them in the scope of pragma omp parallel
7     int private_number=0;
8
9 #pragma omp for
10    for (int i = 0; i < n; i++) {
11        // ... iterations will be distributed across available threads...
12    }
13    // ... code placed here will be executed by all threads
14 }
```



Loop Scheduling Modes in OpenMP

Scheduling Threads Iterations

| | | |
|--------|---|-------------------------|
| static | 0 | 0 1 2 3 4 5 6 7 |
| | 1 | 8 9 10 11 12 13 14 15 |
| | 2 | 16 17 18 19 20 21 22 23 |
| | 3 | 24 25 26 27 28 29 30 31 |

| | | |
|---------|---|------------------|
| dynamic | 0 | 0 7 10 12 17 ... |
| | 1 | 1 4 9 14 18 ... |
| | 2 | 2 5 8 11 15 ... |
| | 3 | 3 6 9 13 16 ... |

| | | |
|--------|---|----------------------------|
| guided | 0 | 0 1 2 3 16 17 24 29 |
| | 1 | 4 5 6 7 20 21 25 30 |
| | 2 | 8 9 10 11 18 19 26 28 |
| | 3 | 12 13 14 15 22 23 27 31 32 |

Time →

Scheduling Threads Iterations

| | | |
|----------|---|-----------------------|
| static,1 | 0 | 0 4 8 12 16 20 24 28 |
| | 1 | 1 5 9 13 17 21 25 29 |
| | 2 | 2 6 10 14 18 22 26 30 |
| | 3 | 3 7 11 15 19 23 27 31 |

| | | |
|-----------|---|---------------------|
| dynamic,2 | 0 | 0 1 10 11 16 17 ... |
| | 1 | 2 3 12 13 18 19 ... |
| | 2 | 4 5 8 9 14 15 ... |
| | 3 | 6 7 20 21 ... |

| | | |
|----------|---|-------------------------|
| guided,2 | 0 | 0 1 2 3 16 17 24 25 |
| | 1 | 4 5 6 7 20 21 26 27 |
| | 2 | 8 9 10 11 18 19 28 29 |
| | 3 | 12 13 14 15 22 23 30 31 |

Time →



§5. Example: Stencil Code



Stencil Operators

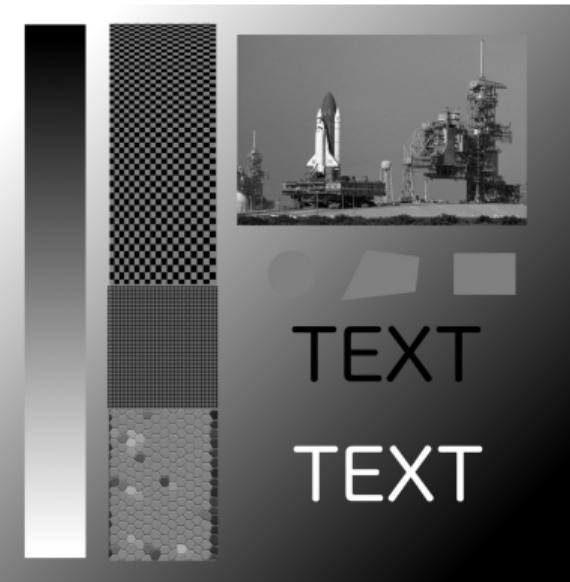
- ▷ Linear systems of equations
- ▷ Partial differential equations

$$Q_{x,y} = c_{00}P_{x-1,y-1} + c_{01}P_{x,y-1} + c_{02}P_{x+1,y-1} + \\ c_{10}P_{x-1,y} + c_{11}P_{x,y} + c_{12}P_{x+1,y} + \\ c_{20}P_{x-1,y+1} + c_{21}P_{x,y+1} + c_{22}P_{x+1,y+1}$$

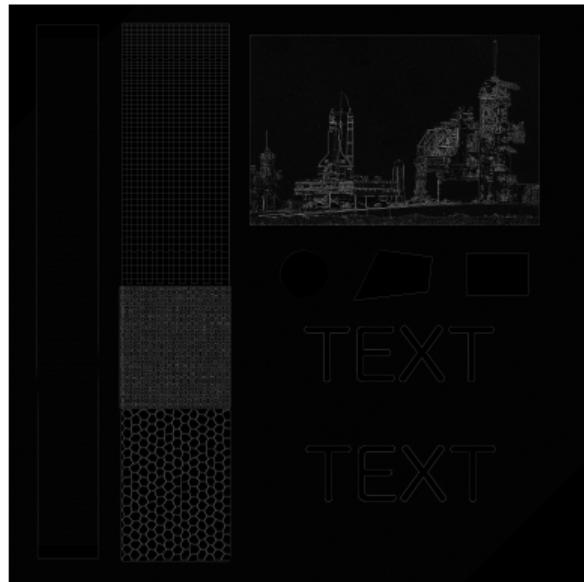
Fluid dynamics, heat transfer, image processing (convolution matrix),
cellular automata.



Edge Detection



$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \rightarrow$$



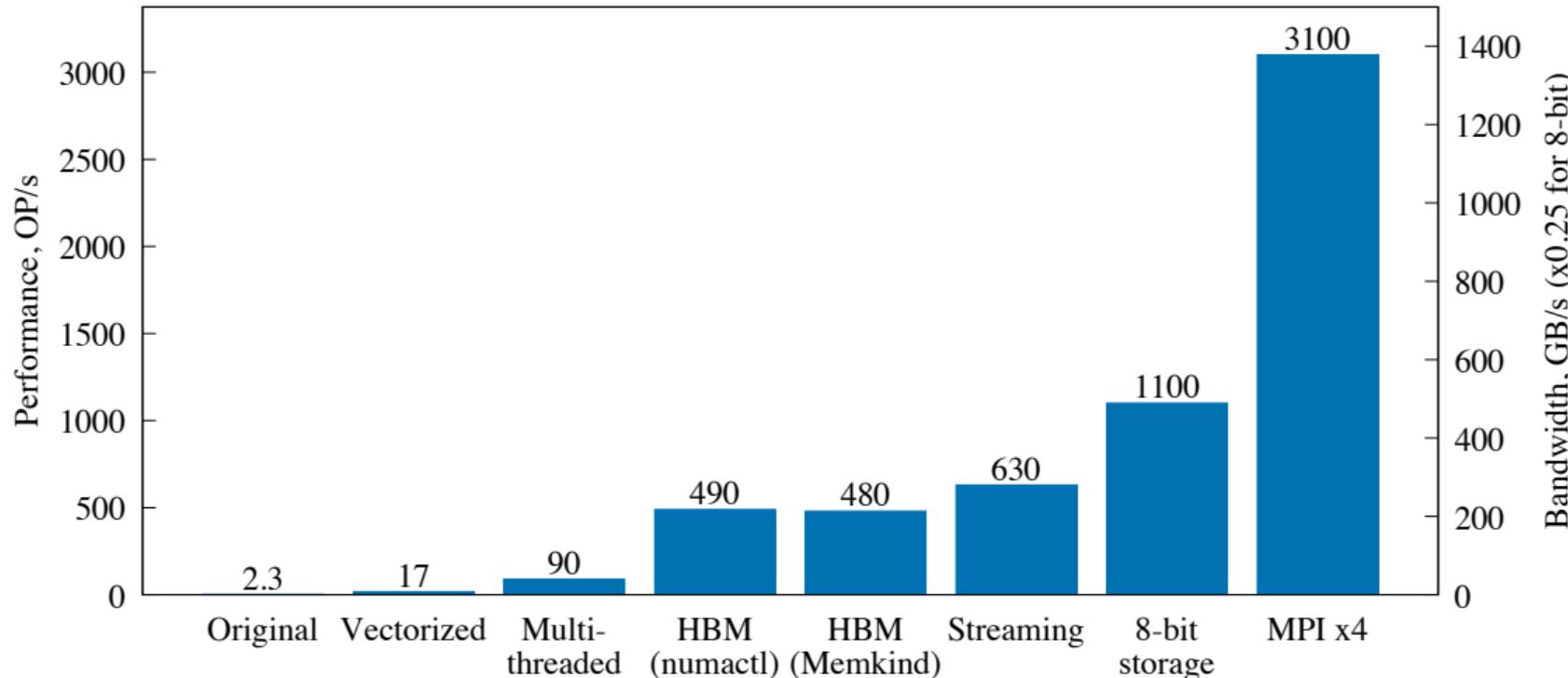
Multithreading

```
user@vega% icpc -c -qopenmp -xMIC-AVX512 stencil.cc
```

```
1 #pragma omp parallel for
2 for (int i = 1; i < height-1; i++)
3 #pragma omp simd
4 for (int j = 1; j < width-1; j++)
5     out[i*width + j] =
6         -in[(i-1)*width + j-1] - in[(i-1)*width + j] - in[(i-1)*width + j+1]
7         -in[(i    )*width + j-1] + 8*in[(i    )*width + j] - in[(i    )*width + j+1]
8         -in[(i+1)*width + j-1] - in[(i+1)*width + j] - in[(i+1)*width + j+1];
```



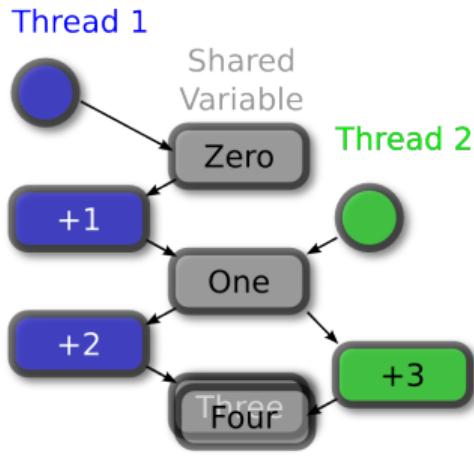
Performance



§6. Data Races and Mutexes



Race Conditions and Unpredictable Program Behavior



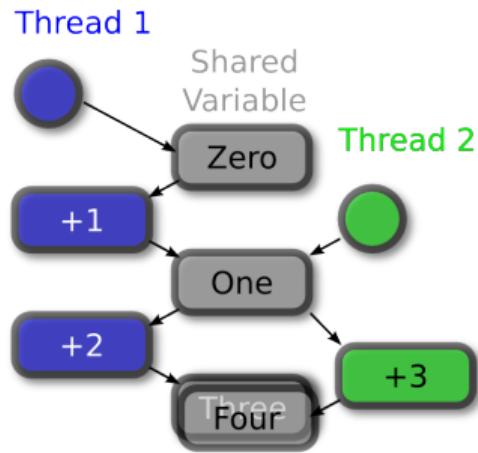
Race Condition!

```
1 int total = 0;  
2 #pragma omp parallel for  
3 for (int i = 0; i < n; i++) {  
4     // Race condition  
5     total = total + i;  
6 }
```

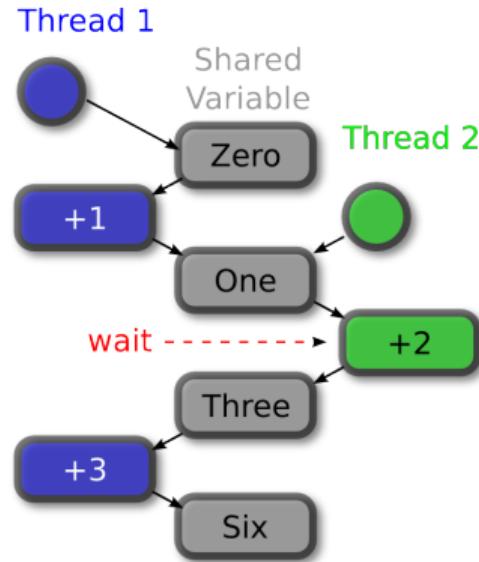
- Occurs when 2 or more threads access the same memory address, and at least one of these accesses is for writing



Mutexes



Race Condition!



- ▷ Mutual exclusion conditions (mutexes) protect data races by serializing code.



Mutexes in OpenMP

```
1 #pragma omp parallel
2 {
3     // parallel code
4 #pragma omp critical
5 {
6     // protected code
7     // multiple lines
8     // many variables
9 }
10 }
```

```
1 #pragma omp parallel
2 {
3     // parallel code
4 #pragma omp atomic
5     // protected code
6     // one line
7     // specific operations
8     // on scalars
9     total += i;
10 }
```

Good parallel codes minimize the use of mutexes.



§7. Parallel Reduction



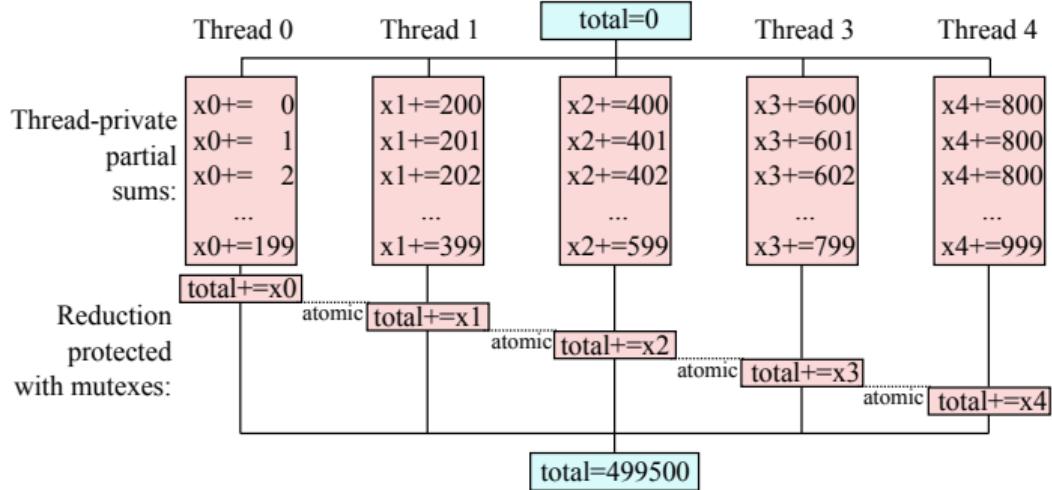
Built-in reduction

```
1 int total = 0;  
2 #pragma omp parallel for reduction(+: total)  
3 for (int i = 0; i < n; i++) {  
    total += i
```



Reduction with Thread-Private Storage

```
1 int total = 0;
2 #pragma omp parallel
3 {
4     int total_thr = 0;
5     #pragma omp for
6     for (int i=0; i<n; i++)
7         total_thr += i;
8
9     #pragma omp atomic
10    total += total_thr;
11
12 }
```



§8. Example: Numerical Integration



Midpoint Rectangle Method

$$I(a, b) = \int_0^a f(x) dx \approx \sum_{i=0}^{n-1} f\left(x_{i+\frac{1}{2}}\right) \Delta x,$$

where

$$\Delta x = \frac{a}{n}, \quad x_{i+\frac{1}{2}} = \left(i + \frac{1}{2}\right) \Delta x.$$



Serial Implementation

```
1 const double dx = a/(double)n;
2 double integral = 0.0;
3
4 for (int i = 0; i < n; i++) {
5     const double xip12 = dx*((double)i + 0.5);
6     const double dI = BlackBoxFunction(xip12)*dx;
7
8     integral += dI;
9 }
```



Unprotected Race Condition

```
1 const double dx = a/(double)n;
2 double integral = 0.0;
3 #pragma omp parallel for
4 for (int i = 0; i < n; i++) {
5     const double xip12 = dx*((double)i + 0.5);
6     const double dI = BlackBoxFunction(xip12)*dx;
7
8     integral += dI;
9 }
```



Excessive Use of Mutexes

```
1 const double dx = a/(double)n;
2 double integral = 0.0;
3 #pragma omp parallel for
4 for (int i = 0; i < n; i++) {
5     const double xip12 = dx*((double)i + 0.5);
6     const double dI = BlackBoxFunction(xip12)*dx;
7 #pragma omp atomic
8     integral += dI;
9 }
```



Built-in reduction

```
1 const double dx = a/(double)n;
2 double integral = 0.0;
3 #pragma omp parallel for reduction(+: integral)
4 for (int i = 0; i < n; i++) {
5     const double xip12 = dx*((double)i + 0.5);
6     const double dI = BlackBoxFunction(xip12)*dx;
7
8     integral += dI;
9 }
```

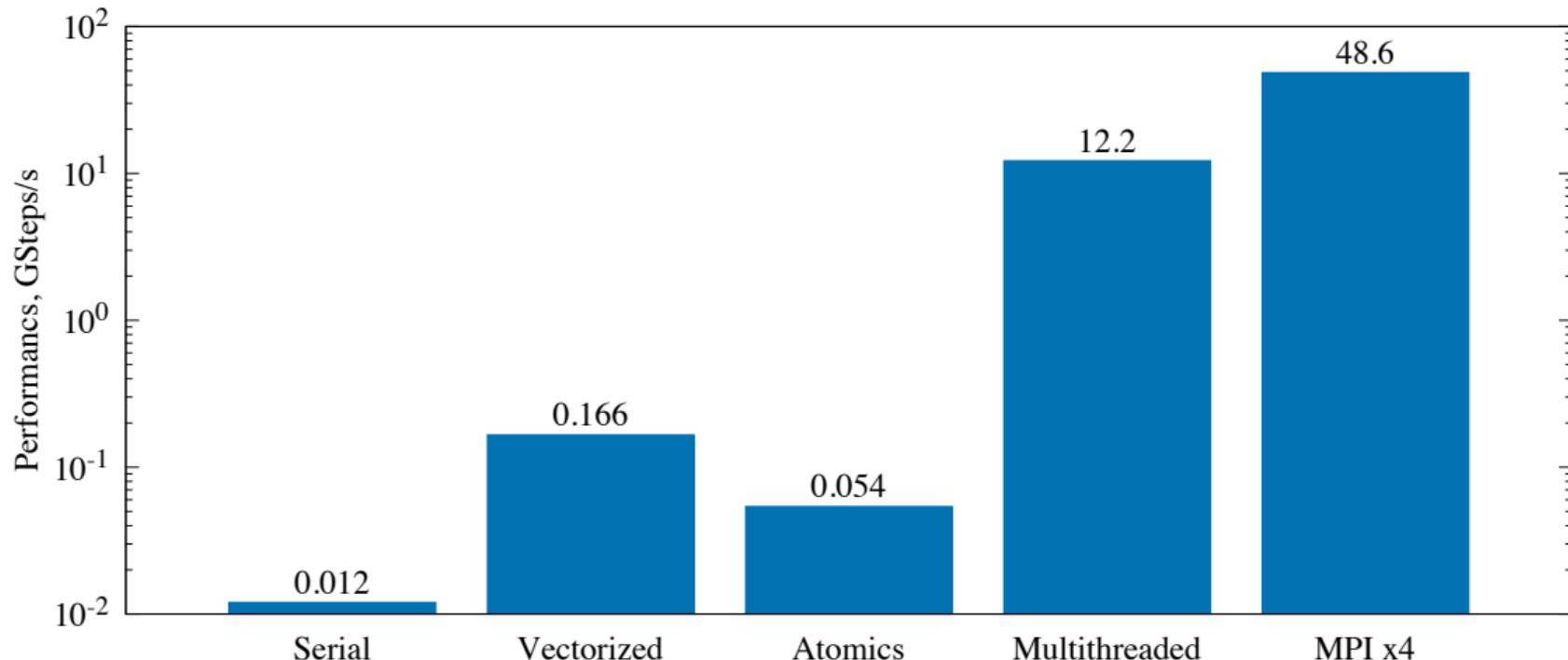


Reduction with Thread-Private Storage

```
1 const double dx = a/(double)n;
2 double integral = 0.0;
3 #pragma omp parallel
4 {
5     double integral_th = 0.0;
6 #pragma omp for
7 for (int i = 0; i < n; i++) {
8     const double xip12 = dx*((double)i + 0.5);
9     const double dI = BlackBoxFunction(xip12)*dx;
10    integral_th += dI;
11 }
12 #pragma omp atomic
13    integral += integral_th;
14 }
```



Performance



§9. Learn more



OpenMP Concepts and Constructs

#pragma omp parallel – create threads

#pragma omp for – process loop with threads

#pragma omp task/taskyield – asynchronous tasks

#pragma omp critical/atomic – mutexes

#pragma omp barrier/taskwait – synchronization points

#pragma omp sections/single – blocks of code for individual threads

#pragma omp flush – enforce memory consistency

#pragma omp ordered – partial loop serialization

OMP_* – environment variables, omp_*() – functions

Click construct names for links to the OpenMP reference from the LLNL

