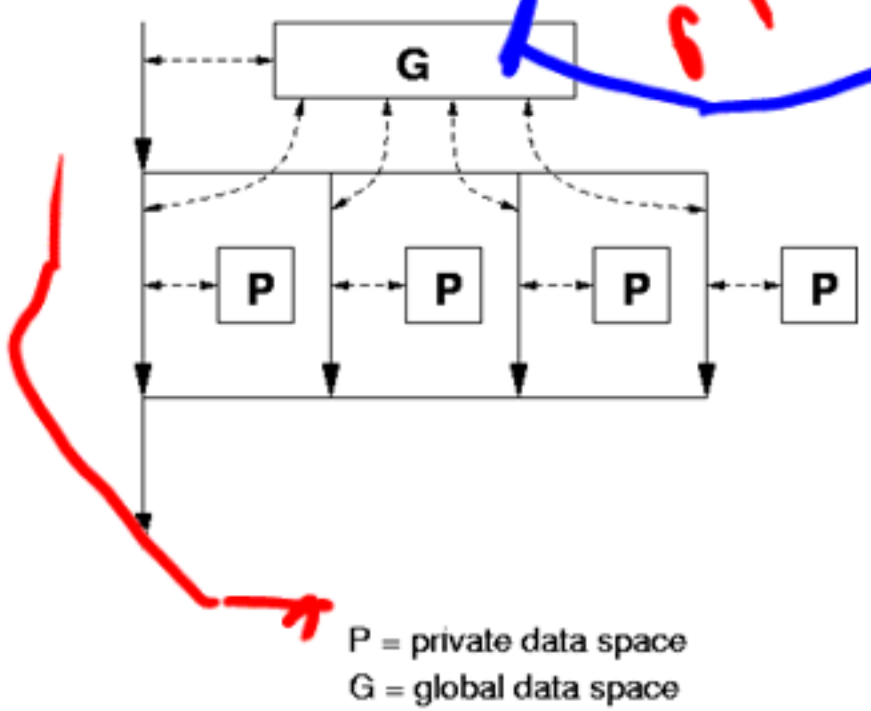
Open MP

Lecture Objectives:

1) Explain the relationship between private and shared variables in OpenMP.
2) Explain the concept of the OpenMP critical pragma
3) Construct a segment of code using a reduction operator and OpenMP.
4) Explain the purpose for the schedule clause in openMP
5) Implement a simple OpenMP algorithm for calculating a mathematical value.

Data model

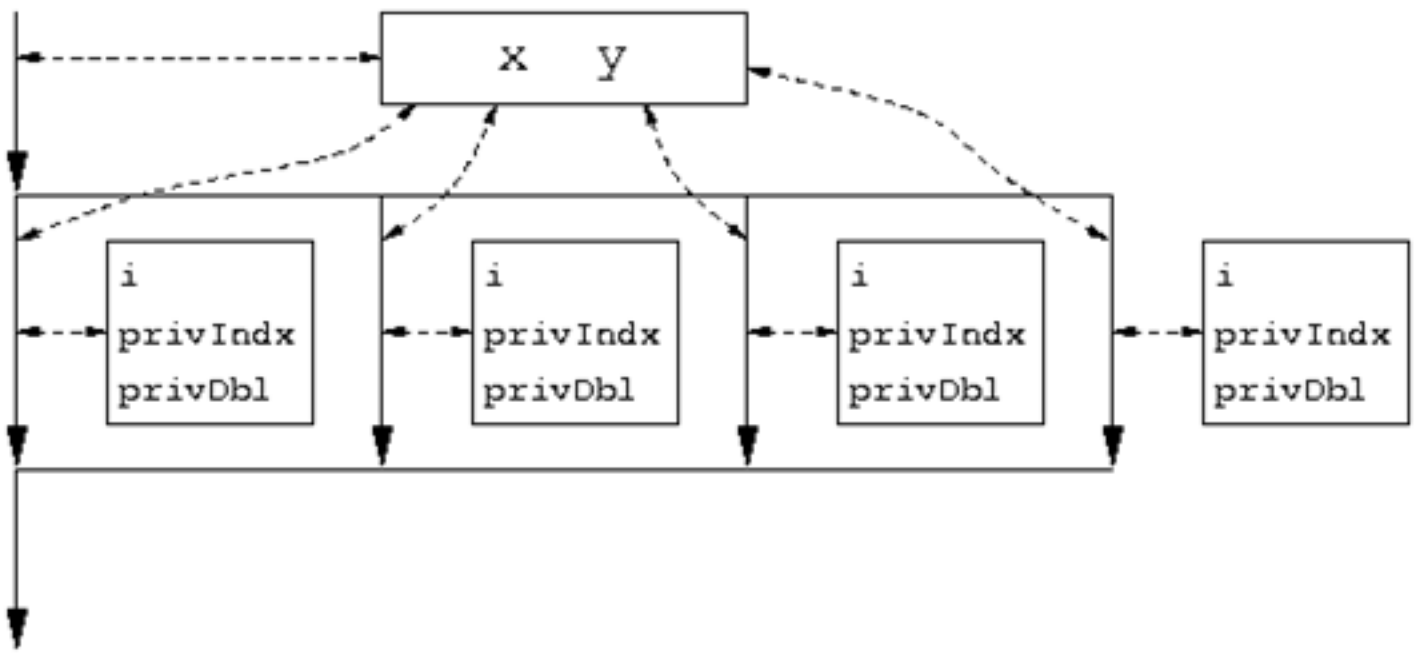


- Private and shared variables
  - Variables in the global data space are accessed by all parallel threads (shared variables).
  - Variables in a thread's private space can only be accessed by the thread (private variables)
    - several variations, depending on the initial values and whether the results are copied outside the region.

```
#pragma omp parallel for private( privIndx, privDbl )
  for ( i = 0; i < arraySize; i++ ) {
     for ( privIndx = 0; privIndx < 16; privIndx++ ) {
      privDbl = ( (double) privIndx ) / 16;
       y[i] = sin( exp( cos( - exp( sin(x[i]) ) ) ) ) + cos(
      privDbl );
   }
}
```

private variables

Parallel for loop index is Private by default.



execution context for "arrayUpdate_II"

- When can we mark a loop a parallel loop?
  - How should we declare variables shared or private?

```
for ( i = 0; i < arraySize; i++ ) {
  for ( privIndx = 0; privIndx < 16; privIndx++ ) {
     privDbl = ( (double) privIndx ) / 16;
     y[i] = sin( exp( cos( - exp( sin(x[i]) ) ) ) ) + cos(   privDbl );
}
}
```

Parallel loop: executing each iteration concurrently is the same as executing each iteration sequentially.

no loop carry dependencies: an iteration does not produce any data that will be consumed by another iteration.

- y[i] is different for each iteration. privDbl is not (must make it private to be correct).

OpenMP directives

- Format:

  #pragma omp directive-name [clause,..] newline

  (use '\' for multiple lines)

- Example:

  #pragma omp parallel default(shared)
  private(beta,pi)

- Scope of a directive is a block of statements {
  …}

Parallel region construct

- A block of code that will be executed by multiple threads.

```
#pragma omp parallel [clause …]
{
  ……
} (implied barrier)
```

Scope for parallel code

Example clauses: if (expression), private *(list),* shared *(list),*
default (shared | none), reduction *(operator: list),*
firstprivate*(list),* lastprivate*(list)*

- if (expression): only in parallel if expression evaluates to true
- private(list): everything private and local (no relation with variables outside the block).
- shared(list): data accessed by all threads
- default (none|shared)

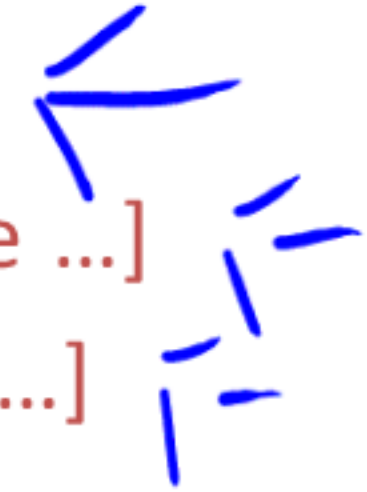
- The reduction clause:

```
Sum = 0.0;
#pragma parallel default(none) shared (n, x) private (I) reduction(+ : sum)
{
  For(I=0; I<n; I++) sum = sum + x(I);
}
```

  - Updating sum must avoid racing condition
  - With the reduction clause, OpenMP generates code such that the race condition is avoided.
  - See example3.c and example3a.c

Work-sharing constructs

- #pragma omp for [clause …]
- #pragma omp section [clause …]
- #pragma omp single [clause …]



- The work is distributed over the threads
- Must be enclosed in parallel region
- No implied barrier on entry, implied barrier on exit (unless specified otherwise)

The omp for directive: example

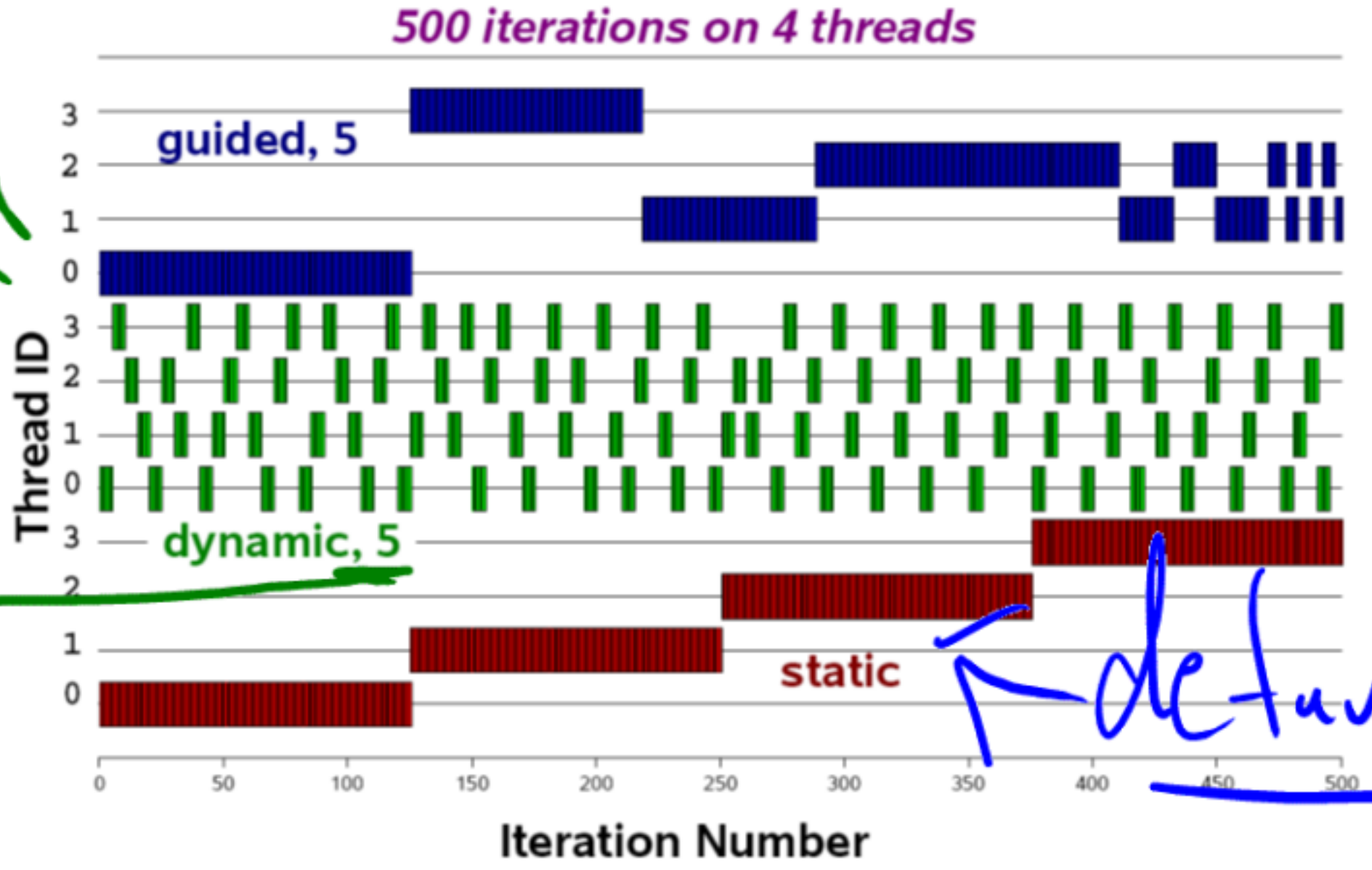
```
#pragma omp parallel default(none)\
        shared(n,a,b,c,d) private(i)
  {
    #pragma omp for nowait
      for (i=0; i<n-1; i++)
          b[i] = (a[i] + a[i+1])/2;

    #pragma omp for nowait
      for (i=0; i<n; i++)
          d[i] = 1.0/c[i];

  } /*-- End of parallel region --*/
```

(implied barrier)

Do not stop here.

- Schedule clause (decide how the iterations are executed in parallel):

schedule (static | dynamic | guided [, chunk])



The omp session clause - example

```
#pragma omp parallel default(none)\
        shared(n,a,b,c,d) private(i)
  {
    #pragma omp sections nowait
    {
      #pragma omp section
        for (i=0; i<n-1; i++)
          b[i] = (a[i] + a[i+1])/2;

      #pragma omp section
        for (i=0; i<n; i++)
          d[i] = 1.0/c[i];

    } /*-- End of sections --*/

  } /*-- End of parallel region --*/
```

```
#pragma omp parallel
#pragma omp for
    for (...)
```

```
#pragma omp parallel for
for (....)
```

Single PARALLEL loop

```
#pragma omp parallel
#pragma omp sections
{ ...}
```

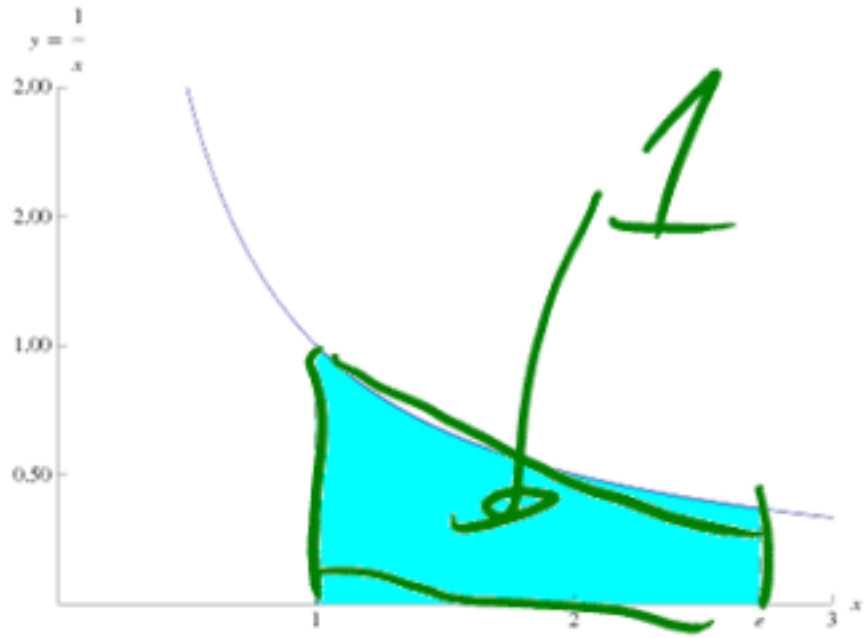
```
#pragma omp parallel sections
{ ... }
```

Single PARALLEL sections

Calculating e

- The constant e is base of the natural logarithm. e is sometimes known as Napier's constant, although its symbol (e) honors Euler.

- e is the unique number with the property that the area of the region bounded by the hyperbola y=1/x, the x-axis, and the vertical lines x=1 and x=e is 1.



$$\int_1^e \frac{dx}{x} = \ln e = 1.$$

$$e = \sum_{k=0} \frac{1}{k!}$$

Calculating e

$$e \equiv \lim_{x \to \infty} \left(1 + \frac{1}{x}\right)^x$$

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}$$

- What can be done in parallel?

Summation can always be done in ll if loops are independ.

Lets calculate this out by hand

- Each student to be assigned a factorial to calculate and a division to make
  - I'll do the summation…

OpenMP code