Multi-core Architecture and Programming

Yang Quansheng(杨全胜)
http://www.njyangqs.com/

School of Computer Science & Engineering
Southeast University

Programming with OpenMP

- Content
  - What is PpenMP
  - Parallel Regions
  - Work-sharing Construct
  - Data scoping to Protect Data
  - Explicit Synchronization
  - Scheduling Clauses
  - Other helpful Construct and Clauses

What is OpenMP

- Compiler directives for multithreaded programming
- Easy to create threaded Fortran and C/C++ codes
- Supports data parallelism model
- Incremental parallelism
    - Combines serial and parallel code in single source

What is OpenMP

C$OMP FLUSH

#pragma omp critical

C$OMP THREADPRIVATE(/ABC/)

CALL OMP_SET_NUM_THREADS(10)

C$OMP parallel do shared(a, b, c)

call omp_test_lock(jlok)

C$OMP MASTER

call OMP_INIT_LOCK (ilok)

MIC

C$OMP SINGLE PRI

/dynamic”

C$OMP PARALLEL

C$OMP ORDERED

C$OMP PARALLE

TIONS

http://www.openmp.org

Current spec is OpenMP 3.0

(combined C/C++ and Fortran)

#pragma omp parallel for private(A, B)

!$OMP BARRIER

C$OMP PARALLEL COPYIN(/blk/)

C$OMP DO lastprivate(XX)

Nthrds = OMP_GET_NUM_PROCS()

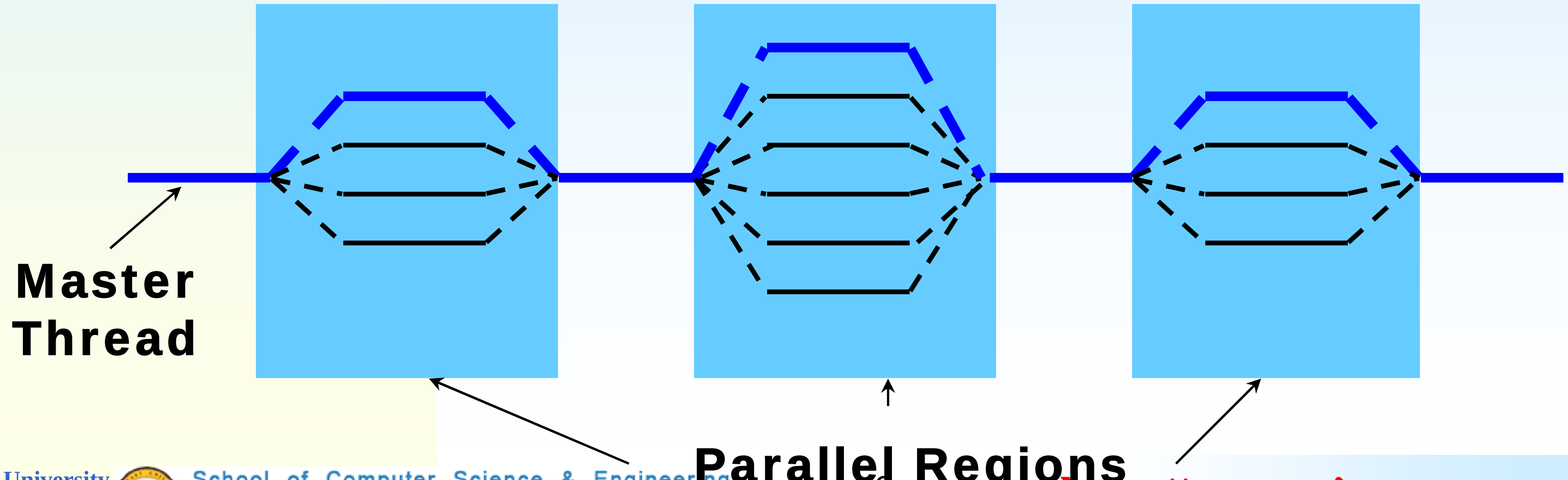
omp_set_lock(lck)

What is OpenMP

- OpenMP* Architecture
- Fork-join model
- Work-sharing constructs
- Data environment constructs
- Synchronization constructs
- Extensive Application Program Interface (API) for finer control

What is OpenMP

- Programming Model
  - Master thread spawns a team of threads as needed
  - Parallelism is added incrementally: the sequential program evolves into a parallel program



What is OpenMP

- OpenMP* Pragma Syntax
  - Most constructs in OpenMP* are compiler directives or pragmas.
    - For C and C++, the pragmas take the form:

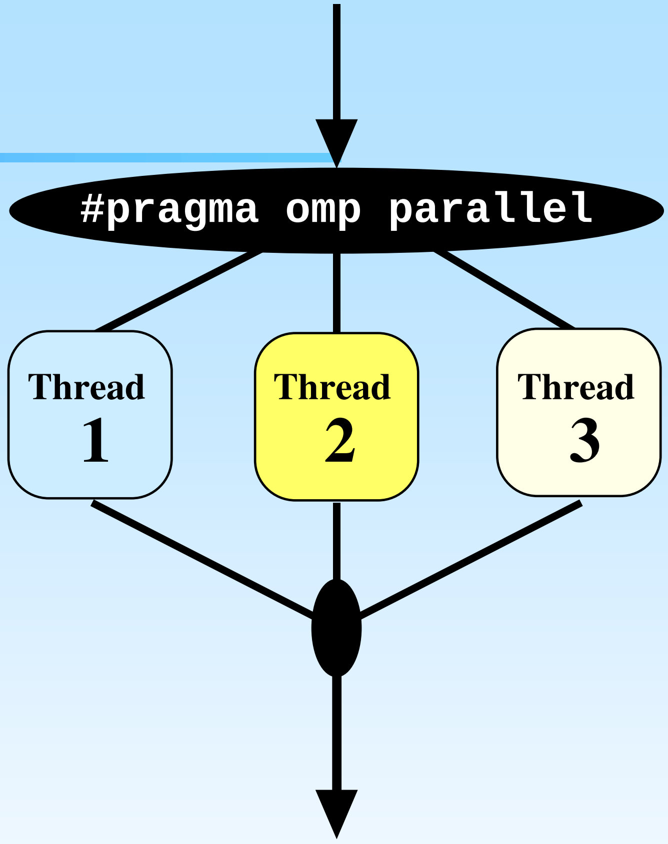
```
#pragma omp construct [clause [clause]…]
```

Programming with OpenMP

- Content
  - What is PpenMP
  - Parallel Regions
  - Work-sharing Construct
  - Data scoping to Protect Data
  - Explicit Synchronization
  - Scheduling Clauses
  - Other helpful Construct and Clauses

Parallel Regions

- Defines parallel region over structured block of code
- Threads are created as 'parallel' pragma is crossed
- Threads block at end of region
- Data is shared among threads unless specified otherwise



C/C++ :

```
#pragma omp parallel
   {
          block
   }
```

Parallel Regions

- How many threads
  - Set environment variable for number of threads

```
set OMP_NUM_THREADS=4
```

  - There is no standard default for this variable
    - Many systems:
      - # of threads = # of processors
      - Intel® compilers use this default

Programming with OpenMP

- Content
  - What is PpenMP
  - Parallel Regions
  - Work-sharing Construct
  - Data scoping to Protect Data
  - Explicit Synchronization
  - Scheduling Clauses
  - Other helpful Construct and Clauses

Work-sharing Construct

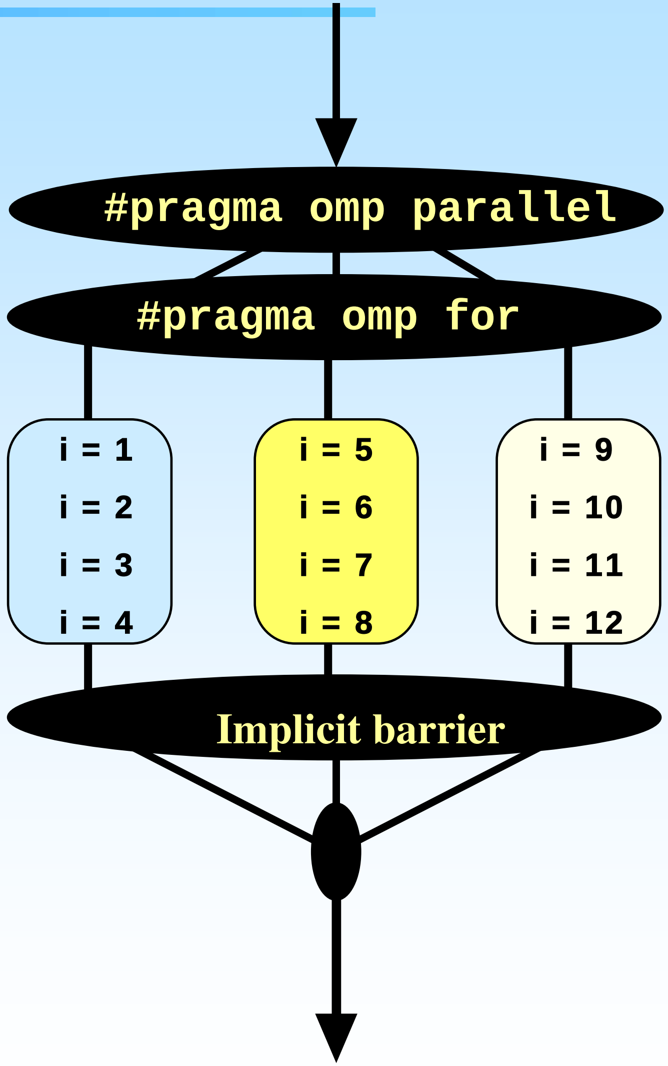
- Manage the threads

```
#pragma omp parallel
#pragma omp for
     for (I=0;I<N;I++){
          Do_Work(I);
     }
```

- Splits loop iterations into threads
- Must be in the parallel region
- Must precede the loop

Work-sharing Construct

```
#pragma omp parallel
#pragma omp for
   for(i = 1, i < 13, i++)
      c[i] = a[i] + b[i]
```



- Threads are assigned an independent set of iterations
- Threads must wait at the end of work-sharing construct

Work-sharing Construct

- Combining pragmas
  - These two code segments are equivalent

```
#pragma omp parallel
{
    #pragma omp for
    for (i=0;i< MAX; i++) {
     res[i] = huge();
    }
}
```

```
#pragma omp parallel for
    for (i=0;i< MAX; i++) {
         res[i] = huge();
    }
```

Programming with OpenMP

- Content
  - What is PpenMP
  - Parallel Regions
  - Work-sharing Construct
  - Data scoping to Protect Data
  - Explicit Synchronization
  - Scheduling Clauses
  - Other helpful Construct and Clauses

Data Scoping to Protect Data

- Data Environment
  - OpenMP uses a shared-memory programming model
    - Most variables are shared by default.
    - Global variables are shared among threads
      - C/C++: File scope variables, static
  - But, not everything is shared...
    - Stack variables in functions called from parallel regions are PRIVATE
    - Automatic variables within a statement block are PRIVATE
    - Loop index variables are private (with exceptions)
      - C/C+: The first loop index variable in nested loops following a #pragma omp for

Data Scoping to Protect Data

- Data Scope Attributes
  - The default status can be modified with

```
default (shared | none)
```

  - Scoping attribute clauses

```
shared(varname,…)
```

```
private(varname,…)
```

Data Scoping to Protect Data

- The Private Clause
  - Reproduces the variable for each thread
    - Variables are un-initialized; C++ object is default constructed
    - Any value external to the parallel region is undefined

```
void* work(float* c, int N) {
  float x, y; int i;
 #pragma omp parallel for private(x,y)
     for(i=0; i<N; i++) {
           x = a[i]; y = b[i];
           c[i] = x + y;
     }
}
```

Data Scoping to Protect Data

- Example: Dot Product

```
float dot_prod(float* a, float* b, int N)
{
  float sum = 0.0;
#pragma omp parallel for shared(sum)
   for(int i=0; i<N; i++) {
     sum += a[i] * b[i];
   }
  return sum;
}
```

What is wrong?

Data Scoping to Protect Data

- Protect Shared Data
  - Must protect access to shared, modifiable data

```
float dot_prod(float* a, float* b, int N)
{
  float sum = 0.0;
#pragma omp parallel for shared(sum)
   for(int i=0; i<N; i++) {
#pragma omp critical
      sum += a[i] * b[i];
   }
  return sum;
}
```

Programming with OpenMP

- Content
  - What is PpenMP
  - Parallel Regions
  - Work-sharing Construct
  - Data scoping to Protect Data
  - Explicit Synchronization
  - Scheduling Clauses
  - Other helpful Construct and Clauses

Explicit Synchronization

- OpenMP* Critical Construct

```
#pragma omp critical [(lock_name)]
```

- Defines a critical region on a structured block

Threads wait their turn –at a time, only one calls consum( ) thereby protecting R1 and R2 from race conditions

Naming the critical construct is optional, but may increase performance

```
float R1,R2;
#pragma omp parallel
{ float B;
#pragma omp for
  for(int i=0; i<niters; i++){
    B = big_job(i);
#pragma omp critical (R1_lock)
    consum (B, &R1);
    A = bigger_job(i)
#pragma omp critical (R1_lock)
    consum (A, &R2);
  }
}
```

Explicit Synchronization

- OpenMP* Reduction Clause

reduction (*op* : *list*)

- The variables in "*list*" must be shared in the enclosing parallel region
- Inside parallel or work-sharing construct:
  - A PRIVATE copy of each list variable is created and initialized depending on the "op"
  - These copies are updated locally by threads
  - At end of construct, local copies are combined through "op" into a single value and combined with the value in the original SHARED variable

Explicit Synchronization

- Reduction Example

```
#pragma omp parallel for reduction(+:sum)
  for(i=0; i<N; i++) {
    sum += a[i] * b[i];
  }
```

- Local copy of *sum* for each thread
- All local copies of *sum* added together and stored in “global” variable

Explicit Synchronization

- C/C++ Reduction Operations
  - A range of associative operands can be used with reduction
  - Initial values are the ones that make sense mathematically

Operand	Initial Value
+	0
*	1
-	0
^	0

Operand	Initial Value
&	~0
|	0
&&	1
||	0

Programming with OpenMP

- Content
  - What is PpenMP
  - Parallel Regions
  - Work-sharing Construct
  - Data scoping to Protect Data
  - Explicit Synchronization
  - Scheduling Clauses
  - Other helpful Construct and Clauses

Scheduling Clauses

- Assigning Iterations
  - The Schedule clause affects how loop iterations are mapped onto threads

schedule(*static [, chunk]*)

- Blocks of iterations of size "chunk" to thread
- Round robin distribution

schedule(*dynamic [, chunk]*)

- Threads grab "chunk" iterations
- When done with iterations, thread request next set

schedule(*guided [, chunk]*)

- Dynamic schedule starting with large block
- Size of the blocks shrink; no smaller than "chunk"

Scheduling Clauses

- Which Schedule to Use

Schedule Clause	When To Use
STATIC	Predictable and similar work per iteration
DYNAMIC	Unpredictable, highly variable work per iteration
GUIDED	Special case of dynamic to reduce scheduling overhead

Scheduling Clauses

- Schedule Clauses Example

```
#pragma omp parallel for schedule(static, 8)
  for(I = start; I <= end; i+=2 ) {
    if (TestForPrime(i) ) gPrimesFound++;
  }
```

- Iterations are divided into chunks of 8
  - If start = 3, then first chunk is i={3,5,7,9,11,13,15,17}

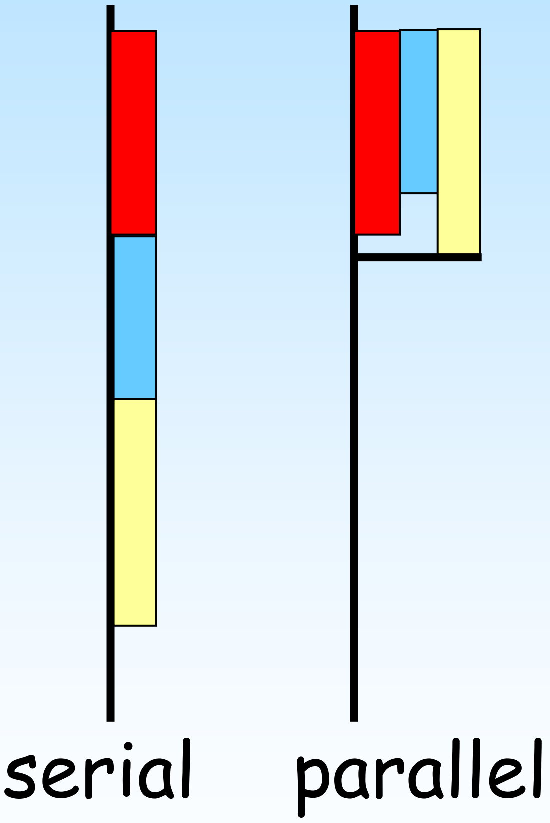
Programming with OpenMP

- Content
  - What is PpenMP
  - Parallel Regions
  - Work-sharing Construct
  - Data scoping to Protect Data
  - Explicit Synchronization
  - Scheduling Clauses
  - Other helpful Construct and Clauses

Other helpful Construct and Clauses

- Parallel Sections

```
#pragma omp parallel sections
{
    #pragma omp section
    phase1();
    #pragma omp section
    phase2();
    #pragma omp section
    phase3();
}
```



Other helpful Construct and Clauses

- Single Construct
  - Denotes block of code to be executed by only one thread
    - First thread to arrive is chosen
  - Implicit barrier at end

```
#pragma omp parallel
{
  DoManyThings();
#pragma omp single
  {
   ExchangeBoundaries();
  }  // threads wait here for single
  DoManyMoreThings();
}
```

Other helpful Construct and Clauses

- Master Construct
  - Denotes block of code to be executed only by the master thread
  - No implicit barrier at end

```
#pragma omp parallel
{
  DoManyThings();
#pragma omp master
  {                    // if not master skip to next stmt
    ExchangeBoundaries();
  }
  DoManyMoreThings();
}
```

Other helpful Construct and Clauses

- Implicit Barriers
  - Several OpenMP* constructs have implicit barriers
    - parallel
    - for
    - single
  - Unnecessary barriers hurt performance
    - Waiting threads accomplish no work!
  - Suppress implicit barriers, when safe, with the nowait clause

Other helpful Construct and Clauses

- Nowait Clause

```
#pragma omp for
nowait
   for(...)
      {...};
```

```
#pragma omp single  nowait
  { [...] }
```

Use when threads would wait between independent computations

```
#pragma omp for  schedule (dynamic, 1) nowait
   for(int i=0; i<n; i++)
      a[i] = bigFunc1(i);

#pragma omp for  schedule (dynamic, 1)
   for(int j=0; j<m; j++)
      b[j] = bigFunc2(j);
```

Other helpful Construct and Clauses

- Barrier Construct
  - Explicit barrier synchronization
  - Each thread waits until all threads arrive

```
#pragma omp parallel shared (A, B, C)
{
        DoSomeWork(A,B);
        printf("Processed A into B\n");
#pragma omp barrier
        DoSomeWork(B,C);
        printf("Processed B into C\n");
}
```

Other helpful Construct and Clauses

- Atomic Construct
  - Special case of a critical section
  - Applies only to simple update of memory location

```
#pragma omp parallel for shared(x, y, index, n)
  for (i = 0; i < n; i++) {
    #pragma omp atomic
      x[index[i]] += work1(i);
    y[i] += work2(i);
  }
```

Other helpful Construct and Clauses

- OpenMP* API
  - Get the thread number within a team

```
int omp_get_thread_num(void)
```

  - Increment semaphore (Post operation)

```
int omp_get_num_thread (void)
```

  - Usually not needed for OpenMP codes
    - Can lead to code not being serially consistent
    - Does have specific uses (debugging)
    - Must include a header file

```
#include <omp.h>
```

Programming with OpenMP What's Been Covered

- OpenMP* is:
  - A simple approach to parallel programming for shared memory machines
- We explored basic OpenMP coding on how to:
  - Make code regions parallel (omp parallel)
  - Split up work (omp for)
  - Categorize variables (omp private….)
  - Synchronize (omp critical…)
- We reinforced fundamental OpenMP concepts through several labs

Advanced Concepts

More OpenMP*

- Data environment constructs
  - FIRSTPRIVATE
  - LASTPRIVATE
  - THREADPRIVATE

Firstprivate Clause

- Variables initialized from shared variable
- C++ objects are copy-constructed

```
incr=0;
#pragma omp parallel for firstprivate(incr)
for (I=0;I<=MAX;I++) {
        if ((I%2)==0) incr++;
        A(I)=incr;
}
```

Lastprivate Clause

- Variables update shared variable using value from last iteration
- C++ objects are updated as if by assignment

```
void sq2(int n, double *lastterm)
{
  double x; int i;
  #pragma omp parallel
  #pragma omp for lastprivate(x)
  for (i = 0; i < n; i++){
     x = a[i]*a[i] + b[i]*b[i];
     b[i] = sqrt(x);
  }
  lastterm = x;
}
```

Threadprivate Clause

- Preserves global scope for per-thread storage
- Legal for name-space-scope and file-scope
- Use copyin to initialize from master thread

```
struct Astruct A;
#pragma omp threadprivate(A)
…

#pragma omp parallel
   copyin(A)
  do_something_to(&A);
…

#pragma omp parallel
  do_something_else_to(&A);
```

Private copies of “A” persist between regions

Performance Issues

- Idle threads do no useful work
- Divide work among threads as evenly as possible
  - Threads should finish parallel tasks at same time
- Synchronization may be necessary
  - Minimize time waiting for protected resources

Load Imbalance

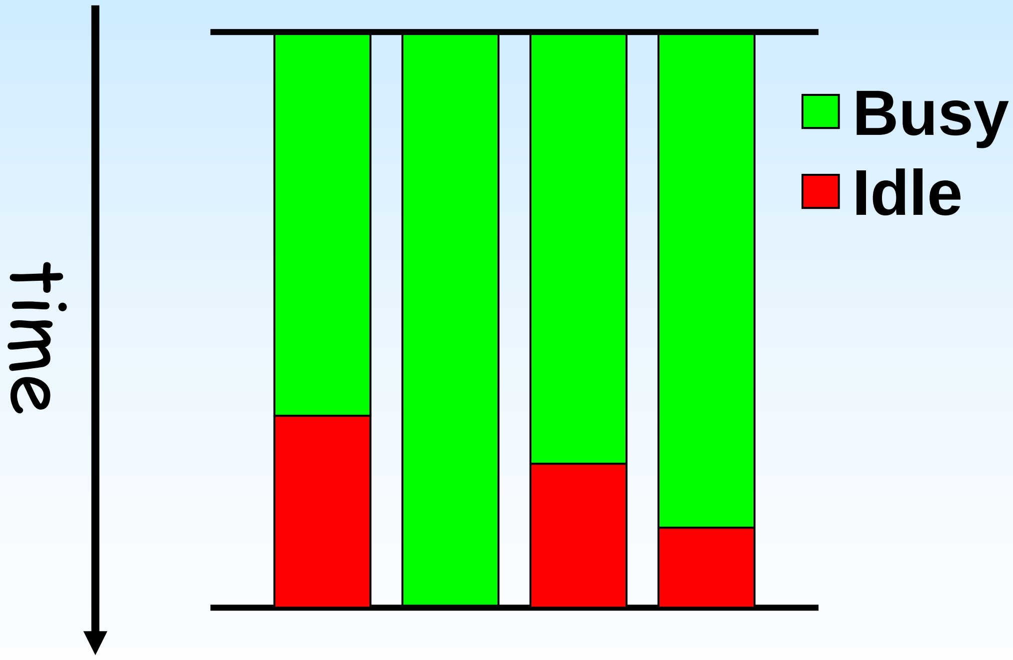
- Unequal work loads lead to idle threads and wasted time.

```
#pragma omp parallel
{

#pragma omp for
  for( ; ; ){

  }

}
```

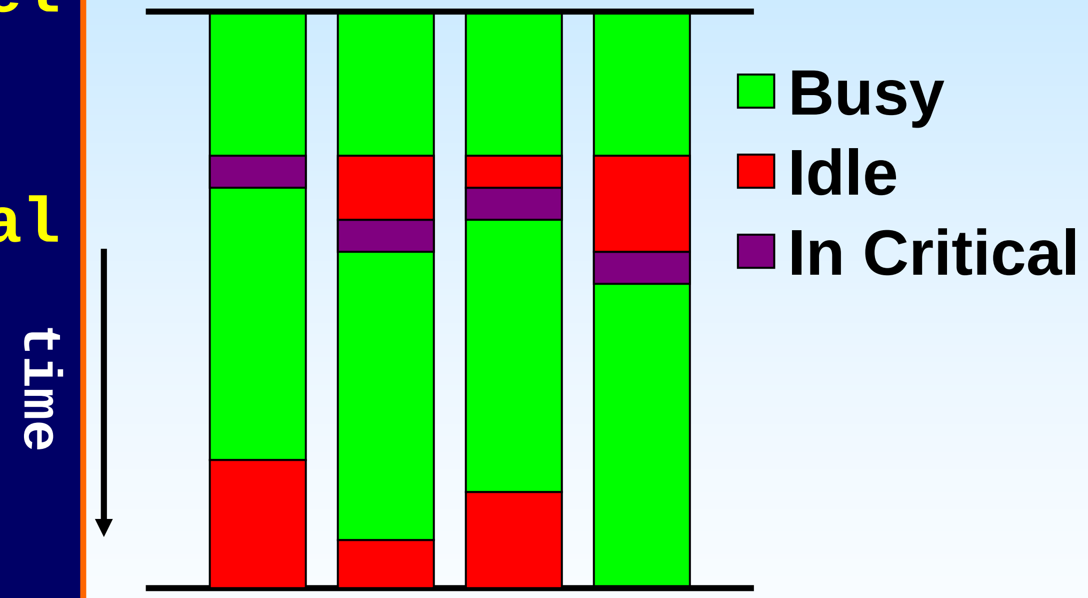


Synchronization

- Lost time waiting for locks

```
#pragma omp parallel
{

#pragma omp critical
  {
    ...
  }
  ...
}
```



Performance Tuning

- Profilers use sampling to provide performance data.
- Traditional profilers are of limited use for tuning OpenMP*:
  - Measure CPU time, not wall clock time
  - Do not report contention for synchronization objects
  - Cannot report load imbalance
  - Are unaware of OpenMP constructs

Programmers need profilers specifically designed for OpenMP.

Static Scheduling: Doing It By Hand

- Must know:
  - Number of threads (Nthrds)
  - Each thread ID number (id)
- Compute start and end iterations:

```
#pragma omp parallel
{
        int i, istart, iend;
        istart = id * N / Nthrds;
        iend = (id+1) * N / Nthrds;
        for(i=istart;i<iend;i++){
          c[i] = a[i] + b[i];}
}
```