**Types of synchronization:**

**High level synchronization:**

- **critical**(it does mutual exclusion, it means one thread is allowed to execute a block of code, then when the thread that was working in that block of code is done, the next one gets ready to execute it)

example:

float res;

#pragma omp parallel

{

float B;

int i, id, nthrds;

id = omp\_get\_thread\_num();

nthrds = omp\_get\_num\_threads();

for(i=id; i< ntires; i+nthrds)

{

B = big\_job(i);

#pragma omp critical

res+=consume(B);

}

}

- **atomic**

- **barrier**(it makes all threads wait till the last one is complete)

example:

#pragma omp parallel

{

int id=omp\_get\_thread\_num();

A[id] = big\_cal1(id);

#pragma omp barrier // all threads wait till everyone finishes

B[id] = big\_calc2(id,A)

}

**low level synchronization:**

- Flush

- locks(both simple and nested)

09 part 1 module 5

The schedule clause:

Dynamic: Unpredictable, highly variable work per iteration. Most work at runtime: complex scheduling used at runtime.

Static: Predetermined and predictable by the programmer. Least work at runtime: scheduling done at compile time.

#pragma omp parallel for == #pragma omp parallel // those are equivalent

#pragma omp for

working with loops

example 1:

int I,j, A[MAX];

j = 5;

for(i=0; i<MAX; i++)

{

j+=2; // in this case we can’t work in parallel with ‘j’ because it depends

// on the previous variable that increments it value.

A[i] = big(j);

}

int i, j, A[MAX];

#pragma omp parallel for // in this version I have no dependencies and we can run it in // parallel

for(i=0; i<MAX; i++)

{

int j=5 + 2\*(i+1); // we declare j inside of the parallel for loop

A[i] = big(j);

}

-----------------------------------------------------------------

example 2:

double ave=0.0, A[MAX];

int I;

for(i=0; i<MAX;i++)

{

ave+=A[i];

}

ave = ave/MAX;

\* always try to find the compute intense loops, modify them so to not have current dependencies, when we can run them in any order and independently.

- Reduction in openmp: A local copy of each list variable is made and initialize depending on the “op” (operator): example 0 for “+”, 1 for “\*”. Updates occur on the local copy. Local copies are reduced into a single value and combined with the original global value.

syntax: #pragma omp parallel reduction(op:list)

code example:

double ave=0.0, A[MAX];

int I;

#pragma omp parallel for reduction (+:ave) // it makes local copies of AVE then it adds all // together with the operator “+”.

for(i=0; i<MAX; i++)

ave+=A[i];

ave = ave/MAX;