OpenMP

Motivation

- Every application uses loops
- Performance can be improved on multi core systems
- Complex parallel programming models
- Crossplatform multithreading support
- Huge amounts of code must be change

New bugs

What is OpenMP?

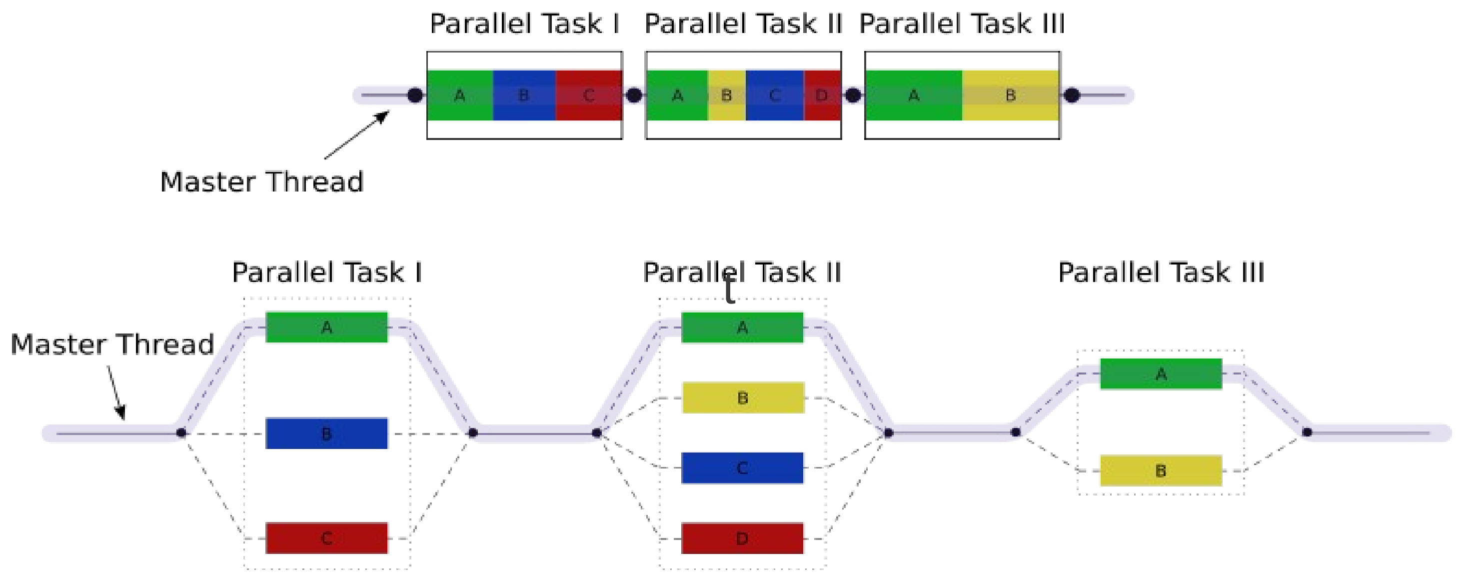
- Compiler directives for C/C++ and Fortran

  ```
  #pragma omp parallel for
  ```

- Can be enable with -fopenmp , /openmp
- Available for windows and unix systems
- Loops can be parallelized automatically
- In general existing serial code doesn't need to be modified
- If the compiler doesn't support OpenMP directives are treated as comments

OpenMP

- The main thread forks several slave threads that execute a part of the workload in parallel
- The number of threads is determined by the system



Source: http://en.wikipedia.org/wiki/File:Fork_join.svg

Example: Matrixmultiplication(1)

Sequential version

```
for (int i = 0; i < MATRIX_A_HEIGHT; i++)
{
    for (int j = 0; j < MATRIX_B_WIDTH; j++)
    {
        C[i][j] = 0.f;
        for (int k = 0; k < MATRIX_A_WIDTH; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Example: Matrixmultiplication(2)

OpenMP version:

```
#pragma omp parallel for schedule(dynamic)
    for (int i = 0; i < MATRIX_A_HEIGHT; i++)
    {
        for (int j = 0; j < MATRIX_B_WIDTH; j++)
        {
            C[i][j] = 0.f;
            for (int k = 0; k < MATRIX_A_WIDTH; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
```

Always the outer loop should be parallelized with the OpenMP directive

OpenMP clauses

Scheduling:

schedule(dynamic)

schedule(static,chunk)

Variable scope:

shared(var)

private(var)

Synchronisation:

#pragma omp barrier

#pragma omp critical

#pragma omp ordered

#pragma omp master

Reduction:

reduction(operator:var)

Examples

Dotproduct

```
int n = 100;
int chunk = 10;
int result = 0.0;

#pragma omp parallel for   \
    schedule(static,chunk) \
    reduction(+:result)    \
for (int i = 0; i < n; i++)
      result += (a[i] * b[i]);
```

- more advanced functions are available through omp.h
- functioncalls must be threadsave

Critical section

```
#include <omp.h>

float x = 0.0f;
float A[1024];
float B[1024];

omp_set_num_threads(4);

#pragma omp parallel shared(x)
{
  int t_ID =
          omp_get_thread_num();
  float res = foo(A,t_ID);
  #pragma omp critical
  x = x + result;
  #pragma omp barrier
  bar(x,B);
}
```

Summary

+

- Very simple to use
- Datadecomposition and distribution handled automatically by directives
- Unified code for parallel and serial application

–

- Lack of fine grained control mechanisms
- Race conditions
- High chance writing false sharing code

Fragen?