Parallel Programming Tutorial - More on OpenMP

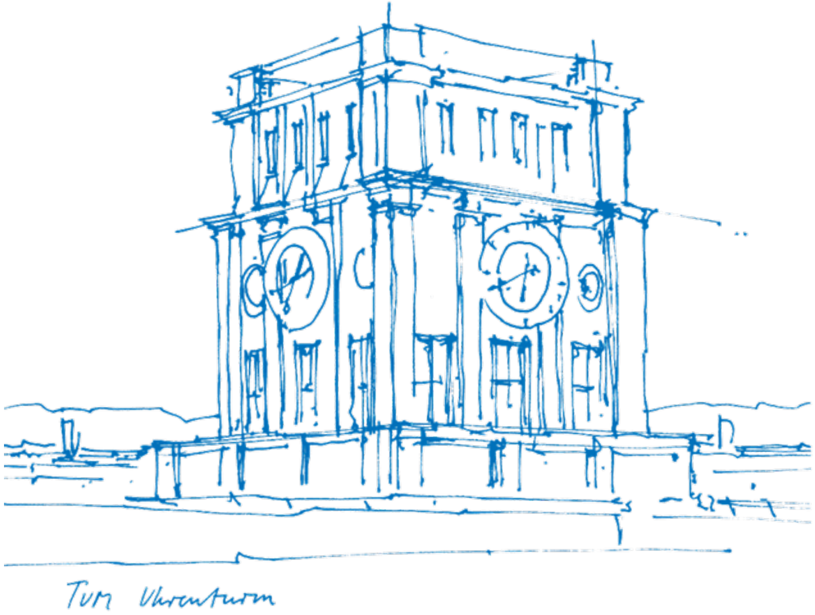
Bengisu Elis, M.Sc.

Philipp Czerner

Chair for Computer Architecture and Parallel Systems (Prof. Schulz)

Technical University of Munich

5. Juni 2019



Organizational

- 12. June (next Wednesday) : Lecture instead of Tutorial session. (Please check the schedule)
- 14. June (next Friday) : In class Q&A session cancelled.

Solution for Assignment 4

Assignment 4

```c
#include <omp.h>

int parallel_traverse(tree *node) {
    if (node == NULL) return 0;

    int father_iq, mother_iq;

    #pragma omp task shared(father_iq)
    father_iq = parallel_traverse(node->father);
    mother_iq = parallel_traverse(node->mother);

    #pragma omp taskwait

    node->IQ = compute_IQ(
        node->data, father_iq, mother_iq
    );
    genius[node->id] = node->IQ;
    return node->IQ;
}
```

```c
int traverse(tree *node, int numThreads) {
    #pragma omp parallel num_threads(numThreads)
    {
      #pragma omp single
      parallel_traverse(node);
    }
    return node->IQ;
}
```

- Helper function for the recursion, so that we can set up the threads
- Use tasks for parallelism

Assignment 4

```
int parallel_traverse(tree *node) {
    if (node == NULL) return 0;

    int father_iq, mother_iq;

    #pragma omp parallel sections
    {
        #pragma omp section
        father_iq = parallel_traverse(node->father);
        #pragma omp section
        mother_iq = parallel_traverse(node->mother);
    }

    node->IQ = compute_IQ(node->data, father_iq, mother_iq);
    genius[node->id] = node->IQ;
    return node->IQ;
}
```

What about this?

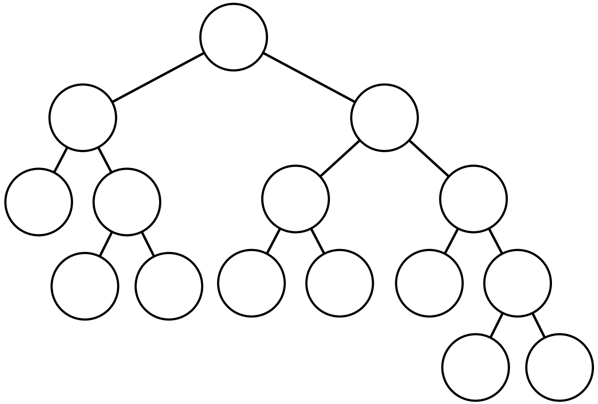
Assignment 4

```c
int parallel_traverse(tree *node) {
    if (node == NULL) return 0;

    int father_iq, mother_iq;

    #pragma omp parallel sections
    {
        #pragma omp section
        father_iq = parallel_traverse(node->father);
        #pragma omp section
        mother_iq = parallel_traverse(node->mother);
    }

    node->IQ = compute_IQ(node->data, father_iq, mother_iq);
    genius[node->id] = node->IQ;
    return node->IQ;
}
```



Does not work! Tree is unbalanced.

What about this?

Assignment 4 – Trick

```c
uint64_t val = ...
for (int i = 0; i < 200000; ++i) {
    val ^= val << 13;
    val ^= val >> 7;
    val ^= val << 17;
}
```

- Expensive part of the computation
- xorshift random number generator
- Can be optimised using linear algebra!

Assignment 4 – Trick

```
uint64_t val = ...
for (int i = 0; i < 200000; ++i) {
    val ^= val << 13;
    val ^= val >> 7;
    val ^= val << 17;
}
```

- Expensive part of the computation
- xorshift random number generator
- Can be optimised using linear algebra!

```
uint64_t val = ...
val =    (val>>0&1)*0xc47563c1f4a1b004ull
       ^ (val>>1&1)*0x97491f0a1292b246ull
       ^ (val>>2&1)*0x738047610b2051d4ull
       ^ (val>>3&1)*0x27897897d28cd376ull
       ^ (val>>4&1)*0x23fd69ab27cab726ull
       ^ (val>>5&1)*0xac2edfc94cb05e39ull
         ...
       ^ (val>>62&1)*0x719f95f67e9e31cdull
       ^ (val>>63&1)*0xd8bea87b21e77e2ull;
```

OpenMP Wrap-Up

Nested parallel regions revisited

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

int main(){

    int num_threads=4;
    omp_set_num_threads(num_threads);

    #pragma omp parallel
    {
        #pragma omp parallel for
        for (int i = 0; i < num_threads; i++)
        {
            #pragma omp critical
            std::cout << "My id is: "
                      << omp_get_thread_num() << std::endl;
        }
    }
}
```

Nested parallel regions revisited

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

int main(){

    int num_threads=4;
    omp_set_num_threads(num_threads);

    #pragma omp parallel
    {
        #pragma omp parallel for
        for (int i = 0; i < num_threads; i++)
        {
            #pragma omp critical
            std::cout << "My id is: "
                      << omp_get_thread_num() << std::endl;
        }
    }
}
```

```
./example4

My id is: 0
My id is: 0
My id is: 0
My id is: 0
My id is: 0
My id is: 0
My id is: 0
My id is: 0
My id is: 0
My id is: 0
My id is: 0
My id is: 0
My id is: 0
My id is: 0
My id is: 0
My id is: 0
```

Nested parallel regions revisited (Cont.)

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

int main(){

    int num_threads=4;
    omp_set_num_threads(num_threads);
    omp_set_nested(1);

    #pragma omp parallel
    {
        #pragma omp parallel for
        for (int i = 0; i < num_threads; i++)
        {
            #pragma omp critical
            std::cout << "My id is: "
                      << omp_get_thread_num() << std::endl;
        }
    }
}
```

Nested parallel regions revisited (Cont.)

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

int main(){

    int num_threads=4;
    omp_set_num_threads(num_threads);
    omp_set_nested(1);

    #pragma omp parallel
    {
        #pragma omp parallel for
        for (int i = 0; i < num_threads; i++)
        {
            #pragma omp critical
            std::cout << "My id is: "
                      << omp_get_thread_num() << std::endl;
        }
    }
}
```

```
./example5

My id is: 1
My id is: 0
My id is: 2
My id is: 3
My id is: 1
My id is: 2
My id is: 0
My id is: 1
My id is: 1
My id is: 0
My id is: 3
My id is: 2
My id is: 3
My id is: 0
My id is: 3
My id is: 2
```

Quiz; What is the problem with this program?

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

int main(){

    int id;
    #pragma omp parallel num_threads(4)
    {
        id = omp_get_thread_num();
        #pragma omp critical
        std::cout << "My id is: " << id << std::endl;
    }

}
```

Quiz; What is the problem with this program?

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

int main(){

    int id;
    #pragma omp parallel num_threads(4)
    {
        id = omp_get_thread_num();
        #pragma omp critical
        std::cout << "My id is: " << id << std::endl;
    }

}
```

```
./example

My id is: 0
My id is: 0
My id is: 3
My id is: 2
```

Quiz; What is the problem with this program?

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

int main(){

    int id;
    #pragma omp parallel num_threads(4)
    {
        id = omp_get_thread_num();
        #pragma omp critical
        std::cout << "My id is: " << id << std::endl;
    }

}
```

```
./example

My id is: 0
My id is: 0
My id is: 3
My id is: 2
```

```
./example

My id is: 2
My id is: 2
My id is: 0
My id is: 0
```

Quiz; What is the problem with this program? (Cont.)

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

int main(){

    int id;
    #pragma omp parallel num_threads(4) private(id)
    {
        id = omp_get_thread_num();
        #pragma omp critical
        std::cout << "My id is: " << id << std::endl;
    }

}
```

Quiz; What is the problem with this program? (Cont.)

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

int main(){

    int id;
    #pragma omp parallel num_threads(4) private(id)
    {
        id = omp_get_thread_num();
        #pragma omp critical
        std::cout << "My id is: " << id << std::endl;
    }

}
```

```
./example

My id is: 3
My id is: 0
My id is: 2
My id is: 1
```

Quiz; Data scoping

```
int main (){
    int a =1, b =2, c =3;
    #pragma omp parallel num_threads(4) private(b) firstprivate(c)
    {
        #pragma omp critical
        {
            a++;


        }
        #pragma omp barrier
        if (omp_get_thread_num()==0){
            printf("a: %d\n", a);
            printf("b: %d\n", b);
            printf("c: %d\n", c);
        }
    }

    printf("a: %d\n", a);
    printf("b: %d\n", b);
    printf("c: %d\n", c);
    return 0;
}
```

Quiz; Data scoping

```
int main (){
    int a =1, b =2, c =3;
    #pragma omp parallel num_threads(4) private(b) firstprivate(c)
    {
        #pragma omp critical
        {
            a++;  // -> shared
            b++;

        }
        #pragma omp barrier
        if (omp_get_thread_num()==0){
            printf("a: %d\n", a);
            printf("b: %d\n", b);
            printf("c: %d\n", c);
        }
    }

    printf("a: %d\n", a);
    printf("b: %d\n", b);
    printf("c: %d\n", c);
    return 0;
}
```

Quiz; Data scoping

```c
int main (){
    int a =1, b =2, c =3;
    #pragma omp parallel num_threads(4) private(b) firstprivate(c)
    {
        #pragma omp critical
        {
            a++;  // -> shared
            b++;  // -> private
            c++;
        }
        #pragma omp barrier
        if (omp_get_thread_num()==0){
            printf("a: %d\n", a);
            printf("b: %d\n", b);
            printf("c: %d\n", c);
        }
    }

    printf("a: %d\n", a);
    printf("b: %d\n", b);
    printf("c: %d\n", c);
    return 0;
}
```

Quiz; Data scoping

```c
int main (){
    int a =1, b =2, c =3;
    #pragma omp parallel num_threads(4) private(b) firstprivate(c)
    {
        #pragma omp critical
        {
            a++;  // -> shared
            b++;  // -> private
            c++;  // -> firstprivate
        }
        #pragma omp barrier
        if (omp_get_thread_num()==0){
            printf("a: %d\n", a);
            printf("b: %d\n", b);
            printf("c: %d\n", c);
        }
    }

    printf("a: %d\n", a);
    printf("b: %d\n", b);
    printf("c: %d\n", c);
    return 0;
}
```

Quiz; Data scoping

```
int main (){
    int a =1, b =2, c =3;
    #pragma omp parallel num_threads(4) private(b) firstprivate(c)
    {
        #pragma omp critical
        {
            a++;  // -> shared
            b++;  // -> private
            c++;  // -> firstprivate
        }
        #pragma omp barrier
        if (omp_get_thread_num()==0){
            printf("a: %d\n", a);
            printf("b: %d\n", b);
            printf("c: %d\n", c);
        }
    }

    printf("a: %d\n", a);
    printf("b: %d\n", b);
    printf("c: %d\n", c);
    return 0;
}
```

```
./example


a: 5
```

Quiz; Data scoping

```
int main (){
    int a =1, b =2, c =3;
    #pragma omp parallel num_threads(4) private(b) firstprivate(c)
    {
        #pragma omp critical
        {
            a++;  // -> shared
            b++;  // -> private
            c++;  // -> firstprivate
        }
        #pragma omp barrier
        if (omp_get_thread_num()==0){
            printf("a: %d\n", a);
            printf("b: %d\n", b);
            printf("c: %d\n", c);
        }
    }

    printf("a: %d\n", a);
    printf("b: %d\n", b);
    printf("c: %d\n", c);
    return 0;
}
```

./example

```
a: 5
b: ?
```

Quiz; Data scoping

```
int main (){
    int a =1, b =2, c =3;
    #pragma omp parallel num_threads(4) private(b) firstprivate(c)
    {
        #pragma omp critical
        {
            a++;  // -> shared
            b++;  // -> private
            c++;  // -> firstprivate
        }
        #pragma omp barrier
        if (omp_get_thread_num()==0){
            printf("a: %d\n", a);
            printf("b: %d\n", b);
            printf("c: %d\n", c);
        }
    }

    printf("a: %d\n", a);
    printf("b: %d\n", b);
    printf("c: %d\n", c);
    return 0;
}
```

```
./example


a: 5
b: ?
c: 4
```

Quiz; Data scoping

```
int main (){
    int a =1, b =2, c =3;
    #pragma omp parallel num_threads(4) private(b) firstprivate(c)
    {
        #pragma omp critical
        {
            a++;  // -> shared
            b++;  // -> private
            c++;  // -> firstprivate
        }
        #pragma omp barrier
        if (omp_get_thread_num()==0){
            printf("a: %d\n", a);
            printf("b: %d\n", b);
            printf("c: %d\n", c);
        }
    }

    printf("a: %d\n", a);
    printf("b: %d\n", b);
    printf("c: %d\n", c);
    return 0;
}
```

```
./example


a: 5
b: ?
c: 4
a: 5
```

Quiz; Data scoping

```c
int main (){
    int a =1, b =2, c =3;
    #pragma omp parallel num_threads(4) private(b) firstprivate(c)
    {
        #pragma omp critical
        {
            a++;  // -> shared
            b++;  // -> private
            c++;  // -> firstprivate
        }
        #pragma omp barrier
        if (omp_get_thread_num()==0){
            printf("a: %d\n", a);
            printf("b: %d\n", b);
            printf("c: %d\n", c);
        }
    }

    printf("a: %d\n", a);
    printf("b: %d\n", b);
    printf("c: %d\n", c);
    return 0;
}
```

```
./example


a: 5
b: ?
c: 4
a: 5
b: 2
```

Quiz; Data scoping

```c
int main (){
    int a =1, b =2, c =3;
    #pragma omp parallel num_threads(4) private(b) firstprivate(c)
    {
        #pragma omp critical
        {
            a++;  // -> shared
            b++;  // -> private
            c++;  // -> firstprivate
        }
        #pragma omp barrier
        if (omp_get_thread_num()==0){
            printf("a: %d\n", a);
            printf("b: %d\n", b);
            printf("c: %d\n", c);
        }
    }

    printf("a: %d\n", a);
    printf("b: %d\n", b);
    printf("c: %d\n", c);
    return 0;
}
```

```
./example


a: 5
b: ?
c: 4
a: 5
b: 2
c: 3
```

Quiz; Task data scoping

```
int a=1;
void parallel_function()
{
    int b=2, c=3;
    #pragma omp parallel shared(b)
    #pragma omp parallel private(b)
    {
        int d=4;
        #pragma omp task
        {
            int e=5;
            a





        }
    }
}
```

Quiz; Task data scoping

```
int a=1;
void parallel_function()
{
    int b=2, c=3;
    #pragma omp parallel shared(b)
    #pragma omp parallel private(b)
    {
        int d=4;
        #pragma omp task
        {
            int e=5;
            a  // shared




        }
    }
}
```

Quiz; Task data scoping

```
int a=1;
void parallel_function()
{
    int b=2, c=3;
    #pragma omp parallel shared(b)
    #pragma omp parallel private(b)
    {
        int d=4;
        #pragma omp task
        {
            int e=5;
            a  // shared          -> a=1
            b




        }
    }
}
```

Quiz; Task data scoping

```
int a=1;
void parallel_function()
{
    int b=2, c=3;
    #pragma omp parallel shared(b)
    #pragma omp parallel private(b)
    {
        int d=4;
        #pragma omp task
        {
            int e=5;
            a  // shared         -> a=1
            b  // firstprivate



        }
    }
}
```

Quiz; Task data scoping

```
int a=1;
void parallel_function()
{
    int b=2, c=3;
    #pragma omp parallel shared(b)
    #pragma omp parallel private(b)
    {
        int d=4;
        #pragma omp task
        {
            int e=5;
            a  // shared         -> a=1
            b  // firstprivate   -> b=?
            c


        }
    }
}
```

Quiz; Task data scoping

```
int a=1;
void parallel_function()
{
    int b=2, c=3;
    #pragma omp parallel shared(b)
    #pragma omp parallel private(b)
    {
        int d=4;
        #pragma omp task
        {
            int e=5;
            a  // shared         -> a=1
            b  // firstprivate   -> b=?
            c  // shared


        }
    }
}
```

Quiz; Task data scoping

```
int a=1;
void parallel_function()
{
    int b=2, c=3;
    #pragma omp parallel shared(b)
    #pragma omp parallel private(b)
    {
        int d=4;
        #pragma omp task
        {
            int e=5;
            a  // shared         -> a=1
            b  // firstprivate   -> b=?
            c  // shared         -> c=3
            d

        }
    }
}
```

Quiz; Task data scoping

```
int a=1;
void parallel_function()
{
    int b=2, c=3;
    #pragma omp parallel shared(b)
    #pragma omp parallel private(b)
    {
        int d=4;
        #pragma omp task
        {
            int e=5;
            a  // shared        -> a=1
            b  // firstprivate  -> b=?
            c  // shared        -> c=3
            d  // firstprivate

        }
    }
}
```

Quiz; Task data scoping

```c
int a=1;
void parallel_function()
{
    int b=2, c=3;
    #pragma omp parallel shared(b)
    #pragma omp parallel private(b)
    {
        int d=4;
        #pragma omp task
        {
            int e=5;
            a  // shared          -> a=1
            b  // firstprivate    -> b=?
            c  // shared          -> c=3
            d  // firstprivate    -> d=4
            e
        }
    }
}
```

Quiz; Task data scoping

```
int a=1;
void parallel_function()
{
    int b=2, c=3;
    #pragma omp parallel shared(b)
    #pragma omp parallel private(b)
    {
        int d=4;
        #pragma omp task
        {
            int e=5;
            a  // shared         -> a=1
            b  // firstprivate   -> b=?
            c  // shared         -> c=3
            d  // firstprivate   -> d=4
            e  // private
        }
    }
}
```

Quiz; Task data scoping

```
int a=1;
void parallel_function()
{
    int b=2, c=3;
    #pragma omp parallel shared(b)
    #pragma omp parallel private(b)
    {
        int d=4;
        #pragma omp task
        {
            int e=5;
            a  // shared          -> a=1
            b  // firstprivate    -> b=?
            c  // shared          -> c=3
            d  // firstprivate    -> d=4
            e  // private         -> e=5
        }
    }
}
```

Quiz; Coarse-grained parallelization

```c
#define N 10000
#define ITER 100
double A[N + 2][N + 2];

int main(int argc, char **argv)
{

    for (int i = 0; i < N + 2; i++)      // Initialization
        for (int j = 0; j < N + 2; j++)
            A[i][j] = 0.0;

    for (int i = 0; i < N + 2; i++){     // Boundary conditions
        A[i][0] = 1.0; A[i][N + 2] = 1.0;
    }

    for (int n = 0; n < 100; n++){       // Main iteration loop

        for (int i = 1; i < N + 1; i++)
            for (int j = 1; j < N + 1; j++)
                A[i][j] = (A[i+1][j+1] + A[i-1][j-1] + A[i+1][j-1] + A[i-1][j+1])/4;
    }
    return 0;
}
```

Quiz; Coarse-grained parallelization

```
#define N 10000
#define ITER 100
double A[N + 2][N + 2];

int main(int argc, char **argv)
{

    for (int i = 0; i < N + 2; i++)      // Initialization
        for (int j = 0; j < N + 2; j++)
            A[i][j] = 0.0;

    for (int i = 0; i < N + 2; i++){     // Boundary conditions
        A[i][0] = 1.0; A[i][N + 2] = 1.0;
    }

    for (int n = 0; n < 100; n++){       // Main iteration loop
        #pragma omp parallel for         // Coarse-grained parallelization
        for (int i = 1; i < N + 1; i++)
            for (int j = 1; j < N + 1; j++)
                A[i][j] = (A[i+1][j+1] + A[i-1][j-1] + A[i+1][j-1] + A[i-1][j+1])/4;
    }
    return 0;
}
```

Quiz; Coarse-grained parallelization

```c
#define N 10000
#define ITER 100
double A[N + 2][N + 2];

int main(int argc, char **argv)
{
    #pragma omp parallel for             // First touch
    for (int i = 0; i < N + 2; i++)      // Initialization
        for (int j = 0; j < N + 2; j++)
            A[i][j] = 0.0;

    for (int i = 0; i < N + 2; i++){     // Boundary conditions
        A[i][0] = 1.0; A[i][N + 2] = 1.0;
    }

    for (int n = 0; n < 100; n++){       // Main iteration loop
        #pragma omp parallel for         // Coarse-grained parallelization
        for (int i = 1; i < N + 1; i++)
            for (int j = 1; j < N + 1; j++)
                A[i][j] = (A[i+1][j+1] + A[i-1][j-1] + A[i+1][j-1] + A[i-1][j+1])/4;
    }
    return 0;
}
```

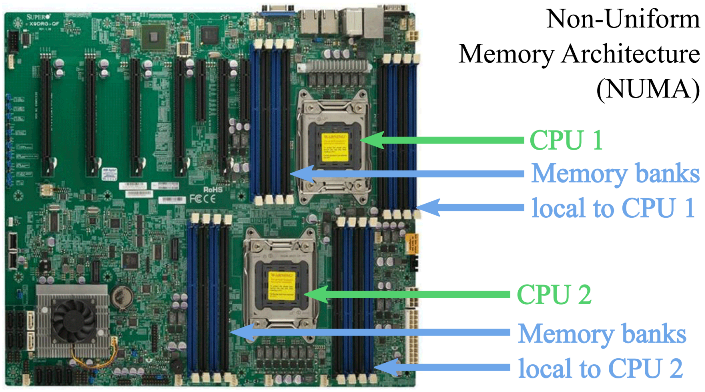
Quiz; Coarse-grained parallelization

```c
#define N 10000
#define ITER 100
double A[N + 2][N + 2];

int main(int argc, char **argv)
{
    #pragma omp parallel for             // First touch
    for (int i = 0; i < N + 2; i++)      // Initialization
        for (int j = 0; j < N + 2; j++)
            A[i][j] = 0.0;

    for (int i = 0; i < N + 2; i++){     // Boundary conditions
        A[i][0] = 1.0; A[i][N + 2] = 1.0;
    }

    for (int n = 0; n < 100; n++){       // Main iteration loop
        #pragma omp parallel for         // Coarse-grained parallelization
        for (int i = 1; i < N + 1; i++)
            for (int j = 1; j < N + 1; j++)
                A[i][j] = (A[i+1][j+1] + A[i-1][j-1] + A[i+1][j-1] + A[i-1][j+1])/4;
    }
    return 0;
}
```



Non-Uniform
Memory Architecture
(NUMA)

CPU 1

Memory banks
local to CPU 1

CPU 2

Memory banks
local to CPU 2

Figure: Example of a NUMA architecture: server board of our SXP8600 workstation based on a two-way (i.e., dual-socket) Intel Xeon processor.
From: Vladimirov, A., Asai, R., & Karpusenko, V. (2019). Parallel programming and optimization with Intel Xeon Phi coprocessors: Handbook on the development and optimization of parallel applications for Intel Xeon processors and Intel Xeon Phi coprocessors. Retrieved June 4, 2019, from https://colfaxresearch.com/second-edition-of-parallel-programming-and-optimization-with-intel-xeon-phi-coprocessors/

Typical patterns that come up in parallel programming

- Loop parallelization (Worksharing)
  - Parallelize the for loops that are time consuming in the code
  - Make sure the loops are parallelizable (dependency analysis)
  - Put the pragmas and take care of the data attributes

- Example:

```
// Initialization ...

for (int n = 0; n < 100; n++){
    #pragma omp parallel for
    for (int i = 1; i < N + 1; i++)
        for (int j = 1; j < N + 1; j++)
            A[i][j] = (A[i+1][j+1] + A[i-1][j-1] + A[i+1][j-1] + A[i-1][j+1])/4;
}
```

Typical patterns that come up in parallel programming (Cont.)

- Divide and conquer and unstructured parallelism (Tasking)
  - Split the problem into subproblems
  - Solve the subproblems in parallel
  - Fits the Tasking in OpenMP (v3 and later)

- Example:

```
struct node
{
    struct node* left;
    struct node* right;
};

void traverse( struct node*p ) {
    if(p->left)
        #pragma omp task
        traverse(p->left);
    if(p->right)
        #pragma omp task
        traverse(p->right);
    process(p);
}
```

```
// main

#pragma omp parallel
{
    #pragma omp single
    traverse(root);
}
```

Assignment 5 - Laplace 2D

Assignment 5 - Laplace 2D

- 2d Laplace equation with fixed boundaries
- Problem domain is unit square with uniform mesh
- Finite differences are used for the discretization
- We use Jacobi iterative method to solve the equation
- Look into the code and find the bottlenecks
- Use OpenMP to parallelize the solver
- You need to get a speedup of 16 on our server with 32 logical cores
- The server has 2 NUMA nodes each with 8 cores
- Pay attention to data locality on the cores

Assignment 5 - Laplace 2D - Provided Files

- Makefile
  - contains rules to build executables
  - available targets: parallel, sequential, unit_test, all (default), clean
  - 'mode=debug make [target]' to build debug version, use 'make clean' before
- main.c
  - main function - argument handling + call initialization of arrays and main iteration loop
- laplace.h
  - Header and definitions for the arrays
- laplace_seq.h
  - Sequential version of `time_step()`.
- student/laplace_par.h
  - Implement the parallel version in this file
- unit_test.c
  - The unit tests that execute both the serial and parallel version to compare results.