

Parallel Programming Tutorial - OpenMP Wrap-Up

Amir Raoofy, M.Sc.

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

Technical University of Munich

16. Mai 2018



TUM Uhrenturm

Few organizational notes

Organization

- Deadline for assignment 5 is extended; May 22nd
- We have an optional assignment on SIMD (will be released next week)
 - Based on guest lecture by Dr. Michael Klemm
- On June 25th we will have a lecture on Optimization of sequential programs by M.Sc. Alexis Engelke
 - Topics: (tentative)
 - Compiler optimizations
 - Floating-point optimizations
 - (Auto-)Vectorization
 - Cache optimizations
 - Eventually profiling

OpenMP Wrap-Up

Nested parallel regions revisited

```

1  #include <iostream>
2  #include<omp.h>
3
4  int main(){
5
6      int num_threads=4;
7      omp_set_num_threads(num_threads);
8
9      #pragma omp parallel
10     {
11         #pragma omp parallel for
12         for (int i = 0; i < num_threads; i++)
13         {
14             #pragma omp critical
15             std::cout << "My id is: "
16                     << omp_get_thread_num() << std::endl;
17         }
18     }
19 }
```

Nested parallel regions revisited

```

1  #include <iostream>
2  #include <omp.h>
3
4  int main(){
5
6      int num_threads=4;
7      omp_set_num_threads(num_threads);
8
9      #pragma omp parallel
10     {
11         #pragma omp parallel for
12         for (int i = 0; i < num_threads; i++)
13         {
14             #pragma omp critical
15             std::cout << "My id is: "
16                     << omp_get_thread_num() << std::endl;
17         }
18     }
19 }
```

./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

Nested parallel regions revisited (Cont.)

```

1  #include <iostream>
2  #include<omp.h>
3
4  int main(){
5
6      int num_threads=4;
7      omp_set_num_threads(num_threads);
8      omp_set_nested(1);
9
10     #pragma omp parallel
11     {
12         #pragma omp parallel for
13         for (int i = 0; i < num_threads; i++)
14         {
15             #pragma omp critical
16             std::cout << "My id is: "
17                     << omp_get_thread_num() << std::endl;
18         }
19     }
20 }
```

Nested parallel regions revisited (Cont.)

```

1  #include <iostream>
2  #include<omp.h>
3
4  int main(){
5
6      int num_threads=4;
7      omp_set_num_threads(num_threads);
8      omp_set_nested(1);
9
10     #pragma omp parallel
11     {
12         #pragma omp parallel for
13         for (int i = 0; i < num_threads; i++)
14         {
15             #pragma omp critical
16             std::cout << "My id is: "
17                     << omp_get_thread_num() << std::endl;
18         }
19     }
20 }
```

./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?

```

1  #include <iostream>
2  #include <omp.h>
3
4  int main(){
5
6      int id;
7      #pragma omp parallel num_threads(4)
8      {
9          id = omp_get_thread_num();
10         #pragma omp critical
11         std::cout << "My id is: " << id << std::endl;
12     }
13
14 }
```

Quiz; What is the problem with this program?

```

1  #include <iostream>
2  #include <omp.h>
3
4  int main(){
5
6      int id;
7      #pragma omp parallel num_threads(4)
8      {
9          id = omp_get_thread_num();
10         #pragma omp critical
11         std::cout << "My id is: " << id << std::endl;
12     }
13
14 }
```

./example

Quiz; What is the problem with this program?

```

1  #include <iostream>
2  #include <omp.h>
3
4  int main(){
5
6      int id;
7      #pragma omp parallel num_threads(4)
8      {
9          id = omp_get_thread_num();
10         #pragma omp critical
11         std::cout << "My id is: " << id << std::endl;
12     }
13
14 }
```

./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?

```

1  #include <iostream>
2  #include <omp.h>
3
4  int main(){
5
6      int id;
7      #pragma omp parallel num_threads(4)
8      {
9          id = omp_get_thread_num();
10         #pragma omp critical
11         std::cout << "My id is: " << id << std::endl;
12     }
13
14 }
```

./example

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

./example

Quiz; What is the problem with this program?

```

1  #include <iostream>
2  #include <omp.h>
3
4  int main(){
5
6      int id;
7      #pragma omp parallel num_threads(4)
8      {
9          id = omp_get_thread_num();
10         #pragma omp critical
11         std::cout << "My id is: " << id << std::endl;
12     }
13
14 }
```

./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.)

```
1  #include <iostream>
2  #include <omp.h>
3
4  int main(){
5
6      int id;
7      #pragma omp parallel num_threads(4) private(id)
8      {
9          id = omp_get_thread_num();
10         #pragma omp critical
11         std::cout << "My id is: " << id << std::endl;
12     }
13
14 }
```

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

./example

```
1  #include <iostream>
2  #include <omp.h>
3
4  int main(){
5
6      int id;
7      #pragma omp parallel num_threads(4) private(id)
8      {
9          id = omp_get_thread_num();
10         #pragma omp critical
11         std::cout << "My id is: " << id << std::endl;
12     }
13
14 }
```

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

```
1  #include <iostream>
2  #include <omp.h>
3
4  int main(){
5
6      int id;
7      #pragma omp parallel num_threads(4) private(id)
8      {
9          id = omp_get_thread_num();
10         #pragma omp critical
11         std::cout << "My id is: " << id << std::endl;
12     }
13
14 }
```

./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

```
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

```
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

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

```
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

```
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

```

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

```

1  #define N 10000
2  #define ITER 100
3  double A[N + 2][N + 2];
4
5  int main(int argc, char **argv)
6  {
7
8      for (int i = 0; i < N + 2; i++)                // Initialization
9          for (int j = 0; j < N + 2; j++)
10             A[i][j] = 0.0;
11
12     for (int i = 0; i < N + 2; i++){                // Boundary conditions
13         A[i][0] = 1.0; A[i][N + 2] = 1.0;
14     }
15
16     for (int n = 0; n < 100; n++){                // Main iteration loop
17
18         for (int i = 1; i < N + 1; i++)
19             for (int j = 1; j < N + 1; j++)
20                 A[i][j] = (A[i+1][j+1] + A[i-1][j-1] + A[i+1][j-1] + A[i-1][j+1])/4;
21     }
22     return 0;
23 }

```

Quiz; Coarse-grained parallelization

```
1 #define N 10000
2 #define ITER 100
3 double A[N + 2][N + 2];
4
5 int main(int argc, char **argv)
6 {
7
8     for (int i = 0; i < N + 2; i++)           // Initialization
9         for (int j = 0; j < N + 2; j++)
10             A[i][j] = 0.0;
11
12     for (int i = 0; i < N + 2; i++){           // Boundary conditions
13         A[i][0] = 1.0; A[i][N + 2] = 1.0;
14     }
15
16     for (int n = 0; n < 100; n++){             // Main iteration loop
17         #pragma omp parallel for               // Coarse-grained parallelization
18         for (int i = 1; i < N + 1; i++)
19             for (int j = 1; j < N + 1; j++)
20                 A[i][j] = (A[i+1][j+1] + A[i-1][j-1] + A[i+1][j-1] + A[i-1][j+1])/4;
21     }
22     return 0;
23 }
```


Quiz; Coarse-grained parallelization

```

1  #define N 10000
2  #define ITER 100
3  double A[N + 2][N + 2];
4
5  int main(int argc, char **argv)
6  {
7      #pragma omp parallel for                // First touch
8      for (int i = 0; i < N + 2; i++)          // Initialization
9          for (int j = 0; j < N + 2; j++)
10             A[i][j] = 0.0;
11
12     for (int i = 0; i < N + 2; i++){          // Boundary conditions
13         A[i][0] = 1.0; A[i][N + 2] = 1.0;
14     }
15
16     for (int n = 0; n < 100; n++){           // Main iteration loop
17         #pragma omp parallel for            // Coarse-grained parallelization
18         for (int i = 1; i < N + 1; i++)
19             for (int j = 1; j < N + 1; j++)
20                 A[i][j] = (A[i+1][j+1] + A[i-1][j-1] + A[i+1][j-1] + A[i-1][j+1])/4;
21     }
22     return 0;
23 }

```

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:

```
1 // Initialization ...
2
3 for (int n = 0; n < 100; n++){
4     #pragma omp parallel for
5     for (int i = 1; i < N + 1; i++)
6         for (int j = 1; j < N + 1; j++)
7             A[i][j] = (A[i+1][j+1] + A[i-1][j-1] + A[i+1][j-1] + A[i-1][j+1])/4;
8 }
```

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:

```
1 struct node
2 {
3     struct node* left;
4     struct node* right;
5 };
6
7 void traverse( struct node*p ) {
8     if(p->left)
9         #pragma omp task
10        traverse(p->left);
11    if(p->right)
12        #pragma omp task
13        traverse(p->right);
14    process(p);
15 }
```

```
1 // main
2
3 #pragma omp parallel
4 {
5     #pragma omp single
6     traverse(root);
7 }
```

Solution for Assignment 3

Solution for Assignment 3

```

1  template <typename SrcView, typename DstView>
2  void x_gradient(const SrcView &src, const DstView &dst, int num_threads)
3  {
4      int start = 0;
5      int chunk_size = 16;
6      std::mutex mtx;
7
8      std::vector<std::thread> threads;
9
10     for (int i = 0; i < num_threads; ++i)
11     {
12         threads.push_back(std::thread(x_gradient_kernel<SrcView, DstView>, \
13                                     std::ref(src), std::ref(dst), \
14                                     std::ref(start), std::ref(chunk_size), std::ref(mtx)));
15     }
16
17     for (auto &th : threads)
18         th.join();
19 }

```

Solution for Assignment 3 (Cont.)

```

1  template <typename SrcView, typename DstView>
2  void *x_gradient_kernel(const SrcView &src, const DstView &dst,
3                          int &start, int &chunk_size, std::mutex &mtx)
4  {
5      typedef typename channel_type<DstView>::type dst_channel_t;
6      int local_start;
7
8      while (true)
9      {
10         // next slide
11     }
12 }

```

Solution for Assignment 3 (Cont.)

```
1  while (true)
2  {
3      mtx.lock();
4      local_start = start;
5      if (src.height() - start < 1) {
6          mtx.unlock(); break;
7      }
8
9      if (src.height() - local_start < chunk_size)
10         chunk_size = src.height() - local_start;
11
12     start += chunk_size;
13     mtx.unlock();
14
15     for (int y = local_start; y < local_start + chunk_size; ++y) {
16         typename SrcView::x_iterator src_it = src.row_begin(y);
17         typename DstView::x_iterator dst_it = dst.row_begin(y);
18
19         for (int x = 1; x < src.width() - 1; ++x)
20             static_transform(src_it[x - 1], src_it[x + 1], dst_it[x],
21                             halfdiff_cast_channels<dst_channel_t>());
22     }
23 }
```

Solution for Assignment 4

Solution for Assignment 4

```

1  template <typename SrcView, typename DstView>
2  void x_gradient(const SrcView& src, const DstView& dst, int num_threads) {
3      typedef typename channel_type<DstView>::type dst_channel_t;
4
5      omp_set_num_threads(num_threads);
6
7      #pragma omp parallel for schedule (dynamic,5)
8      for (int y=0; y<src.height(); ++y) {
9          typename SrcView::x_iterator src_it = src.row_begin(y);
10         typename DstView::x_iterator dst_it = dst.row_begin(y);
11
12         for (int x=1; x<src.width()-1; ++x) {
13             static_transform(src_it[x-1], src_it[x+1], dst_it[x],
14                             halfdiff_cast_channels<dst_channel_t>());
15         }
16     }
17 }

```

Hints for Assignment 5

Hints for Assignment 5

- If you use sections:
 - Pay attention to the recursive structure of `traverse`
 - Make sure that nesting is enabled
 - Stop nesting at a specific level to prevent creating so many parallel regions
- If you use tasks:
 - Again, pay attention to the recursive structure of `traverse`
 - Make sure that you create appropriate number of tasks
 - Try to restrict the number of created tasks using the appropriate clauses

Assignment 6 - Laplace 2D

Assignment 6 - 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 6 - 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.