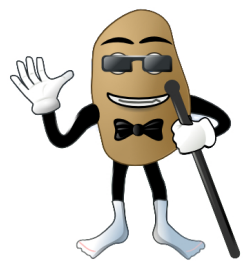
Część 3
OpenMP

Maciej J. Mrowiński

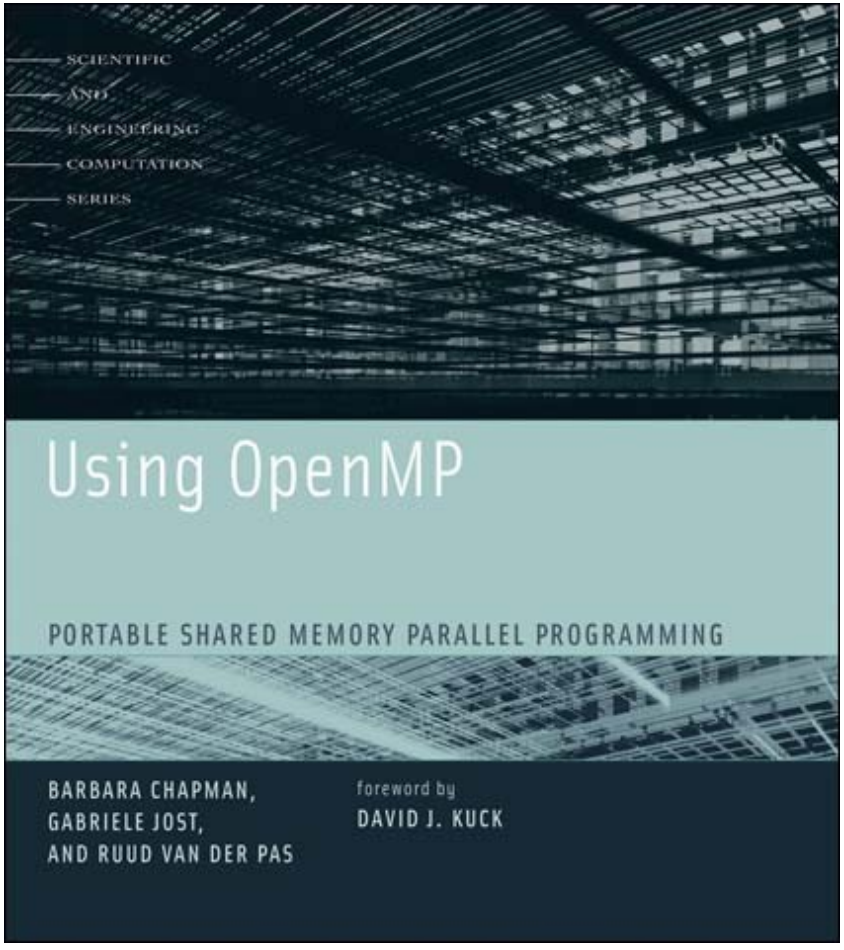
mrow@if.pw.edu.pl

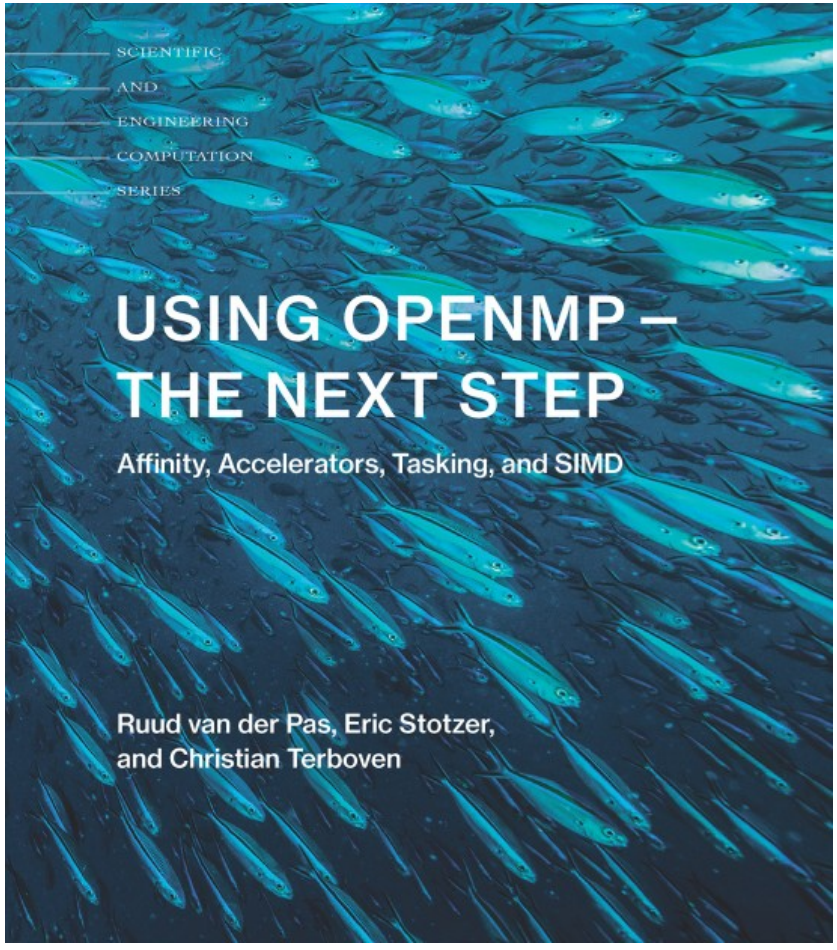
Wydział Fizyki
Politechnika Warszawska

21 grudnia 2018



Dzisiejszy Odcinek Sponsoruje...





Wprowadzenie

Hello World - #pragma omp parallel

```
1   #include<stdio.h>
2   #include<stdlib.h>
3   #include<omp.h>
4
5   int main(int argc, char *argv[])
6   {
7      #pragma omp parallel
8      {
9         printf("HelloOpenMP %d\n", omp_get_thread_num());
10     }
11
12     return 0;
13  }
```

Hello World - #pragma omp parallel

```
HelloOpenMP 1
HelloOpenMP 0
HelloOpenMP 2
HelloOpenMP 6
HelloOpenMP 5
HelloOpenMP 4
HelloOpenMP 3
HelloOpenMP 7
```

Współdzielenie pracy

Współdzielenie pracy - przykład prostacki

```
1  #pragma omp parallel
2  {
3     if(omp_get_thread_num() == 2)
4        printf("HelloOpenMP %d\n", omp_get_thread_num());
5     if(omp_get_thread_num() == 4)
6        printf("HelloOpenMP %d\n", omp_get_thread_num());
7  }
```

Współdzielenie pracy - przykład prostacki

```
HelloOpenMP 2
HelloOpenMP 4
```

Współdzielenie pracy - sections

```
1   #pragma omp parallel
2   {
3      #pragma omp sections
4      {
5         #pragma omp section
6            printf("Section 1 - %d\n", omp_get_thread_num());
7         #pragma omp section
8            printf("Section 2 - %d\n", omp_get_thread_num());
9      }
10  }
```

Współdzielenie pracy - sections

```
Section 1 - 1
Section 2 - 2
```

Współdzielenie pracy - sections

```
1  #pragma omp parallel sections
2  {
3     #pragma omp section
4        printf("Section 1 - %d\n", omp_get_thread_num());
5     #pragma omp section
6        printf("Section 2 - %d\n", omp_get_thread_num());
7  }
```

Współdzielenie pracy - single

```
1  #pragma omp parallel
2  {
3     #pragma omp single
4        printf("Single - %d\n", omp_get_thread_num());
5
6     printf("Parallel - %d\n", omp_get_thread_num());
7  }
```

Współdzielenie pracy - single

```
Single - 1
Parallel - 3
Parallel - 4
Parallel - 6
Parallel - 5
Parallel - 2
Parallel - 1
Parallel - 0
Parallel - 7
```

Współdzielenie pracy - for

```
1  int i;
2  int n = 10;
3
4  #pragma omp parallel private(i) shared(n)
5  {
6     #pragma omp for
7     for(i = 0; i < n; ++i)
8        printf("HelloOpenMP %d - %d\n", i, omp_get_thread_num());
9  }
```

Współdzielenie pracy - for

```
HelloOpenMP 2 - 1
HelloOpenMP 7 - 5
HelloOpenMP 3 - 1
HelloOpenMP 8 - 6
HelloOpenMP 0 - 0
HelloOpenMP 4 - 2
HelloOpenMP 1 - 0
HelloOpenMP 5 - 3
HelloOpenMP 6 - 4
HelloOpenMP 9 - 7
```

Współdzielenie pracy - for

```
1  int i;
2  int n = 10;
3
4  #pragma omp parallel
5  {
6     #pragma omp for
7     for(i = 0; i < n; ++i)
8        printf("HelloOpenMP %d - %d\n", i, omp_get_thread_num());
9  }
```

Współdzielenie pracy - for

```
1  int n = 10;
2
3  #pragma omp parallel
4  {
5     #pragma omp for
6     for(int i = 0; i < n; ++i)
7        printf("HelloOpenMP %d - %d\n", i, omp_get_thread_num());
8  }
```

Współdzielenie pracy - for

```
1   int n = 10;
2
3   #pragma omp parallel for
4      for(i = 0; i < n; ++i)
5         printf("HelloOpenMP %d - %d\n", i, omp_get_thread_num());
```

Inne ważne #pragma

Inne ważne pragma - barrier

```
1   void functionA()
2   {
3      int sum = 0;
4      for(int i = 0; i < 1000000; ++i)
5         ++sum;
6
7      printf("%d: function A done!\n", omp_get_thread_num());
8   }
9
10  void functionB()
11  {
12     int sum = 0;
13     for(int i = 0; i < 1000000; ++i)
14        ++sum;
15
16     printf("%d: function B done!\n", omp_get_thread_num());
17  }
18
19  int main(int argc, char *argv[])
20  {
21     #pragma omp parallel
22     {
23        functionA();
24
25        functionB();
26     }
27     return 0;
28  }
```

Inne ważne pragma - barrier

```
4: function A done!
0: function A done!
1: function A done!
3: function A done!
2: function A done!
5: function A done!
6: function A done!
4: function B done!
0: function B done!
1: function B done!
3: function B done!
7: function A done!
2: function B done!
5: function B done!
6: function B done!
7: function B done!
```

Inne ważne pragma - barrier

```
1   void functionA()
2   {
3      int sum = 0;
4      for(int i = 0; i < 1000000; ++i)
5         ++sum;
6
7      printf("%d: function A done!\n", omp_get_thread_num());
8   }
9
10  void functionB()
11  {
12     int sum = 0;
13     for(int i = 0; i < 1000000; ++i)
14        ++sum;
15
16     printf("%d: function B done!\n", omp_get_thread_num());
17  }
18
19  int main(int argc, char *argv[])
20  {
21     #pragma omp parallel
22     {
23        functionA();
24        #pragma omp barrier
25        functionB();
26     }
27     return 0;
28  }
```

Inne ważne pragma - barrier

```
1: function A done!
6: function A done!
5: function A done!
4: function A done!
7: function A done!
3: function A done!
2: function A done!
0: function A done!
7: function B done!
1: function B done!
2: function B done!
6: function B done!
0: function B done!
3: function B done!
4: function B done!
5: function B done!
```

Inne ważne pragma - atomic

```
1  int counter = 0;
2  int n = 1000000;
3  #pragma omp parallel for
4  for(int i = 0; i < n; ++i)
5  {
6     ++counter;
7  }
8
9  printf("Counter: %d\n", counter);
```

Inne ważne pragma - atomic

```
Counter: 141559
```

Inne ważne pragma - atomic

```
1   int counter = 0;
2   int n = 1000000;
3   #pragma omp parallel for
4   for(int i = 0; i < n; ++i)
5   {
6      #pragma omp atomic
7      ++counter;
8   }
9
10  printf("Counter: %d\n", counter);
```

Inne ważne pragma - atomic

```
Counter: 1000000
```

Inne ważne pragma - critical

```
1   int sum = 0;
2   int n = 1000000;
3
4   int* array = malloc(n * sizeof(int));
5   for(int i = 0; i < n; ++i)
6      array[i] = 1;
7
8
9
10  #pragma omp parallel
11  {
12     int localSum = 0;
13
14     #pragma omp for
15     for(int i = 0; i < n; ++i)
16     {
17        localSum += array[i];
18     }
19
20     sum += localSum;
21  }
22
23  printf("Sum: %d\n", sum);
24
25  free(array);
```

Inne ważne pragma - critical

```
Sum: 875000
```

Inne ważne pragma - critical

```
1   int sum = 0;
2   int n = 1000000;
3
4   int* array = malloc(n * sizeof(int));
5   for(int i = 0; i < n; ++i)
6      array[i] = 1;
7
8
9
10  #pragma omp parallel
11  {
12     int localSum = 0;
13
14     #pragma omp for
15     for(int i = 0; i < n; ++i)
16     {
17        localSum += array[i];
18     }
19
20     #pragma omp critical (update_sum)
21     sum += localSum;
22  }
23
24  printf("Sum: %d\n", sum);
25
26  free(array);
```

Inne ważne pragma - critical

```
Sum: 1000000
```

Clauses

Clauses - parallel

- if(scalar-expression)
- num_threads(integer-expression)
- private(list)
- firstprivate(list)
- shared(list)
- default(none|shared)
- copyin(list)
- reduction(operator:list)

Clauses - for

- private(list)
- firstprivate(list)
- lastprivate(list)
- reduction(operator:list)
- ordered
- schedule(kind[,chunk size)]
- nowait

Clauses - sections

- private(list)
- firstprivate(list)
- lastprivate(list)
- reduction(operator:list)
- nowait

Clauses - num_threads

```
1   int n = 10;
2
3   #pragma omp parallel for num_threads(2)
4      for(i = 0; i < n; ++i)
5         printf("HelloOpenMP %d - %d\n", i, omp_get_thread_num());
```

Clauses - num_threads

```
HelloOpenMP 0 - 0
HelloOpenMP 5 - 1
HelloOpenMP 6 - 1
HelloOpenMP 7 - 1
HelloOpenMP 8 - 1
HelloOpenMP 9 - 1
HelloOpenMP 1 - 0
HelloOpenMP 2 - 0
HelloOpenMP 3 - 0
HelloOpenMP 4 - 0
```

Clauses - if

```
1   int n = 4;
2
3   #pragma omp parallel for if(n > 5)
4   for(int i = 0; i < n; ++i)
5      printf("HelloOpenMP %d - %d\n", i, omp_get_thread_num());
```

Clauses - if

```
HelloOpenMP 0 - 0
HelloOpenMP 1 - 0
HelloOpenMP 2 - 0
HelloOpenMP 3 - 0
```

Clauses - firstprivate

```
1  int n = 10;
2  #pragma omp parallel private(n)
3  {
4     printf("HelloOpenMP %d\n", n);
5  }
```

Clauses - firstprivate

```
HelloOpenMP 0
HelloOpenMP 4199585
HelloOpenMP 0
HelloOpenMP 0
HelloOpenMP 0
HelloOpenMP 0
HelloOpenMP 0
HelloOpenMP 0
```

Clauses - firstprivate

```
1  int n = 10;
2  #pragma omp parallel firstprivate(n)
3  {
4     printf("HelloOpenMP %d\n", n);
5  }
```

Clauses - firstprivate

```
HelloOpenMP 10
HelloOpenMP 10
HelloOpenMP 10
HelloOpenMP 10
HelloOpenMP 10
HelloOpenMP 10
HelloOpenMP 10
HelloOpenMP 10
```

Clauses - lastprivate

```
1  int i;
2  int n = 10;
3  #pragma omp parallel for private(i) shared(n)
4     for(i = 0; i < n; ++i)
5        printf("HelloOpenMP %d - %d\n", i, omp_get_thread_num());
6
7  printf("After loop: %d\n", i);
```

Clauses - lastprivate

```
HelloOpenMP 2 - 1
HelloOpenMP 7 - 5
HelloOpenMP 3 - 1
HelloOpenMP 4 - 2
HelloOpenMP 0 - 0
HelloOpenMP 6 - 4
HelloOpenMP 9 - 7
HelloOpenMP 5 - 3
HelloOpenMP 8 - 6
HelloOpenMP 1 - 0
After loop: 2147344384
```

Clauses - lastprivate

```
1  int i;
2  int n = 10;
3  #pragma omp parallel for lastprivate(i) shared(n)
4     for(i = 0; i < n; ++i)
5        printf("HelloOpenMP %d - %d\n", i, omp_get_thread_num());
6
7  printf("After loop: %d\n", i);
```

Clauses - lastprivate

```
HelloOpenMP 2 - 1
HelloOpenMP 0 - 0
HelloOpenMP 3 - 1
HelloOpenMP 1 - 0
HelloOpenMP 8 - 6
HelloOpenMP 5 - 3
HelloOpenMP 7 - 5
HelloOpenMP 6 - 4
HelloOpenMP 4 - 2
HelloOpenMP 9 - 7
After loop: 10
```

Clauses - reduction

```
1   int sum = 0;
2   int n = 1000000;
3
4   int* array = malloc(n * sizeof(int));
5   for(int i = 0; i < n; ++i)
6      array[i] = 1;
7
8   #pragma omp parallel for reduction(+:sum)
9   for(int i = 0; i < n; ++i)
10     sum += array[i];
11
12  printf("Sum: %d\n", sum);
13
14  free(array);
```

Clauses - reduction

```
Sum: 1000000
```

Clauses - schedule

```
1   int sum = 0;
2   int n = 10;
3
4   int* array = malloc(n * sizeof(int));
5   for(int i = 0; i < n; ++i)
6      array[i] = 1;
7
8   #pragma omp parallel for num_threads(3) reduction(+:sum) schedule(static, 2)
9   for(int i = 0; i < n; ++i)
10  {
11     printf("%d - %d\n", omp_get_thread_num(), i);
12     sum += array[i];
13  }
14
15  printf("Sum: %d\n", sum);
16
17  free(array);
```

Clauses - schedule

```
1 - 2
1 - 3
1 - 8
1 - 9
0 - 0
0 - 1
0 - 6
0 - 7
2 - 4
2 - 5
Sum: 10
```

Clauses - schedule

```
1   int sum = 0;
2   int n = 10;
3
4   int* array = malloc(n * sizeof(int));
5   for(int i = 0; i < n; ++i)
6      array[i] = 1;
7
8   #pragma omp parallel for num_threads(3) reduction(+:sum) schedule(dynamic, 2)
9   for(int i = 0; i < n; ++i)
10  {
11     printf("%d - %d\n", omp_get_thread_num(), i);
12     sum += array[i];
13  }
14
15  printf("Sum: %d\n", sum);
16
17  free(array);
```

Clauses - schedule

```
1 - 0
1 - 1
2 - 2
2 - 3
0 - 4
0 - 5
1 - 6
1 - 7
0 - 8
0 - 9
Sum: 10
```

Locks

```
1   int sum = 0;
2   int n = 1000000;
3
4   int* array = malloc(n * sizeof(int));
5   for(int i = 0; i < n; ++i)
6      array[i] = 1;
7
8   #pragma omp parallel
9   {
10     int localSum = 0;
11
12     #pragma omp for
13     for(int i = 0; i < n; ++i)
14     {
15        localSum += array[i];
16     }
17
18     sum += localSum;
19  }
20
21  printf("Sum: %d\n", sum);
22
23  free(array);
```

Clauses - schedule

```
Sum: 875000
```

Locks

- void omp_init_lock (omp_lock_t *)
- void omp_init_lock_with_hint (omp_lock_t *, omp_lock_hint_t)
- void omp_destroy_lock (omp_lock_t *)
- void omp_set_lock (omp_lock_t *)
- void omp_unset_lock (omp_lock_t *);
- int omp_test_lock (omp_lock_t *)
- void omp_init_nest_lock (omp_nest_lock_t)
- ...

Locks - hints

- omp_lock_hint_none = 0,
- omp_lock_hint_uncontended = 1
- omp_lock_hint_contended = 2
- omp_lock_hint_nonspeculative = 4
- omp_lock_hint_speculative = 8
- ...

Locks

```
1   int sum = 0;
2   int n = 1000000;
3
4   int* array = malloc(n * sizeof(int));
5   for(int i = 0; i < n; ++i)
6      array[i] = 1;
7
8   omp_lock_t lock;
9   omp_init_lock(&lock);
10
11  #pragma omp parallel
12  {
13     int localSum = 0;
14
15     #pragma omp for
16     for(int i = 0; i < n; ++i)
17     {
18        localSum += array[i];
19     }
20
21     omp_set_lock(&lock);
22     sum += localSum;
23     omp_unset_lock(&lock);
24  }
25
26  omp_destroy_lock(&lock);
27
28  printf("Sum: %d\n", sum);
29
30  free(array);
```

Clauses - schedule

```
Sum: 1000000
```

Tasks

```
1   #pragma omp parallel
2   {
3      #pragma omp single
4      {
5         #pragma omp task
6            printf("Jan ");
7         #pragma omp task
8            printf("Kowalski ");
9      }
10  }
```

Tasks

Jan Kowalski

Tasks

```
1   #pragma omp parallel
2   {
3      #pragma omp single
4      {
5         printf("Ten ");
6         #pragma omp task
7            printf("Jan ");
8         #pragma omp task
9            printf("Kowalski ");
10     }
11  }
```

Tasks

Ten Jan Kowalski

Tasks

```
1   #pragma omp parallel
2   {
3      #pragma omp single
4      {
5         printf("Ten ");
6         #pragma omp task
7            printf("Jan ");
8         #pragma omp task
9            printf("Kowalski ");
10        printf("to kawal drania! ");
11     }
12  }
```

Tasks

Ten to kawal drania! Jan Kowalski

Tasks

```
1   #pragma omp parallel
2   {
3      #pragma omp single
4      {
5         printf("Ten ");
6         #pragma omp task
7            printf("Jan ");
8         #pragma omp task
9            printf("Kowalski ");
10
11        #pragma omp taskwait
12        printf("to kawal drania! ");
13     }
14  }
```

Tasks

```
Ten Kowalski Jan to kawal drania!
```

Tasks

```
1   #include<stdio.h>
2   #include<stdlib.h>
3   #include<omp.h>
4
5   struct LinkedListNode
6   {
7      int data;
8      struct LinkedListNode* next;
9   };
10
11  typedef struct LinkedListNode Node;
12
13  int work(int data)
14  {
15     return data * data * data * data;
16  }
```

Tasks

```
18  int main(int argc, char *argv[])
19  {
20     int ntasks = 15;
21
22     Node* head = malloc(sizeof(Node));
23     Node* prev = head;
24     for(int i = 0; i < ntasks; ++i)
25     {
26        Node* next = malloc(sizeof(Node));
27        next->data = i;
28        next->next = NULL;
29
30        prev->next = next;
31        prev = next;
32     }
33
34     Node* next = head->next;
35     while(next != NULL)
36     {
37        printf("Thread %d: %d - %d\n", omp_get_thread_num(), next->data,
                work(next->data));
38        next = next->next;
39     }
40     printf("Finished!\n");
41
42     Node* current = head;
43     while(current != NULL)
44     {
45        next = current->next;
46        free(current);
47        current = next;
48     }
49     return 0;
```

Tasks

```
Thread 0: 0 - 0
Thread 0: 1 - 1
Thread 0: 2 - 16
Thread 0: 3 - 81
Thread 0: 4 - 256
Thread 0: 5 - 625
Thread 0: 6 - 1296
Thread 0: 7 - 2401
Thread 0: 8 - 4096
Thread 0: 9 - 6561
Thread 0: 10 - 10000
Thread 0: 11 - 14641
Thread 0: 12 - 20736
Thread 0: 13 - 28561
Thread 0: 14 - 38416
Finished!
```

Tasks

```
34      #pragma omp parallel
35      {
36         #pragma omp single nowait
37         {
38            Node* next = head->next;
39            while(next != NULL)
40            {
41               #pragma omp task firstprivate(next)
42               {
43                  printf("Thread %d: %d - %d\n", omp_get_thread_num(), next->data,
                          work(next->data));
44               }
45               next = next->next;
46            }
47         }
48      }
```

Tasks

```
Thread 2: 0 - 0
Thread 2: 1 - 1
Thread 2: 2 - 16
Thread 6: 3 - 81
Thread 0: 4 - 256
Thread 5: 5 - 625
Thread 3: 6 - 1296
Thread 3: 7 - 2401
Thread 4: 8 - 4096
Thread 4: 9 - 6561
Thread 6: 10 - 10000
Thread 5: 11 - 14641
Thread 5: 12 - 20736
Thread 7: 13 - 28561
Thread 5: 14 - 38416
Finished!
```

Tasks

```
1   void swap(int* a, int i, int j)
2   {
3      int tmp = a[i];
4      a[i] = a[j];
5      a[j] = tmp;
6   }
7
8   int partition(int* a, int lo, int hi) {
9      int j = lo - 1;
10     int p = a[hi - 1];
11     for(int i = lo; i < hi - 1; ++i)
12        if(a[i] <= p)
13           swap(a, ++j, i);
14
15     swap(a, ++j, hi - 1);
16     return j;
17  }
18
19  int quicksrt(int* a, int lo, int hi) {
20     if(lo < hi)
21     {
22        int p = partition(a, lo, hi);
23        quicksrt(a, lo, p);
24        quicksrt(a, p + 1, hi);
25     }
26  }
```

Tasks

```
27  int main(int argc, char *argv[])
28  {
29      int array[] = {5, 1, 6, 4, 7, 2, 3, 6, 7, 9, 0, 1, 4, 2, 9, 3, 2};
30      quicksrt(array, 0, 16);
31      for(int i = 0; i < 16; ++i)
32          printf("%d", array[i]);
33
34      return 0;
35  }
```

Tasks

0112233445667799

Tasks

```
1   void swap(int* a, int i, int j)
2   {
3      int tmp = a[i];
4      a[i] = a[j];
5      a[j] = tmp;
6   }
7
8   int partition(int* a, int lo, int hi) {
9      int j = lo - 1;
10     int p = a[hi - 1];
11     for(int i = lo; i < hi - 1; ++i)
12        if(a[i] <= p)
13           swap(a, ++j, i);
14
15     swap(a, ++j, hi - 1);
16     return j;
17  }
18
19  int quicksrt(int* a, int lo, int hi) {
20     printf("Thread %d: %d - %d\n", omp_get_thread_num(), lo, hi);
21     if(lo < hi)
22     {
23        int p = partition(a, lo, hi);
24
25        #pragma omp task firstprivate(lo, p)
26        quicksrt(a, lo, p);
27
28        #pragma omp task firstprivate(p, hi)
29        quicksrt(a, p + 1, hi);
30     }
31  }
```

Tasks

```
32  int main(int argc, char *argv[])
33  {
34     int array[] = {5, 1, 6, 4, 7, 2, 3, 6, 7, 9, 0, 1, 4, 2, 9, 3, 2};
35     #pragma omp parallel
36     {
37        #pragma omp single nowait
38        {
39           quicksrt(array, 0, 16);
40        }
41     }
42     for(int i = 0; i < 16; ++i)
43        printf("%d", array[i]);
44
45     return 0;
46  }
```

Tasks

```
19  int quicksrt(int* a, int lo, int hi) {
20     printf("Thread %d: %d - %d\n", omp_get_thread_num(), lo, hi);
21     if(lo < hi)
22     {
23        int p = partition(a, lo, hi);
24
25        #pragma omp task firstprivate(lo, p)
26        quicksrt(a, lo, p);
27
28        quicksrt(a, p + 1, hi);
29     }
30  }
```

Tasks

```
19  int quicksrt(int* a, int lo, int hi) {
20     printf("Thread %d: %d - %d\n", omp_get_thread_num(), lo, hi);
21     if(lo < hi)
22     {
23        int p = partition(a, lo, hi);
24
25        #pragma omp task firstprivate(lo, p) final(hi - lo < 5)
26        quicksrt(a, lo, p);
27
28        #pragma omp task firstprivate(p, hi) final(hi - lo < 5)
29        quicksrt(a, p + 1, hi);
30     }
31  }
```

Tasks

```
1   int numtasks = 15;
2   #pragma omp parallel
3   {
4      #pragma omp single nowait
5      {
6         #pragma omp taskloop
7         for(int i = 0; i < numtasks; ++i)
8            printf("Thread %d: %d\n", omp_get_thread_num(), i);
9      }
10  }
```

Tasks

```
Thread 1: 14
Thread 7: 12
Thread 2: 2
Thread 2: 3
Thread 0: 8
Thread 4: 4
Thread 3: 10
Thread 3: 11
Thread 4: 5
Thread 0: 9
Thread 7: 13
Thread 5: 6
Thread 6: 0
Thread 5: 7
Thread 6: 1
```

Tasks

```
1   int numtasks = 15;
2   #pragma omp parallel
3   {
4      #pragma omp single nowait
5      {
6         #pragma omp taskloop num_tasks(3)
7         for(int i = 0; i < numtasks; ++i)
8            printf("Thread %d: %d\n", omp_get_thread_num(), i);
9      }
10  }
```

Tasks

```
Thread 1: 10
Thread 3: 0
Thread 1: 11
Thread 1: 12
Thread 1: 13
Thread 1: 14
Thread 3: 1
Thread 4: 5
Thread 3: 2
Thread 4: 6
Thread 3: 3
Thread 4: 7
Thread 4: 8
Thread 4: 9
Thread 3: 4
```

Tasks

```
1   int numtasks = 15;
2   #pragma omp parallel
3   {
4      #pragma omp single nowait
5      {
6         #pragma omp taskloop grainsize(5)
7         for(int i = 0; i < numtasks; ++i)
8            printf("Thread %d: %d\n", omp_get_thread_num(), i);
9      }
10  }
```

Tasks

```
Thread 2: 10
Thread 3: 0
Thread 0: 5
Thread 3: 1
Thread 3: 2
Thread 3: 3
Thread 3: 4
Thread 2: 11
Thread 0: 6
Thread 2: 12
Thread 2: 13
Thread 2: 14
Thread 0: 7
Thread 0: 8
Thread 0: 9
```