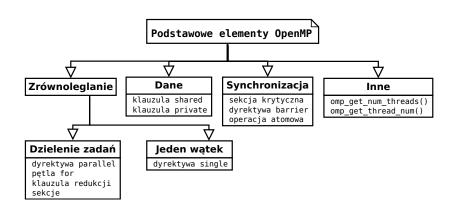
#### Podstawy interfejsu OpenMP

Zaawansowane techniki programowania



#### Dyrektywa parallel

```
#include <omp.h>
#include <iostream>
int main (int argc, char *argv[]) {
  #pragma omp parallel
    std::cout << "Jestem watek nr "</pre>
               << omp_get_thread_num() << "\n";</pre>
    if (omp_get_thread_num() == 0)
      std::cout << "Dodatkowe obliczenia przez "</pre>
                 << "watek nr 0 \n";
  return 0;
```

### Przykładowa sesja (w Linuksie)

```
ww@pcb34:~$ g++ -fopenmp main.cpp -o main
ww@pcb34:~$ ./main
Jestem watek nr 1
Jestem watek nr 0
Dodatkowe obliczenia przez watek nr 0
ww@pcb34:~$
```

### Zmienne prywatne i współdzielone

```
#include <omp.h>
#include <iostream>
int main (int argc, char *argv[]) {
  int wspolna=0, indyw=-1;
  #pragma omp parallel private(indyw) shared(wspolna)
   #pragma omp atomic
   wspolna += 1;
    indyw = omp_get_thread_num();
 std::cout << wspolna << " " << indyw << "\n";
  return 0;
```

```
ww@pcb34:~$ g++ -fopenmp main.cpp -o main
ww@pcb34:~$ ./main
2 -1
ww@pcb34:~$
```

#### Dyrektywa for

```
#include <omp.h>
#include <iostream>
int main (int argc, char *argv[]) {
  int tab[] = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\};
  #pragma omp parallel shared(tab)
    #pragma omp for
    for (int i=0; i<10; i++)
      tab[i] = tab[i]*tab[i]:
  for (int i=0: i<10: i++)
    std::cout << tab[i] << " ":
  std::cout << std::endl:
  return 0;
```

```
ww@pcb34:~$ g++ -fopenmp main.cpp -o main
ww@pcb34:~$ ./main
1 4 9 16 25 36 49 64 81 100
ww@pcb34:~$
```

#### Klauzula reduction

```
#include <omp.h>
#include <iostream>
int main (int argc, char *argv[]) {
  int suma=0, tab[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  #pragma omp parallel shared(suma, tab)
    #pragma omp for reduction(+:suma)
    for (int i=0: i<10: i++)
      suma = suma + tab[i];
  std::cout << suma << std::endl;</pre>
  return 0;
```

```
ww@pcb34:~$ g++ -fopenmp main.cpp -o main
ww@pcb34:~$ ./main
55
ww@pcb34:~$
```

# Operatory oraz wartości początkowe redukcji

Operator	Wartość	Operator	Wartość
+	0		0
*	1	^	0
_	0	&&	1
&	~0		0

### Inny przykład dotyczący redukcji

```
#include <omp.h>
#include <iostream>
int main (int argc, char *argv[]) {
  int iloczyn=0, tab[] = \{1, 2, 3, 4, 5\};
  #pragma omp parallel shared(iloczyn, tab)
    #pragma omp for reduction(*:iloczyn)
    for (int i=0; i<5; i++) {
      iloczyn = iloczyn * tab[i];
      std::cout << iloczyn << std::endl;</pre>
  std::cout << iloczyn << std::endl;</pre>
  return 0;
```

```
ww@pcb34:~$ g++ -fopenmp main.cpp -o main
ww@pcb34:~$ ./main
4
1
2
6
20
0
ww@pcb34:~$
```

# Sekcje i pojedynczy wątek (nast. slajd)

#include <omp.h>

```
#include <iostream>
#include <cmath>
int main (int argc, char *argv[]) {
  double wsin, wcos;
  #pragma omp parallel
    #pragma omp sections
      #pragma omp section
      \{ wsin = sin(0.1234); \}
      #pragma omp section
      \{ wcos = cos(0.1234); \}
  std::cout << wsin+wcos << std::endl;</pre>
                               Podstawy interfeisu OpenMP
```

```
ww@pcb34:~$ g++ -fopenmp main.cpp -o main
ww@pcb34:~$ ./main
1.11548
ww@pcb34:~$
```

```
#include <omp.h>
#include <iostream>
int main (int argc, char *argv[]) {
 double x, kwadrat, odwr;
  #pragma omp parallel
   #pragma omp single
    { std::cout << "Podaj x: "; std::cin >> x; }
   #pragma omp sections
      #pragma omp section
      { kwadrat = x*x; }
      #pragma omp section
      \{ odwr = 1.0/x; \}
  std::cout << kwadrat << " " << odwr << "\n";
```

```
ww@pcb34:~$ g++ -fopenmp main.cpp -o main
ww@pcb34:~$ ./main
Podaj x: 0.25
0.0625 4
ww@pcb34:~$
```

```
#include <omp.h>
#include <iostream>
int main (int argc, char *argv[]) {
 double x=1.0, kwadrat, odwr;
  #pragma omp parallel
   #pragma omp single nowait
    { std::cout << "Podaj x: "; std::cin >> x; }
   #pragma omp sections
      #pragma omp section
      { kwadrat = x*x; }
      #pragma omp section
      \{ odwr = 1.0/x; \}
  std::cout << kwadrat << " " << odwr << "\n";
```

```
ww@pcb34:~$ g++ -fopenmp main.cpp -o main
ww@pcb34:~$ ./main
Podaj x: 0.25
1 1
ww@pcb34:~$
```

```
#include <omp.h>
#include <iostream>
#include <set>
using namespace std;
int main (int argc, char *argv[]) {
  set<int> zbior;
  int i, x;
  #pragma omp parallel
    #pragma omp for private(i, x)
    for (i=1; i<=10; i+=2) {
      x = 3*i*i-2*i;
      #pragma omp critical
      { zbior.insert(x); } // sekcja krytyczna
  cout << zbior.size() << endl;</pre>
```

```
ww@pcb34:~$ g++ -fopenmp main.cpp -o main
ww@pcb34:~$ ./main
5
ww@pcb34:~$
```

```
#include <omp.h>
#include <iostream>
int main (int argc, char *argv[]) {
  int a=0, b=0, c=0, d=0;
  #pragma omp parallel sections
    #pragma omp section
    \{ a = 1; \}
    #pragma omp section
    \{ b = 2; \}
    #pragma omp section
    \{ c = a+b; \}
    #pragma omp section
    \{ d = c+1; \}
  std::cout << d << "\n";
} // Jaki będzie wynik działania?
```

```
ww@pcb34:~$ g++ -fopenmp main.cpp -o main
ww@pcb34:~$ ./main
1
ww@pcb34:~$
```

### Prawidłowy sposób

```
#include <omp.h>
#include <iostream>
int main (int argc, char *argv[]) {
  int a=0, b=0, c=0, d=0;
  #pragma omp parallel sections
    #pragma omp section
    \{ a = 1; \}
    #pragma omp section
   \{ b = 2; \}
  c = a+b;
  d = c+1;
  std::cout << d << "\n";
```

```
ww@pcb34:~$ g++ -fopenmp main.cpp -o main
ww@pcb34:~$ ./main
4
ww@pcb34:~$
```

### Dyrektywa barrier

```
#include <omp.h>
#include <iostream>
using namespace std;
int tab1[5] = \{0,0,0,0,0,0\};
int tab2[5][10] = {
  \{1,2,3,4,5,6,7,8,9,10\},
  \{11, 12, 13, 14, 15, 16, 17, 18, 19, 20\},\
  \{21,22,23,24,25,26,27,28,29,30\},
  {31,32,33,34,35,36,37,38,39,40},
  {41,42,43,44,45,46,47,48,49,50}
};
```

```
// ...
int min(int* a, int n) {
  int m = a[0];
  for(int i=1; i<n; i++)
    if (a[i] < m)
        m = a[i];
  return m;
}
// ...</pre>
```

```
int main (int argc, char *argv[]) {
  int i, p, wynik=0;
  #pragma omp parallel private(i, p)
    i = omp_get_thread_num();
    p = omp_get_num_threads();
    while (i < 5) {
      tab1[i] = min(tab2[i], 10);
      i += p;
    #pragma omp barrier
    #pragma omp for reduction(+:wynik)
    for (int j=0; j<5; j++)
      wynik = wynik + tab1[j];
  cout << wynik << "\n";</pre>
```

```
ww@pcb34:~$ g++ -fopenmp main.cpp -o main
ww@pcb34:~$ ./main
105
ww@pcb34:~$
```

## Ćwiczenie 1. (trójkąt Pascala)

```
#include <omp.h>
#include <iostream>
using namespace std;
int t[5][5] = {
  {1},
  \{1, 1\},\
  \{1, 0, 1\},\
  \{1, 0, 0, 1\},\
  \{1, 0, 0, 0, 1\}
```

### Gdzie wstawić dyrektywę parallel for?

```
int main (int argc, char *argv[]) {
  int i, j;
 // Tu? #pragma omp parallel for private(i, j)
 for (i=2; i<5; i++) {
   // Czy tu? #pragma omp parallel for
    for (j=1; j<i; j++)
      t[i][j] = t[i-1][j-1] + t[i-1][j];
  cout << t[4][1] << t[4][2] << t[4][3] << "\n";
```

# Ćwiczenie 2. (jaki będzie wynik działania?)

```
#include <omp.h>
#include <iostream>
int t[10] = \{0,0,0,0,0,0,0,0,0,0,0,0\};
int main (int argc, char *argv[]) {
  int i, n=0;
  #pragma omp parallel for shared(t, n)
  for (i=2; i<6; i++) {
    int x = i*i - 1:
    if (x \% 2 == 0) t[n++] = x:
  std::cout << t[n-1] << "\n";
```

## Prawidłowa konstrukcja (z sekcją krytyczną)

```
#include <omp.h>
#include <iostream>
int t[10] = \{0,0,0,0,0,0,0,0,0,0,0,0\};
int main (int argc, char *argv[]) {
  int i, n=0;
  #pragma omp parallel for shared(t, n)
  for (i=2; i<6; i++) {
    int x = i*i - 1:
    if (x \% 2 == 0) {
      #pragma omp critical
      t[n++] = x:
  std::cout << t[n-1] << "\n";
} // wynik: 8 lub 24, ale nie 0
```

#### Wnioski końcowe

OpenMP jest standardem programowania równoległego, który zdobył popularność ze względu na:

- poparcie czołowych firm komputerowych (IBM, Intel, Compaq i in.),
- możliwość programowania równoległego z użyciem pamięci wspólnej,
- to, że elementy tego interfejsu są akceptowane przez kompilatory języków C, C++ i Fortran.