# Programowanie równoległe z użyciem biblioteki MPI

Zaawansowane techniki programowania

#### Plan wykładu

- Wprowadzenie
  - Programowanie równoległe
  - Standard MPI
- 2 Klasy problemów rozwiązywanych równolegle
- 3 Podstawowe funkcje MPI
  - Struktura programu MPI
  - Sposoby komunikacji
- Przykłady
- Podsumowanie

#### Podstawowe pojęcia

**Proces** — program (sekwencyjny) w trakcie wykonywania.

Mówimy, że dwa procesy działają **równolegle** (są **współbieżne**), jeśli jeden z nich rozpoczyna się przed zakończeniem drugiego.

Gdy programowanie równoległe realizowane jest w systemie wielokomputerowym (a nie na komputerze wieloprocesorowym), to mówimy o programowaniu **rozproszonym**.

#### Klasyfikacja komputerów wg Flynn'a

- SISD single-instruction single-data
- MISD multiple-instruction single-data
- SIMD single-instruction multiple-data
- MIMD multiple-instruction multiple-data
  - z pamięcią współdzieloną
  - z pamięcią rozproszoną

#### Czym jest MPI?

Message-Passing Interface (MPI)									
jest	nie jest								
<ul><li>standardem programowania</li></ul>									
współbieżnego w systemach	<ul><li>językiem pro-</li></ul>								
MIMD z pamięcią rozproszoną	gramowania								
<ul><li>specyfikacją syntaktyki</li></ul>	<ul><li>programem</li></ul>								
i semantyki podprogramów za	komputero-								
pomocą których realizowana jest	wym								
komunikacja między procesami									

#### Dlaczego warto znać MPI?

- Niekwestionowany standard realizacji obliczeń równoległych
  - Klastry obliczeniowe w Centrum Informatycznym TASK
  - Częstochowski Akademicki Klaster Obliczeniowy
  - Earlham College Cluster Computing Lab
  - Earth Simulator (szybkość do 40 Tflops)
- ② Dostępność bibliotek MPI działających także na sieciach TCP/IP (np. MPICH2 pod adresem http://www.mcs.anl.gov/research/projects/mpich2/)

#### Otwarte problemy w teorii złożoności obliczeniowej

- P ≠ NP ?
- $NC \neq P$  ?

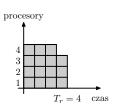
NC — od: Nick Pippenger's class

#### Podstawowe pojęcia i oznaczenia

- p liczba procesorów
- n rozmiar problemu (danych wejściowych)
- $T(p, n) = \sup_{d \in D_n} \{t(p, d)\}$  pesymistyczna złożoność czasowa algorytmu równoległego
- $S(p, n) = \frac{T(1,n)}{T(p,n)}$  przyspieszenie algorytmu równoległego

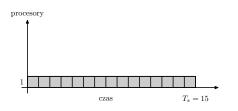
#### Przyspieszenie algorytmu równoległego

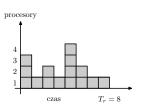




Przyspieszenie idealne:  $S_{id} = \frac{T_s}{T_r} = \frac{15}{4} = 3,75$ Idealny (najkrótszy) czas obliczeń równoległych:  $T_r = \lceil \frac{T_s}{p} \rceil$ 

#### Przyspieszenie algorytmu równoległego (c.d.)





Przyspieszenie idealne: 
$$S_{id} = \frac{T_s}{T_r} = \frac{15}{4} = 3,75$$

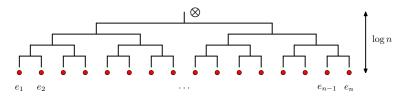
Przyspieszenie rzeczywiste: 
$$S_{rz} = \frac{T_s}{T_r} = \frac{15}{8} = 1,875$$

#### Co i na ile da się przyspieszyć?

- Akceptowalna liczba procesorów wielomianowa,  $n^{O(1)}$
- Rozwiązywania problemów klasy NP nie da się <u>istotnie</u> (w sensie rzędu) przyspieszyć z użyciem wielomianowej liczby procesorów
- Da się to częściowo zrobić dla klasy P w pewnych przypadkach czas równoległego rozwiązywania problemu klasy P jest O(1)

### Problem redukcji — rozwiązywany w czasie $O(\log n)$

 $\otimes$  — dowolna operacja łączna, np. +, \*, or, and, min, max



$$T(p, n) = \log n \quad \leftarrow \text{złożoność obliczeniowa}$$

$$S(p, n) = \frac{n-1}{\log n} = O(n/\log n)$$
  $\leftarrow$  przyspieszenie

Złożoność procesorowa: p(n) = n

Następuje wykładnicze skrócenie czasu obliczeń:

$$n/\log n = 2^{\log n}/\log n = 2^x/x$$



#### Problem sortowania — rozwiązywany w czasie $O(\log n)$

#### Dane:

Ciąg elementów, który należy posortować:  $e_1, e_2, \ldots, e_n$ 

Wirtualna siatka procesorów:  $\{P_{i,j}\}$ ,  $1 \le i, j \le n$ 

$$P_{1,1}$$
  $P_{1,2}$   $P_{1,3}$  ...  $P_{1,n}$   $P_{2,1}$   $P_{2,2}$   $P_{2,3}$  ...  $P_{2,n}$  ...  $P_{n,1}$   $P_{n,2}$   $P_{n,3}$  ...  $P_{n,n}$ 

#### Problem sortowania — przykład

Ciąg do sortowania:  $[e_1, e_2, e_3, e_4] = [5, -1, 2, 0]$ 

Ciąg posortowany:  $[e_2, e_4, e_3, e_1] = [-1, 0, 2, 5]$ 

Złożoność sortowania:  $O(\log n)$ Złożoność procesorowa:  $p(n) = n^2$ 

Dla posortowania ciągu 1000-elementowego potrzeba

1 mln procesorów



#### Problem redukcji — szukanie minimum w czasie O(1)

Szukamy elementu min. w ciągu:  $[e_1, e_2, e_3, e_4] = [5, -1, 2, 0]$ 

	1			4	_t
1	$e_1 > e_1$	$e_1 > e_2$	$e_1 > e_3$	$e_1 > e_4$	1
2	$e_1 > e_1 \ e_2 > e_1$	$e_2 > e_2$	$e_2 > e_3$	$e_2 > e_4$	0
3	$e_3 > e_1$	$e_3 > e_2$	$e_3 > e_3$	$e_3 > e_4$	1
4	$e_3 > e_1$ $e_4 > e_1$	$e_4 > e_2$	$e_4 > e_3$	$e_4 > e_4$	1

Złożoność szukania minimum: O(1)Złożoność procesorowa:  $p(n) = n^2$ 

### Klasa problemów NC — zachodzi $NC \subseteq P$ od: Nick Pippenger's class

- Klasa NC obejmuje problemy, które można rozwiązać za pomocą szybkich algorytmów równoległych, tj.
   o złożonościach czasowych wielomianowo-logarytmicznych, (log n)<sup>O(1)</sup>, oraz wielomianowych złożonościach procesorowych, n<sup>O(1)</sup>.
- O problemach klasy NC mówi się, że są efektywnie rozwiązywane w sposób równoległy (ang. efficiently parallelizable problems).
- Zachodzi NC ⊆ P, tj. problemy klasy NC zawierają się w klasie problemów rozwiązywanych w czasie wielomianowym za pomocą wielomianowej liczby procesorów (zmodyfikowana definicja klasy P).

#### Czy $NC \neq P$ ?

- Aby rozstrzygnąć czy  $NC \neq P$ ? wystarczy podać jeden problem z klasy P, dla którego można udowodnić, że nie da sie go rozwiązać za pomocą algorytmu równoległego o złożoności  $(\log n)^{O(1)}$  przy  $n^{O(1)}$  procesorach.
- Niestety mimo wielu prób nie udało sie jak dotąd znaleźć takiego problemu — a może raczej dowodu, że problem nie należy do klasy NC.

#### Zbiór P - NC, czyli problemy "z natury" sekwencyjne

- Uważa się, że do zbioru P-NC należą problemy, co do których istnieje duże prawdopodobieństwo, że da się je rozwiązać jedynie(!) w czasie wielomianowym  $n^{O(1)}$  za pomocą wielomianowej liczby procesorów  $n^{O(1)}$ .
- Problemy te określa się jako "z natury sekwencyjne" lub inherentnie sekwencyjne.

#### Problemy P-zupełne — najtrudniejsze w klasie P

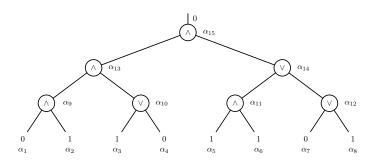
- Wśród problemów "z natury sekwencyjnych" wyodrębniono ok. 140 problemów P-zupełnych, tj. najtrudniejszych w klasie P
- Problem B jest P-zupełny, jeśli da się go rozwiązać w czasie  $n^{O(1)}$  przy użyciu  $n^{O(1)}$  liczby procesorów, oraz każdy problem A z klasy P da się zredukować do problemu B w czasie ( $\log n$ ) $^{O(1)}$  przy użyciu  $n^{O(1)}$  procesorów.

## P-zupełny problem wartości układu logicznego Circuit Value Problem (CVP)

- $B_2 = \{f : \{0,1\}^2 \rightarrow \{0,1\}\}$  zbiór dwuargumentowych funkcji boolowskich
- Funkcje oznaczamy tradycyjnymi symbolami ∧ i ∨
- **Definicja**: Układ logiczny  $\alpha$  o rozmiarze n nad bazą  $B_2$  jest sekwencją składowych  $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ , gdzie każda składowa  $\alpha_i$  jest zmianną wejściową  $x_i$ , lub bramką  $f(\alpha_j, \alpha_k)$  dla pewnej funkcji  $f \in B_2$ , gdzie j, k < i. Wartość  $v(\alpha_i)$  składowej  $\alpha_i$  jest następująca: Jeśli  $\alpha_i = x_i$ , to  $v(\alpha_i) = v(x_i)$ ; jeśli  $\alpha_i = f(\alpha_j, \alpha_k)$ , to  $v(\alpha_i) = f(v(\alpha_j), v(\alpha_k))$ . Wartość układu  $v(\alpha)$  definiujemy jako  $v(\alpha_n)$ .

#### P-zupełny problem wartości układu logicznego (c.d.)

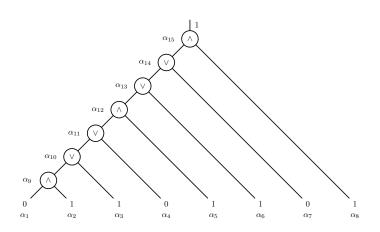
Dany jest układ logiczny  $\alpha$  nad bazą  $B = \{\land, \lor\}$  oraz wartości zmiennych wejściowych  $x_i$  układu równe 0 lub 1. Należy odpowiedzieć na pytanie, czy wartość układu  $v(\alpha) = 1$ .



Wartość układu logicznego o takim kształcie da się wyznaczyć sekwencyjnie w czasie O(n), a równolegle w czasie  $\log((n+1)/2) = O(\log n)$ 



#### P-zupełny problem wartości układu logicznego (c.d.)



Wartość układu logicznego o takim kształcie da się wyznaczyć sekwencyjnie w czasie O(n), a równolegle także w czasie O(n) (?)



#### Podsumowanie dotyczące złożoności

- Na problemy NP-zupełne nie pomoże żaden wyrafinowany algorytm — tak jak są one wykładnicze, tak i pozostaną
- Na problemy P-zupełne nie pomoże żadna liczba procesorów
  - tak jak są one wielomianowe, tak i pozostaną

#### Funkcja inicjalizująca i finalizująca

```
#include "mpi.h"
int main(int argc, char* argv[])
 /* Musi być wywołana jako pierwsza funkcja MPI */
  MPI_Init(&argc, &argv);
  MPI_Finalize();
  /* Po tej funkcji nie można wywoływać żadnych funkcji MPI */
```

#### Odczytywanie numeru i liczby procesów

```
int moj_nr; /* numer procesu */
int p; /* liczba procesów */
/* Odczytaj numer procesu */
MPI_Comm_rank(MPI_COMM_WORLD, &moj_nr);
/* Odczytaj liczbę procesów */
MPI_Comm_size(MPI_COMM_WORLD, &p);
```

#### Funkcje MPI\_Send i MPI\_Recv

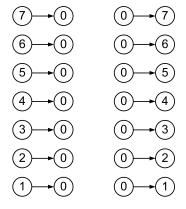
#### Rozgłaszanie

```
// ...
float a;

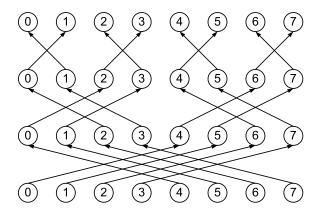
if (moj_nr == 0) {
    printf("Podaj a: ");
    scanf("%f", &a);
}

MPI_Bcast(&a, 1, MPI_FLOAT, 0, MPI_COMM_WORLD);
// ...
```

#### Przykład 1 — sumowanie zawartości zmiennej



#### Przykład 1 — rozwiązanie optymalne



#### Przykład 1 — realizacja w MPI

#### Przykład 2 — problem plecakowy

```
#include <cstdio>
#include <string>
#include "mpi.h"
using namespace std;
int my_rank, p, source, dest, tag = 0, N = 28;
MPI_Status status:
unsigned long dwa_do_N, max_ciag;
unsigned short max_suma = 0, pojemnosc = 2500;
unsigned short tab[] = \{3, 121, 345, 34, 67, 380, 223, 
   169, 10, 12, 56, 22, 52, 499, 419, 442, 99, 7, 29,
  30, 144, 26, 2, 676, 147, 8, 188, 16};
```

```
unsigned short sumuj(unsigned long nr_podzbioru)
 /* Sumuje elementy podzbioru o numerze nr_podzbioru */
void sprawdzaj (unsigned long pocz, unsigned long koniec)
  /* Szuka sumy najbliższej pojemności plecaka dla
     podzbiorów o numerach od "pocz" do "koniec" */
```

```
int main(int argc, char* argv[])
 MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &p);
 dwa_do_N = (1 << N);
  unsigned long odcinek = dwa_do_N / p;
  if (my_rank < p-1)
    sprawdzaj(my_rank*odcinek, (my_rank+1)*odcinek);
 else
   sprawdzaj(my_rank*odcinek, dwa_do_N);
```

```
if (my_rank != 0) {
  dest = 0;
  MPI_Send(&max_ciag, 1, MPI_UNSIGNED_LONG,
           dest, tag, MPI_COMM_WORLD);
else {
  unsigned long odczytany;
  unsigned short x;
```

```
for (source = 1; source < p; source++) {</pre>
    MPI_Recv(&odczytany, 1, MPI_UNSIGNED_LONG,
              source, tag, MPI_COMM_WORLD, &status);
    if ((x = sumuj(odczytany)) > max_suma) {
      max_suma = x;
      max_ciag = odczytany;
  printf("%d, %d", max_suma, max_ciag);
MPI_Finalize():
```

#### Przykład 3 — problem komiwojażera

dij	<i>v</i> <sub>1</sub>	<i>v</i> <sub>2</sub>	<i>v</i> <sub>3</sub>	V4	<i>v</i> <sub>5</sub>	<i>v</i> <sub>6</sub>	V7	<i>v</i> <sub>8</sub>	<i>V</i> 9	v <sub>10</sub>	v <sub>11</sub>	v <sub>12</sub>	v <sub>13</sub>
$v_1$	-	3	121	345	34	67	380	223	169	10	12	2	3
<i>v</i> <sub>2</sub>	564	-	56	22	7	52	499	419	442	99	7	29	30
<i>V</i> 3	33	8	-	244	572	662	463	414	590	54	602	94	3
V4	4	474	461	-	547	333	252	245	570	34	658	7	473
<i>v</i> <sub>5</sub>	2	473	138	673	-	185	554	77	437	49	385	688	197
<i>v</i> <sub>6</sub>	8	572	59	427	19	-	440	173	267	220	657	450	435
ν <sub>7</sub>	44	50	360	412	366	58	-	666	361	347	538	266	476
<i>v</i> <sub>8</sub>	97	41	136	519	597	77	190	-	211	22	505	441	246
<i>V</i> 9	77	344	140	416	480	600	21	407	-	90	144	517	384
v <sub>10</sub>	345	23	619	268	20	369	64	654	56	-	551	451	591
v <sub>11</sub>	2	338	541	62	567	225	483	57	562	346	-	584	239
v <sub>12</sub>	65	144	26	2	676	147	8	188	16	34	12	-	87
v <sub>13</sub>	323	55	89	400	553	178	20	30	637	78	74	632	-

### Reprezentacja tablicy odległości

```
unsigned short d[N][N] = {
\{0,3,121,345,34,67,380,223,169,10,12,2,3\},
{564,0,56,22,7,52,499,419,442,99,7,29,30},
{33,8,0,244,572,662,463,414,590,54,602,94,3},
{4,474,461,0,547,333,252,245,570,34,658,7,473},
{2,473,138,673,0,185,554,77,437,49,385,688,197},
{8,572,59,427,19,0,440,173,267,220,657,450,435},
{44,50,360,412,366,58,0,666,361,347,538,266,476},
{97,41,136,519,597,77,190,0,211,22,505,441,246},
{77,344,140,416,480,600,21,407,0,90,144,517,384},
{345,23,619,268,20,369,64,654,56,0,551,451,591},
{2,338,541,62,567,225,483,57,562,346,0,584,239},
{65,144,26,2,676,147,8,188,16,34,12,0,87},
{323,55,89,400,553,178,20,30,637,78,74,632,0}};
```

## Silniowy system pozycyjny (ang. factoradic)

Silniowy system pozycyjny — pozycyjny system liczbowy, w którym mnożniki poszczególnych pozycji nie są definiowane przez potęgę pewnej liczby (podstawy), lecz silnię kolejnych liczb naturalnych (z zerem), a liczba cyfr używanych na n-tej pozycji wynosi n+1.

$$(9643)_{10} = 1 \times 7! + 6 \times 6! + 2 \times 5! + 1 \times 4! + 3 \times 3! + 0 \times 2! + 1 \times 1! + 0 \times 0!$$

Czyli 
$$f(9643,8) = [1,6,2,1,3,0,1,0]$$



## Implementacja funkcji f

```
vector<int> f(unsigned long x, int n)
  /* Odwzorowuje liczbę całkowita x w "factoradic"
     o długości n */
  if (x == 0) return vector<int>(n, 0);
  else {
    unsigned long s = silnia(n-1);
    int a = x / s:
    vector < int > v = f(x - a*s, n-1);
    v.insert(v.begin(), a);
    return v;
```

# Bijekcja pomiędzy liczbami a permutacjami

$$f(19,4) = [3,0,1,0]$$

### Implementacja funkcji perm

```
vector<int> perm(vector<int> fact)
 /* Tworzy permutację liczb od 0 do n-1
     na podstawie "factoradic" */
  int i, n = fact.size();
  vector<int> t(n);
  for (i=0; i< n; i++)
    t[i] = i:
  vector<int> v;
  for (i=0; i< n; i++) {
    v.push_back(t[fact[i]]);
    remove(t.begin(), t.end(), v.back());
  return v;
```

### Program rozwiązujący problem komiwojażera

```
#include <iostream>
#include <vector>
#include <algorithm>
#include "mpi.h"
using namespace std;
#define N 13
int my_rank, p;
int source, dest, tag = 0;
MPI_Status status:
unsigned long min_dlug = 99999999;
vector<int> min_trasa;
```

```
unsigned short d[N][N] = { /* ... */ };
double st=0.0, et;
unsigned long silnia(unsigned long n) { /* ... */ }
vector<int> f(unsigned long x, int n) { /* ... */ }
vector<int> perm(vector<int> fact) { /* ... */ }
```

```
unsigned long oblicz(vector<int> &v)
 /* Wyznacza długość trasy
  unsigned long wynik=0;
  for (int i=1: i<N-1: i++)
    wynik += d[v[i-1]][v[i]];
  wynik += d[v[N-2]][N-1];
  wynik += d[N-1][v[0]];
  return wynik;
```

```
void sprawdzaj (unsigned long pocz, unsigned long koniec)
  /* Sprawdza, która spośród tras od trasy o numerze "pocz"
     do trasy o numerze "koniec" jest najkrótsza
  unsigned long dlug;
  vector < int > v = perm(f(pocz, N-1));
  for (unsigned long i=pocz; i<koniec; i++) {</pre>
    if ((dlug = oblicz(v)) < min_dlug) {</pre>
      min_dlug = dlug;
      min_trasa = v;
    next_permutation(v.begin(), v.end());
```

```
int main(int argc, char* argv[])
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 MPI_Comm_size(MPI_COMM_WORLD, &p);
  unsigned long odcinek = silnia(N-1) / p;
  cout << "Rozpoczynam obliczenia." << endl;</pre>
  if (my_rank == 0) st = MPI_Wtime();
  if (my\_rank < p-1) {
    sprawdzaj(my_rank*odcinek, (my_rank+1)*odcinek);
    cout << "skończyłem " << min_dlug << endl;</pre>
 else {
    sprawdzaj(my_rank*odcinek, silnia(N-1));
    cout << "skończyłem " << min_dlug << endl;</pre>
```

```
int mt[N-1];
if (my_rank != 0) {
  copy(min_trasa.begin(), min_trasa.end(), mt);
  MPI_Send(mt, N-1, MPI_INT, 0, tag, MPI_COMM_WORLD);
} else {
  unsigned long dlug;
  for (source = 1; source < p; source++) {</pre>
    MPI_Recv(mt, N-1, MPI_INT, source, tag,
              MPI_COMM_WORLD, &status);
    vector < int > v(mt, mt + (N-1));
    if ((dlug = oblicz(v)) < min_dlug) {</pre>
      min_dlug = dlug; min_trasa = v;
  et = MPI_Wtime();
  cout << min_dlug << " " << " Czas: " << et-st;</pre>
MPI_Finalize(); return 0;}
```

# Przykładowa sesja (w systemie MPICH2)

```
ww@pcb34:~/temp$ mpicxx -o tsp tsp.cpp
ww@pcb34:~/temp$ mpd &
[1] 6431
ww@pcb34:~/temp$ mpiexec -l -n 2 ./tsp
0: Rozpoczynam obliczenia.
1: Rozpoczynam obliczenia.
1: skończyłem 310
0: skończyłem 376
0: 310 Czas: 11.7448
ww@pcb34:~/temp$ mpiexec -l -n 1 ./tsp
0: Rozpoczynam obliczenia.
0: skończyłem 310
0: 310 Czas: 23.4105
ww@pcb34:~/temp$ mpdallexit
```

#### Wnioski końcowe

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

- dostępność biblioteki na większość systemów wieloprocesorowych i sieci komputerowe,
- 2 implementację w języku C, C++, Fortran, Python i in.