PARADYGMATY I JĘZYKI PROGRAMOWANIA

Programowanie współbieżne ... (w13)

Treść

- Wstęp
- Procesy i wątki
- Szybkość obliczeń
  - prawo Amdahla
- Wyścig do zasobów
- Synchronizacja i mechanizmy synchronizacji
  - semafory
  - monitory
  - inne mechanizmy synchronizacji
- MPI
  - Podstawy
  - Przykłady
- OpenMP
  - Podstawy

Wstęp

Do czego potrzebujemy OR?

- Aplikacje CAD/CAM
- Przewidywanie pogody
- Systemy inteligentne
- Modelowanie (ekonomia, planowanie itd)
- Opracowywanie danych satelitarnych (...)
- Projektowanie obwodów VLSI
- Problemy energetyki jądrowej
- Reakcje chemiczne i jądrowe
- Aktywność sejsmiczna Ziemi
- Problemy genetyki
- inne

Prawo Amdahla

Gene Amdahl, 1967

Przyspieszenie obliczeń współbieżnych

$$S \leq \frac{1}{f + \frac{(1-f)}{n}}$$

gdzie *f* = część (<1) odpowiadająca obliczeniom sekwencyjnym (*1-f* odpowiada tej części programu, ktora daje sie uwspółbieżnić).

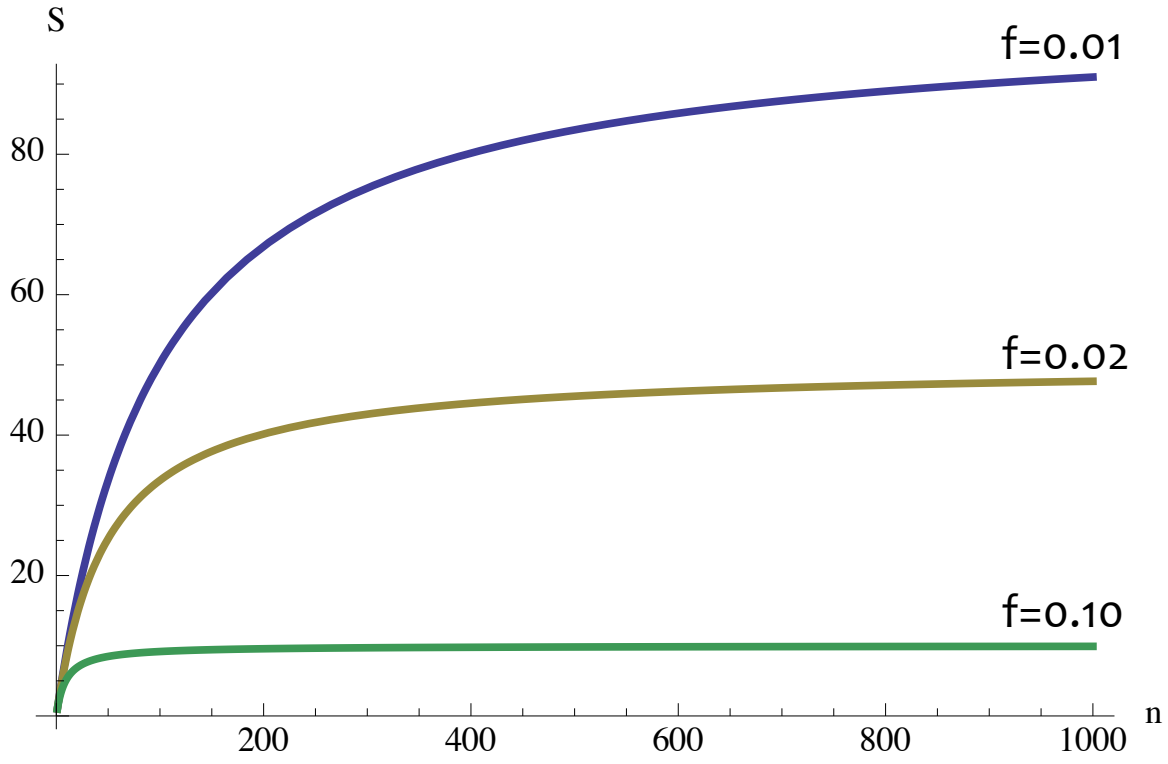
Prawo Amdahla

- Obliczenia na n procesorach:
  n – liczba procesorów
  T – całkowity czas obliczeń na 1 procesorze
  *f*T* – czas obliczeń sekwencyjnych na 1 procesorze
  *(1-f)*T* – czas obl. równoległych na 1 procesorze
  *(1-f)*T/n* – czas obl. równoległych na n procesorach
- Przyspieszenie obliczeń
  (czas obl. 1proc./czas obl. nproc.):

$$S \leq \frac{T}{fT + \frac{(1-f)T}{n}}$$

Stąd prawo Amdahla.

Prawo Amdahla



$$S \leq \frac{1}{f + \frac{(1-f)}{n}}$$

Przykład. Iloczyn skalarny

- 1 proces

Suma: $s = \sum_{i=1}^{N} x_i y_i$ Czas 1 operacji + : δt

Liczba operacji +: N-1; czas obliczeń: $T_1 = (N-1)\delta t$

- 2 procesy

Suma: $s = \sum_{i=1}^{N/2} x_i y_i + \sum_{i=N/2+1}^{N} x_i y_i$

Czas obliczeń: $T_2 = (N/2-1)\delta t + \delta t + C$

gdzie C jest czasem przesyłania wyniku częściowego – czas komunikacji

Przyspieszenie: $S = T_1/T_2 = \frac{N-1}{N/2+C/\delta t} \rightarrow 2$

Jeśli $c << \delta t$

Przykład. Iloczyn skalarny

- P procesorów. Zakładamy, że $P = N, \quad N = 2^q$

$$S_P = \frac{T_1}{T_P} = \frac{(N-1)\delta t}{q\delta t + qC}$$

(qC – całkowity czas komunikacji). Kładąc $\alpha \equiv C/\delta t$ otrzymamy

$$S_P = \frac{N-1}{(1+\alpha)\log_2 P} = \frac{P-1}{(1+\alpha)\log_2 P}$$

Nawet dla $\alpha = 0$

$$S_P = \frac{P-1}{\log_2 P} < P$$

Współbieżność

- Współbieżność – wiele kontekstów obliczeniowych jednocześnie
  - wielowątkowość; wielozadaniowość
  - dodatkowe procesory; szybkość
  - komputery rozproszone; internet
- Różne poziomy zrównoleglania
  - różne poziomy ziarnistości
    - instruction Level Parallelism
    - vector parallelism
    - thread-level parallelism

Wyścig do zasobów

- Wyścig do zasobów ma miejsce jeśli działają co najmniej dwa niezsynchronizowane procesy i wynik obliczeń zależy od kolejności ich działania
- Czasami wyścig do danych nie zagraża wynikom obliczeń (np. pobieranie zadań z kolejki)
- W ogólności, wyścig do zasobów jest niepożądany i należy go kontrolować

Wyścig do zasobów

- Przykład
  Dwa procesy dzielą pamięć, zwiększając o 1 wartość zmiennej X. Większość procesorów nie wykonuje operacji arytmetycznych bezpośrednio na pamięci.
- Każdy z procesorów wykonuje operacje:
  - LOAD X
  - INC
  - STORE X
- Jaki będzie wynik obliczeń jeśli początkowa wartość X=0, a procesy działają równolegle? Czy wynikiem końcowym będzie 1, czy 2?
- Przykład wielowątkowości: serwery sieciowe

Architektura

- Dwa główne typy architektury
  - dzielona pamięć – wiele procesorów jedna pamięć; *OpenMP*
  - każdy procesor posiada własną pamięć – komunikaty; *MPI*

Jaguar & Titan

- http://computing.ornl.gov/
- http://top500.org

Titan is a Cray XK7 system that contains 18,688 nodes, each built from a 16-core AMD Opteron 6274 processor and an NVIDIA Tesla K20X GPU accelerator. Titan also has 710 terabytes of memory.

The OLCF is home to Titan, the world's most powerful supercomputer for open science with a theoretical peak performance exceeding 20 petaflops (quadrillion calculations per second). That kind of computational capability—almost unimaginable—is on par with each of the world's 7 billion people being able to carry out 3 million calculations per second. Image courtesy Oak Ridge National Laboratory.

Synchronizacja

- Tak, czy owak synchronizacja pracy wątków, czy procesów jest podstawą obliczeń równoległych
- Synchronizację zapewnia język programowaia (najczęściej programista)
- Języki i rozszerzenia
  - Java, C#
  - MPI (Message Passing Interface) – wywołania zdalne procedur; dostępność: Fortran, C++; wszystkie platformy
  - Biblioteki: POSIX *pthreads* – biblioteki sieciowe

Narodziny i śmierć wątków

- Systemy programowania współbieżnego pozwalają kreować i niszczyć wątki
- Składnia instrukcji jest różna w różnych systemach
- Przykład, instrukcje „*do*”
  - OpenMP

```
#pragma omp parallel for
for (int i = 0; ji < 3; i++) {
  print („thread %d”\n, i);
}
```

  - C#

```
Parallel.For(0, 3, i => {
  Console.WriteLine(„Thread ” + i);
})
```

    Trzeci argument `Parallel.For` jest delegatem, tutaj wyrażeniem lambda.

Narodziny i śmierć wątków

- Java

```
class ImageRenderer extends Thread {
  ...
  ImageRenderer( args ) {
  // konstruktor
  }
  public void run() {
        // kod wątku
  }
}
...
ImageRenderer rend = new ImageRenderer
(arg_konstruktora)
```

- Wątek uruchamia polecenie `rend.start()`, łączy `rend.join()`
- Podobnie w C#

Synchronizacja

Podstawy

Synchronizacja

- Operacje atomowe lub opóźnienie operacji do chwili gdy spełnione są określone warunki (np. zrealizowanie obliczania tablicy elementów)
- *wykluczanie* – tylko jeden wątek działa w tzw. *sekcji krytycznej*
- *synchronizacja warunkowa* – wątki czekaja na spełnienie określonego warunku
- *barriers*

Mechanizmy synchronizacji

- Semafory

  (Dijkstra, 1968): semafor – licznik o dwóch operacjach P i V (P – *passeren* = dun. przejść, V – *vrijgeven* = zwolnić; Algol 68: *down* i *up*). Wątek, który wywołuje P automatycznie zmniejsza licznik o 1 i czeka do momentu gdy licznik będzie nieujemny. Wątek, wywołujący V, zwiększa licznik o 1 i w ten sposób aktywuje wątek oczekujący (jeśli taki jest); semafor binarny;

Mechanizmy synchronizacji

- Monitory

  (Hansen, 1973; Hoare 1974): moduł lub obiekt, posiadajacy pewne operacje, stan wewnętrzny i jakies zmienne, określające warunki. W danej chwili może byc wykonywana tylko jedna operacja. Wątek, wywołujący zajęty monitor, jest automatycznie wstawiany do kolejki wątków oczekujacych (kolejka wejściowa) do czasu gdy monitor sie zwolni.

Zakleszczenie, impas

- Nieumiejętne posługiwanie się semaforami lub monitorami może prowadzić do zakleszczenia (*deadlock*)

Synchronizacja w Java

- Instrukcja **`synchronized`**

```
synchronized (obiekt_dzielony) {
    // sekcja krytyczna
}
```

- metoda **`wait`**() pozwala zawiesić działanie wątku do odpowiedniego momentu.

```
while (!warunek) {
    wait();
}
```

- Wątek zawieszony może być aktywowany przez inne wątki, wywołujące metode **`notify()`** lub **`notifyAll()`**.

Atomowość inaczej (TM)

- Organizacja atomowych operacji w programach jest skomplikowana
- Lepiej: system zarządza konkurującymi wątkami
- TM – transactional memory; analogicznie, jak w przypadku transakcyjnych baz danych (atomowość, spójność, izolacja, trwałość)
- Podstawowa idea (TM). W programie:

```
atomic {
    -- kod do wykonania
}
```

DPM

Data Parallel Model

Data Parallel Model (DPM)

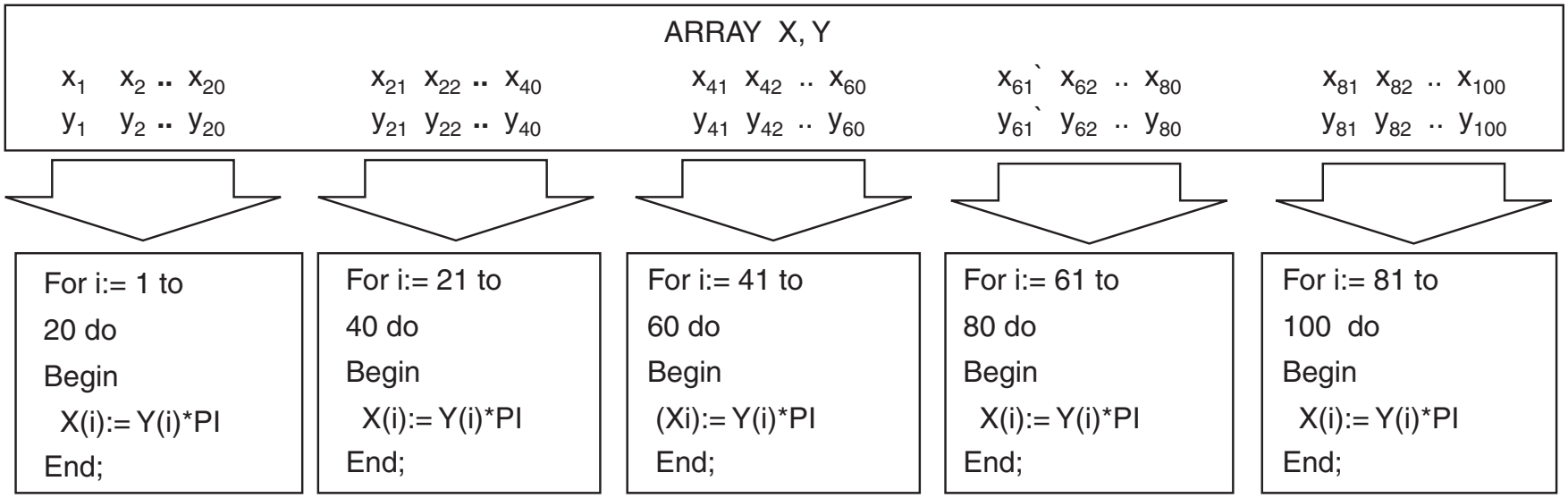
- Operacje na zbiorach danych (np na tablicach)
- Zespół zadań (program) operuje na wspólnej strukturze danych, a każde zadanie pracuje nad częścią danych. Zadania wykonują te same operacje na danych.

- Przykład.

  Utworzyć tablicę Y mnożąc elementy tablicy X przez *pi* (*3.1415...*)
  a) na 1 procesorze
  b) na 4 procesorach
  c) dla n procesorów.

DPM, przykład

Przykład (data parallel model), 4 procesory



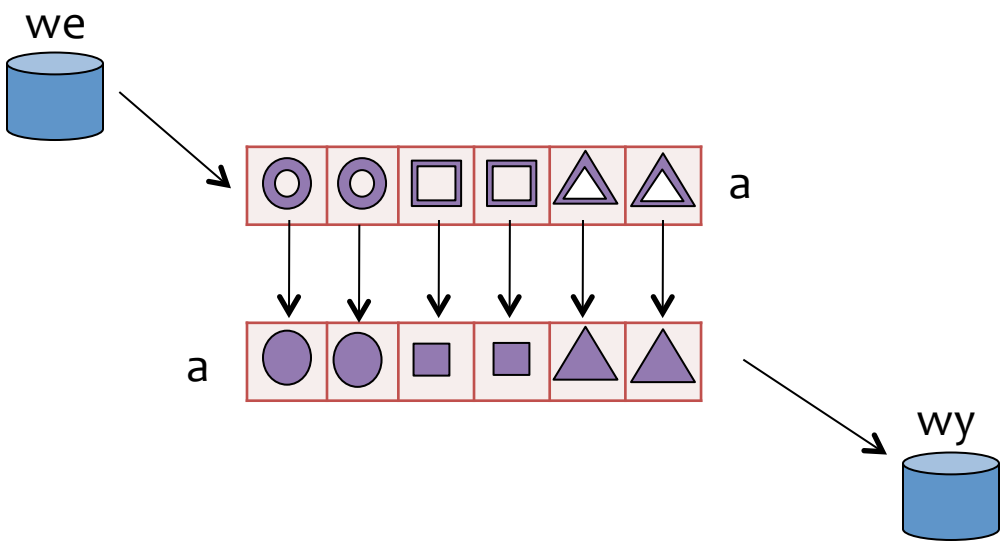
DPM, przykład

Przykład, n procesorów (zakladamy, że indeksy macierzy są 0,1,...)

```
Deklaracja m, n, p, k
Deklaracja X(m), Y(m)
Wczytaj X(m)
n = liczba_procesorów()
//ile elementów macierzy przypada na jeden proces?
k = m/n
//numer procesu lub procesora: (0,...,p)
p = moj_numer()
//ustal granice zmienności indeksów macierzy w "mojej" części
istart = p*k;  iend =(p+1)*k-1
For i=istart to iend do
   Begin
      X(i)=Pi*Y(i)
   End
```

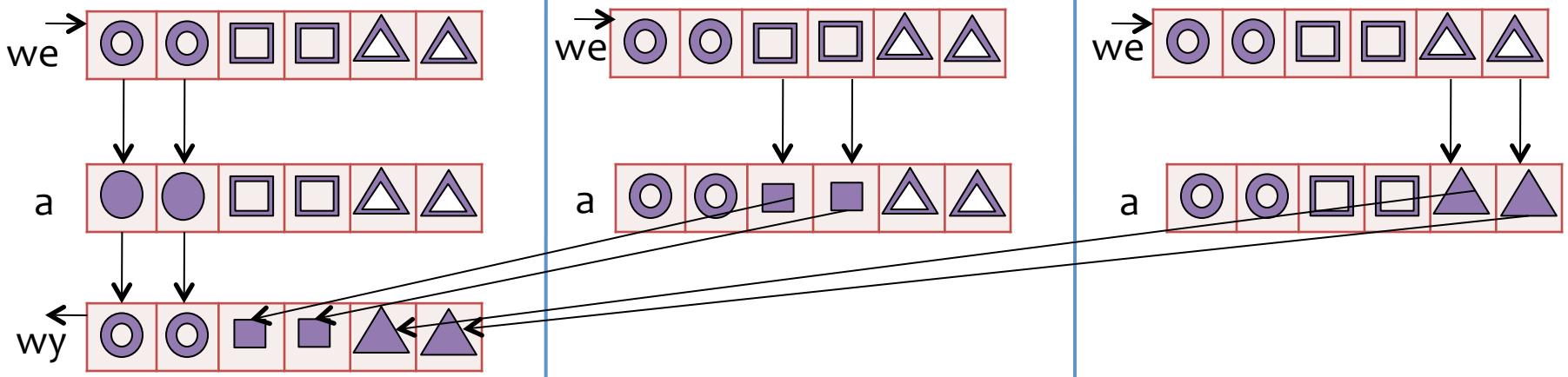
Przykład (1 procesor)

1. Czytaj a() z pliku
2. wykonaj obliczenia dla a()
   od a(i=1) do a(i=6)
3. zapisz a() do pliku



Przykład (3 procesory)

Proces 0	Proces 1	Proces 2
1. Czytaj a() z pliku	1. Czytaj a() z pliku	1. Czytaj a() z pliku
2. Pobierz „rank”	2. Pobierz „rank”	2. Pobierz „rank”
3. If rank==0 then is=1, ie=2	3. If rank==0 then is=1, ie=2	3. If rank==0 then is=1, ie=2
4. If rank==1 then is=3, ie=4	4. If rank==1 then is=3, ie=4	4. If rank==1 then is=3, ie=4
5. If rank==2 then is=5, ie=6	5. If rank==2 then is=5, ie=6	5. If rank==2 then is=5, ie=6
6. Wykonaj obliczenia od a(is) do a(ie)	6. Wykonaj obliczenia od a(is) do a(ie)	6. Wykonaj obliczenia od a(is) do a(ie)
7. Zbierz wyniki; proces 0	7. Zbierz wyniki; proces 0	7. Zbierz wyniki; proces 0
8. If rank==0 zapisz a() do pliku	8. If rank==0 zapisz a() do pliku	8. If rank==0 zapisz a() do pliku



MPI

Message Passing Interface

MPI

- MPI – message passing interface
  - Standard z lat 90-ch; KLASTRY
  - C/C++
  - Fortran

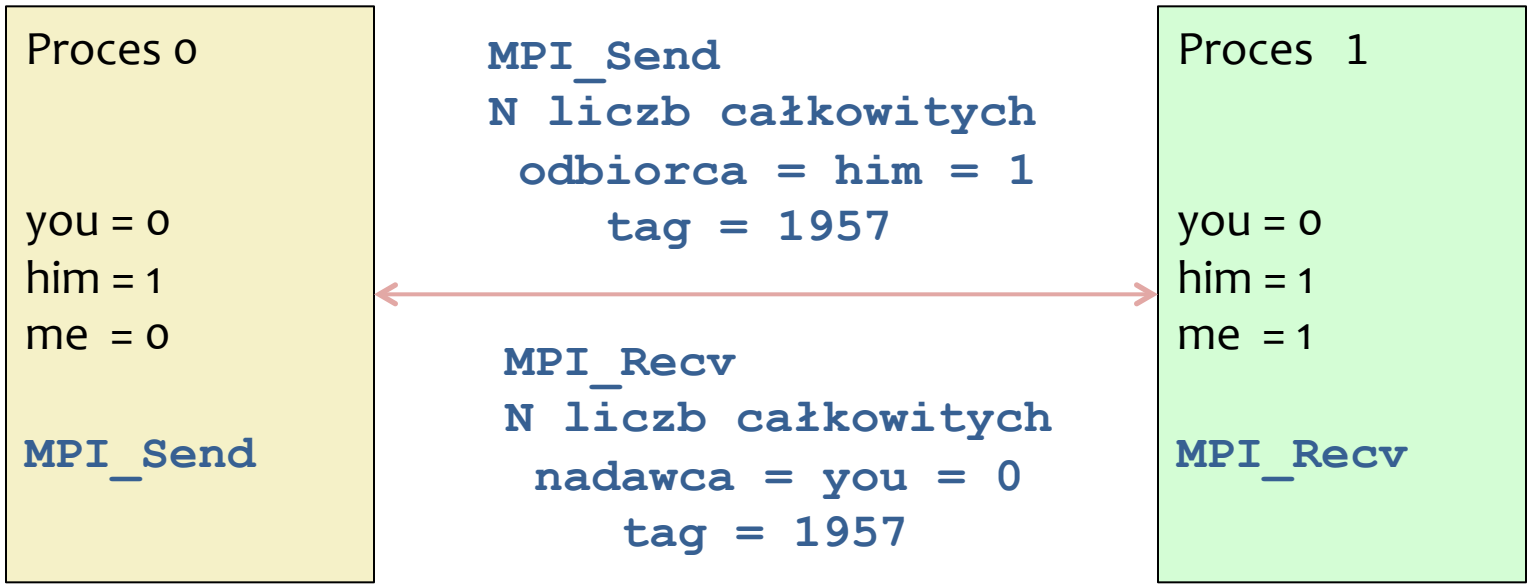
Numerologia ...☺

- Każdy „element logiczny” w komputerze posiada własną nazwę, numer: komórka pamięci, dysk, procesor, proces itd. Np. procesy, wątki mają numery 0, 1, 2, ... Proces może się dowiedzieć jaki numer posiada (Ogólniej może to być numeracja złożona: numer grupy, podgrupy, itd.)

MPI – przykład: Wyślij N liczb int


```
#include <mpi.h>                    include f le
you = 0; him = 1;
MPI_Init(&argc, &argv);             initialize MPI environment
MPI_Comm_rank(MPI_COMM_WORLD, &me); get process ID
if ( me == 0 ) {                    process 0 sends
   error_code = MPI_Send(&data_buffer, N, MPI_INT,
                         him, 1957, MPI_COMM_WORLD);
} else if ( me == 1 ) {             process 1 receives
   error_code = MPI_Recv(&data_buffer, N, MPI_INT,
                         you, 1957, MPI_COMM_WORLD,
                         MPI_STATUS_IGNORE);
}
MPI_Finalize();                     leave the MPI environment
```

MPI - komunikacja



6 FUNKCJI MPI

- Proste programy MPI można tworzyć używając tylko sześciu funkcji

6 funkcji MPI (1)

- MPI_INIT(...)
  Zainicjuj obliczenia MPI

- MPI_FINALIZE(...)
  Zakończ obliczenia MPI

6 funkcji MPI (2)

- MPI_COMM_SIZE(...)
  Określ liczbę procesów w komunikatorze

- MPI_COMM_RANK(...)
  Określ identyfikator procesu w danym komunikatorze

6 podstawowych funkcji (3)

- MPI_SEND(...)
  Wyślij wiadomość do innego procesu

- MPI_RECV(...)
  Odbierz wiadomość od innego procesu

Wymiana danych - problemy

Proces 0	Proces 1
`MPI_send(…,1,…)`	`MPI_send(…,0,…)`
`MPI_recv(…,1,…)`	`MPI_recv(…,0,…)`

Może dojść do impasu (zależy od implementacji). Jeśli komunikaty będą buforowane program może działać. (W standardzie MPI nosi to nazwę *unsafe send/recv*)

Wymiana danych - problemy

Proces 0	Proces 1
`MPI_recv(…,1,…)`	`MPI_recv(…,0,…)`
`MPI_send(…,1,…)`	`MPI_send(…,0,…)`

Zakleszczenie (Deadlock).
`MPI_recv` czeka dopóty, dopóki nie wykona się send
lub odwrotnie: czeka **`MPI_send`**!

Wymiana danych - problemy

Proces 0	Proces 1
`MPI_send(…,1,…)`	`MPI_recv(…,0,…)`
`MPI_recv(…,1,…)`	`MPI_send(…,0,…)`

OK!

Przykład. Obliczanie π

- Wyliczyć wartość π według formuły (pokazać najpierw, że formuła jest słuszna):

$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx$$

Przykład. π w C (1)


```
#include "mpi.h"
#include <math.h>
int main(int argc, char *argv[])
{
   int done = 0, n, myid, numprocs, i, rc;
   double PI25DT = 3.14159265358979323846264643;
   double mypi, pi, h, sum, x, a;
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   while (!done)  {
     if (myid == 0) {
      printf("Enter the number of intervals: (0 quits) ");
      scanf("%d",&n);
     }
     MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
     if (n == 0) break;
```

Przykład. π w C (2)


```
  h   = 1.0 / (double) n;
      sum = 0.0;
      for (i = myid + 1; i <= n; i += numprocs) {
        x = h * ((double)i - 0.5);
        sum += 4.0 / (1.0 + x*x);
      }
      mypi = h * sum;
      MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
                 MPI_COMM_WORLD);
      if (myid == 0)
        printf("pi is approximately %.16f, Error is .16f
   \n",
                pi, fabs(pi - PI25DT));
    }
    MPI_Finalize();
    return 0;
}
```

Komunikacja kolektywna

- Operacje kolektywne wywoływane są przez wszystkie procesy w komunikatorze
- **MPI_BCAST** rozdziela dane z jednego procesu (root) na wszystkie procesory w komunikatorze
- **MPI_REDUCE** zbiera dane od wszystkich procesów w komunikatorze I zwraca do jednego procesu
- W wielu algorytmach numerycznych, **SEND/RECEIVE** można zastąpić parą **BCAST/REDUCE**, upraszczając program i podnosząc efektywność

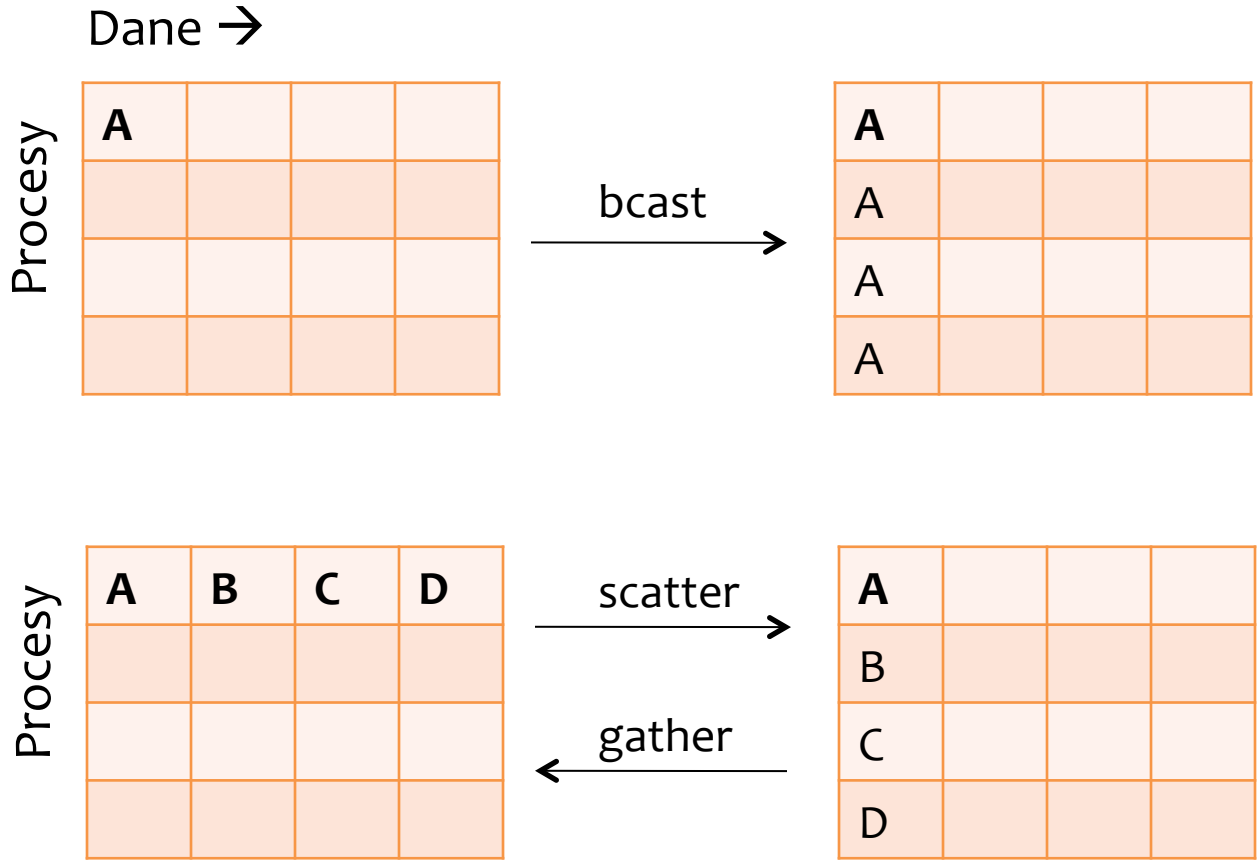
Procedury kolektywne

- **`MPI_Bcast(data, count, type, src, comm)`**
  Rozesłanie danych z **`src`** do wszystkich procesów w komunikatorze.

- **`MPI_Gather(in, count, type, out, count, type, dest, comm)`**
  Zbieranie danych ze wszystkich węzłów do węzła **`dest`**

- **`MPI_Scatter(in, count, type, out, count, type, src, comm)`**
  Wysłanie (spec) danych z węzła **`src`** do wszystkich innych

Procedury kolektywne



Procedury kolektywne

- Funkcje dodatkowe
  - `MPI_Allgather`
  - `MPI_Gatherv`
  - `MPI_Scatterv`
  - `MPI_Allgatherv`
  - `MPI_Alltoall`

Procedury redukcji

`MPI_Reduce(send, recv, count, type, op, root, comm)`

Globalna operacja “op” redukcji; wynik znajduje się w buforze *recv* procesu *root*; “op” może być zdefiniowane przez użytkownika (predefiiowane operacje MPI)

`MPI_Allreduce(send, recv, count, type, op, comm)`

Jak wyżej, lecz wynik jest przekazywany do wszystkich procesów komunikatora.

Procedury redukcji

dane

A0	B0	C0	D0
A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3

reduce →

A0#A1#A2#A3	B0+B1#B2#B3	C0#C1#C2#C3	D0#D1#D2#D3

jest jednym z operatorów redukcji

A0	B0	C0	D0
A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3

allreduce →

A0#A1#A2#A3	B0#B1#B2#B3	C0#C1#C2#C3	D0#D1#D2#D3
A0#A1#A2#A3	B0#B1#B2#B3	C0#C1#C2#C3	D0#D1#D2#D3
A0#A1#A2#A3	B0#B1#B2#B3	C0#C1#C2#C3	D0#D1#D2#D3
A0#A1#A2#A3	B0#B1#B2#B3	C0#C1#C2#C3	D0#D1#D2#D3

Procedury redukcji

- Procedury dodatkowe
  - `MPI_Reduce_scatter()`, `MPI_Scan()`
- Operacje predefiniowane
  - `sum`, `product`, `min`, `max`, ...
- Operacje definiowane przez użytkownika: patrz opis MPI
  - `MPI_Op_create()`

Komunikacja kolektywna

A			
B			
C			
D			

allgather →

A	**B**	**C**	**D**
A	B	C	D
A	B	C	D
A	B	C	D

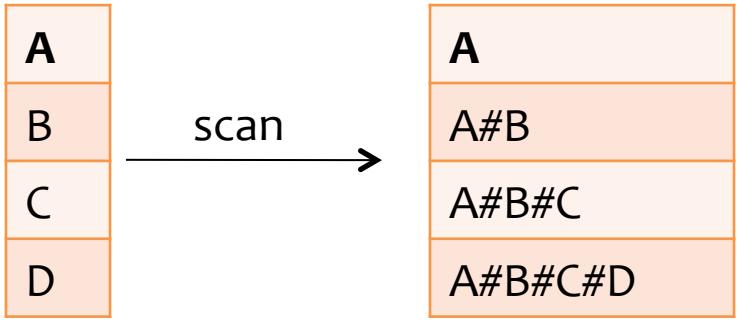
A0	**A1**	**A2**	**A3**
B0	B1	B2	B3
C0	C1	C2	C3
D0	D1	D2	D3

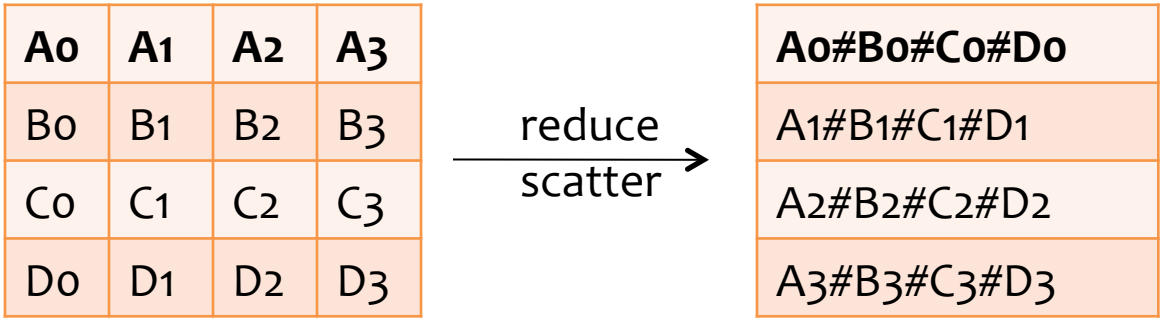
alltoall →

A0	**B0**	**C0**	**D0**
A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3

Komunikacja kolektywna

- operator





OpenMP

Podstawy

Realizacja OpenMP

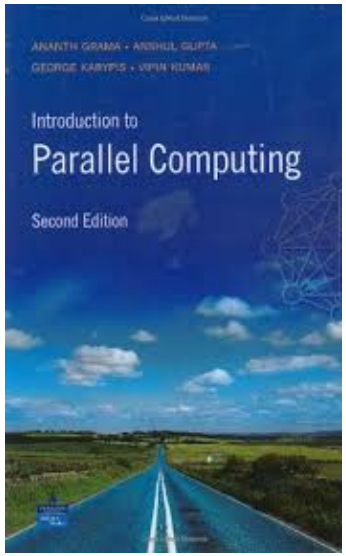
- ✓ **Wszystkie wątki mają dostęp do wspólnej pamięci i danych dzielonych**
- ✓ **Dane mogę być prywatne i wspólne**
- ✓ **Dane prywatne dostępne są tylko dla wątku właściciela**
- ✓ **Transfer danych odbywa się bardziej przejrzyście**
- ✓ **Synchronizacja jest wciąż potrzebna lecz jest przeważnie ukryta**

OpenMP™

http://www.openmp.org

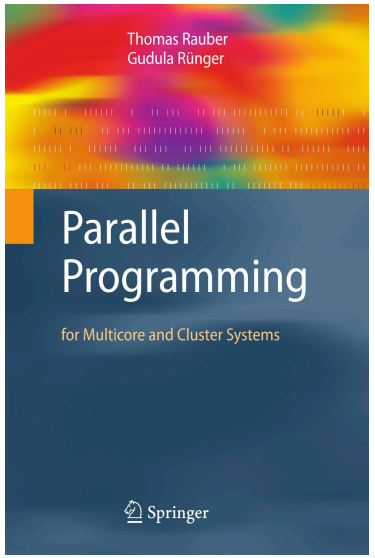
Literatura

Wprowadzenie do obliczeń równoległych. Zbigniew J. Czech. PWN, Warszawa, 2013.



Introduction to Parallel Computing.

A. Grama, G. Karypis, V. Kumar,
B. A. Gupta. Addison-Wesley, 2003.



Parallel Programming. Rauber & Ruenger. Springer, 2010.

Za tydzień ...