KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Projekt "Nowoczesne metody i techniki kształcenia w UMCS. Wzmocnienie potencjału dydaktycznego Wydziału MFil" współfinansowany jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego Program Operacyjny Kapitał Ludzki 2007-2013 Priorytet IV, Poddziałanie 4.1.1 "Wzmocnienie potencjału dydaktycznego uczelni"

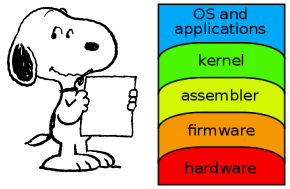
Człowiek - najlepsza inwestycja

Programowanie współbieżne... (2)

Andrzej Baran **2010/11**

LINK: http://kft.umcs.lublin.pl/baran/prir/index.html

Prawo Amdahla - powtórka



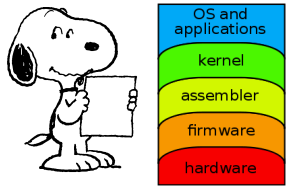
Wydajność E = S/n (na procesor). Stąd

$$S = En \leq \frac{1}{f + \frac{1-f}{n}}$$

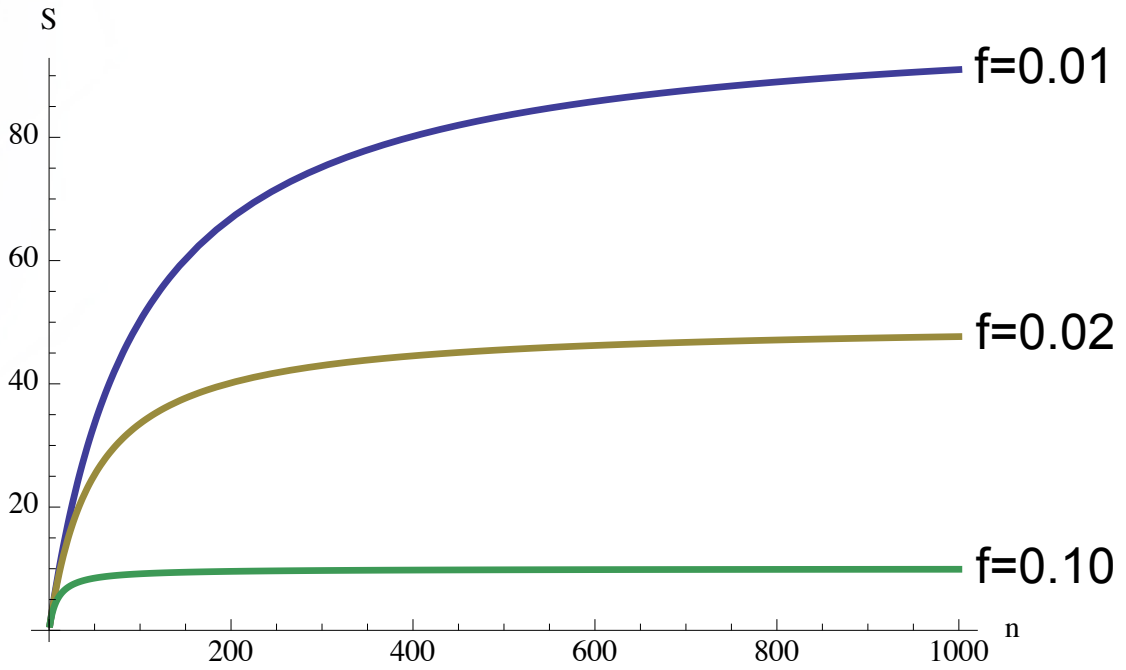
$$E \leq \frac{1}{fn + 1 - f}$$

Przy ustalonej wydajności E ułamek f obliczeń sekwencyjnych algorytmu powinien być odwrotnie proporcjonalny do liczby procesorów. (Czas obliczeń sekwencyjnych T*f ~ 1/n)

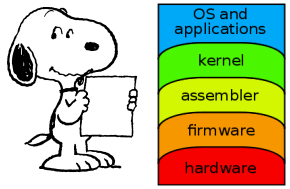
Prawo Amdahla...



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



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

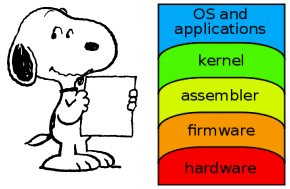
Czas obliczeń: $T_2 = (N/2-1)\delta t + \delta t + C$

gdzie C jest czasem przesyłania wyniku częściowego - czas komunikacji

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

Jeśli $c << \delta t$

Przykład. Iloczyn...



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

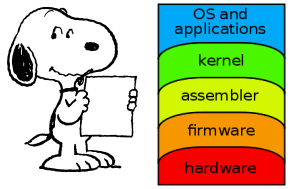
$$S_P = \frac{T_1}{T_P} = \frac{(N-1)\delta t}{q\delta t + qC}$$

(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 $\alpha = 0$ $\quad S_P = \frac{P-1}{\log_2 P} < P$

Minsky (1970)

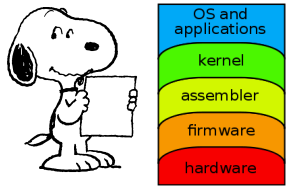


Przyspieszenie obliczeń komputera równoległego o n procesorach jest proporcjonalne do log_2(n)

Flynn (1972) – dowód dla komputerów SIMD

Dla komputerów wielkoskalowych prawo Minsky'ego nie jest spełnione.

Hennesy - Flynn



Przyspieszenie

$$S \leq \frac{n}{\log_2 n}$$

T_1 = czas pojedynczego procesora
f_k = prawdopodobieństwo jednoczesnej pracy k procesorów
T_n = czas potrzebny n proc. do wykonania zadania

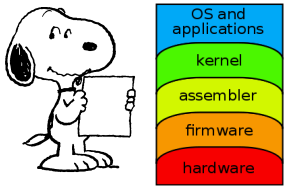
$$T_n = T_1(f_1 + f_2/2 + f_3/3 + \cdots + f_n/n)$$

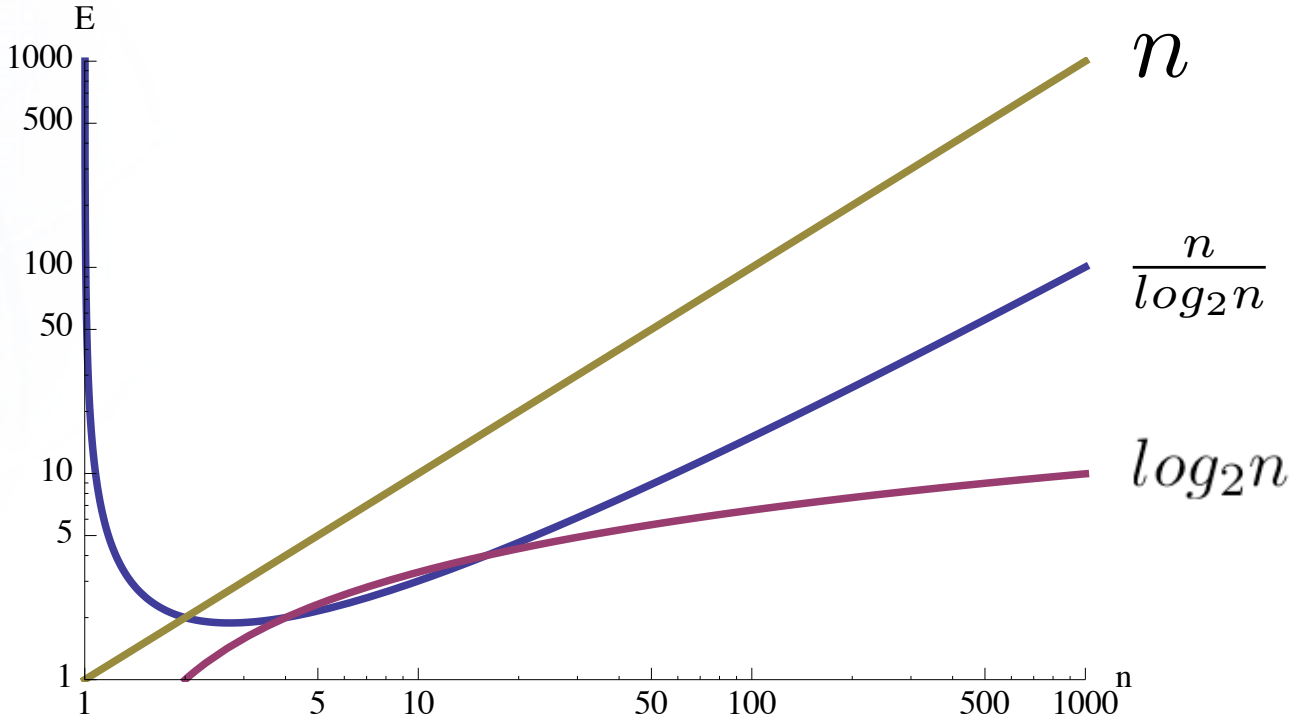
$$S = T_1/T_n = 1/(f_1 + f_2/2 + f_3/3 + \cdots + f_n/n)$$

Przyjmijmy, że $f_k = 1/n$ (dla każdego k, uproszczenie)

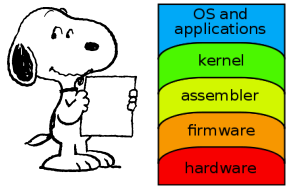
$$T_1/T_n = 1/(1/n(f_1 + f_2/2 + f_3/3 + \cdots + f_n/n)) \quad \longrightarrow \quad S \leq \frac{n}{\log_2 n}$$

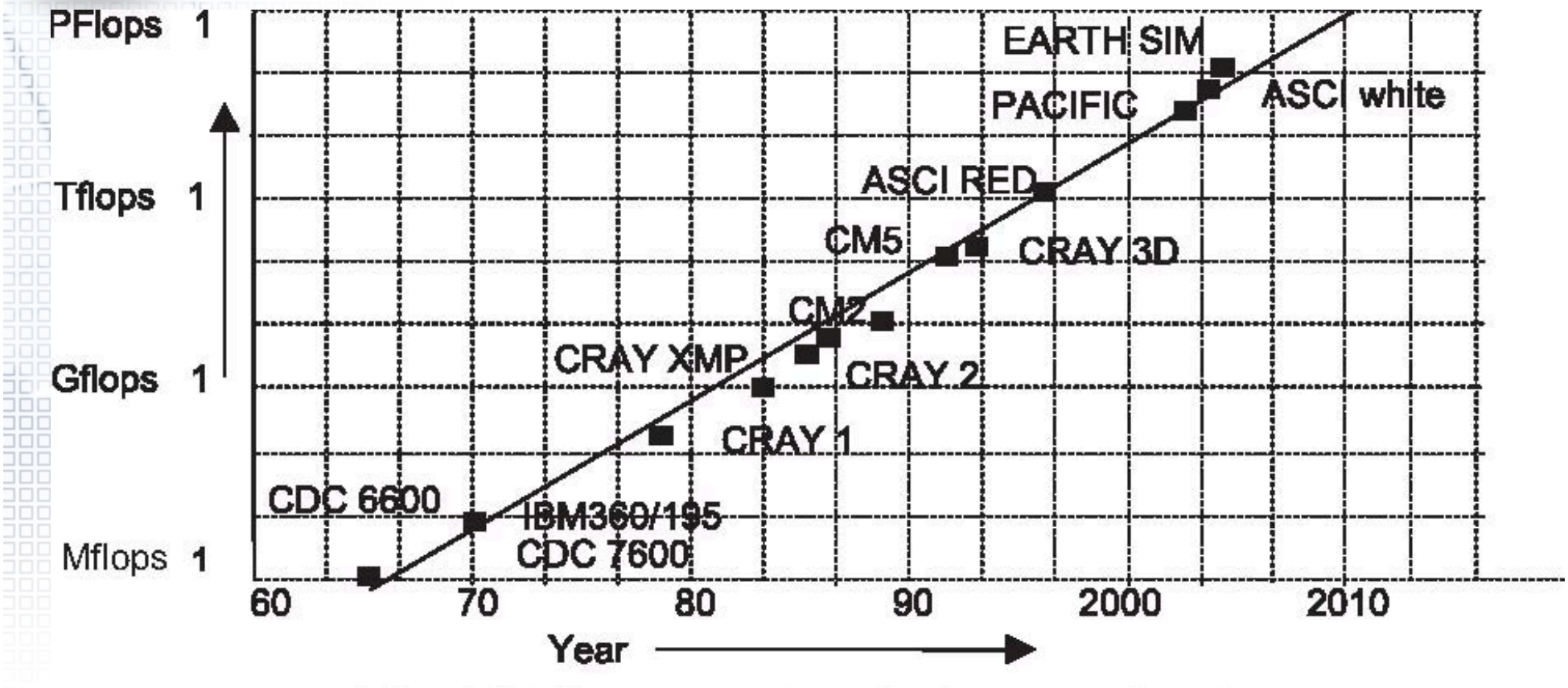
Hennesy - Flynn (cd)





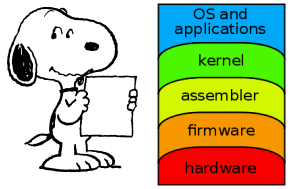
Moce komputerów. Przyszlość



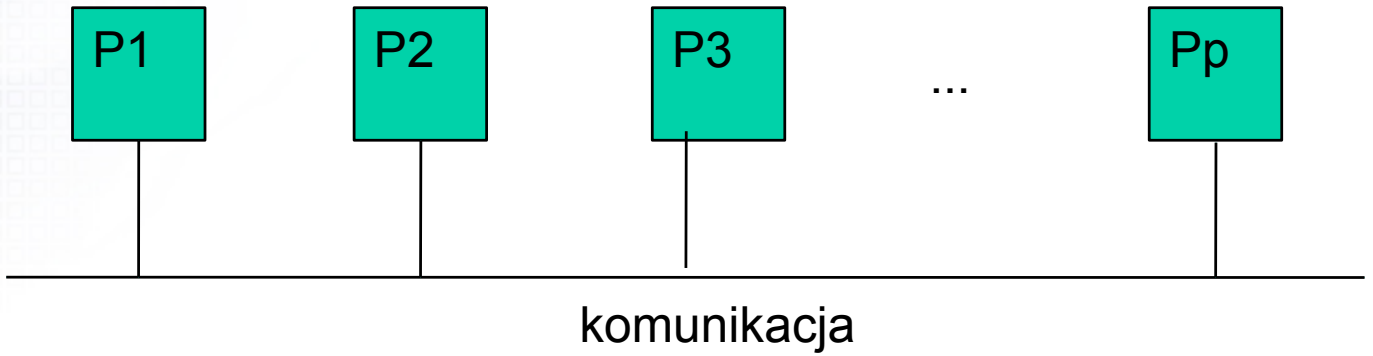


(Prawo Moore'a)

Strategia „Dziel i Zwyciężaj”

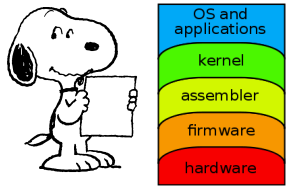


Na podstawie poprzednich przykładow: obliczenia równoległe są strategią typu „dziel i zwyciężaj” – zwycięstwo to przyspieszenie obliczeń.



CO DZIELIĆ? Dane czy „instrukcje”? Jak? Kiedy?

Modele systemów obliczania



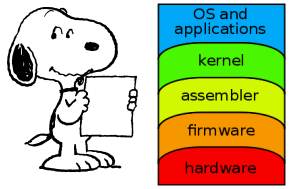
- Współdzielona pamięć (Shared Memory)
- Przekazywanie komunikatów (Message Passing)
- Wątki (Threads)
- Wspolbieżność danych (Data Parallel)

Są to abstrakcyjne modele obliczeń niezależne od sprzętu i architektury pamięci

Wybór modelu zależy od sytuacji

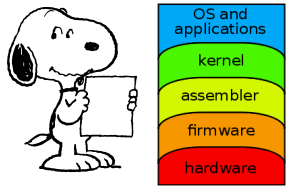
Nie istnieje model najlepszy

Modele: Shared memory (SM)



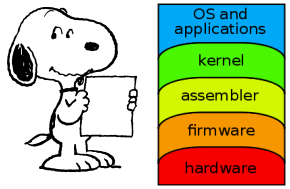
- Wspólna przestrzeń adresowa zapisywana/ odczytywana asynchronicznie
- Mechanizmy kontrolne zapisu/odczytu (semafory, blokady)
- Nie istnieje potrzeba komunikacji między poszczególnymi zadaniami

SM model - realizacja



- Posix Threads (niski poziom)
- OpenMP (standard)
- Paralelizacja automatyczna (robi to kompilator)

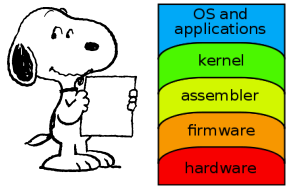
Distributed memory (DM)



Charakterystyka...

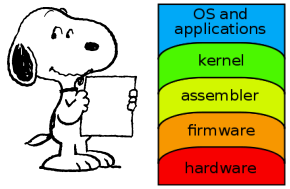
- Pamięć i we/wy są zasobami rozproszonymi
- Czas dostępu do pamięci jest niejednorodny
- Konieczna jest komunikacja p-p (wymiana informacji)
- Na procesorach pracuje N kopii systemu operacyjnego
- Komunikacja: Ethernet, Infiniband, itd.

DM model - realizacja



- Gniazda (sockets; standard; niski poziom)
- PVM – Parallel Virtual Machine ()
- MPI – Message passing Interface (standard)

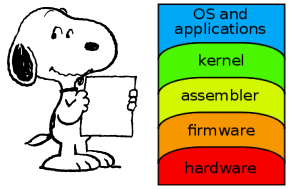
Wątki - Threads



- Wątki dzielą zasoby
- Wykonują się równolegle
- Komunikują sie poprzez wspólną pamięć
- Działają jak podprogramy (ale w tym samym czasie)

- Przykładem jest realizacje PTHREADS, OpenMP

Realizacja OpenMP

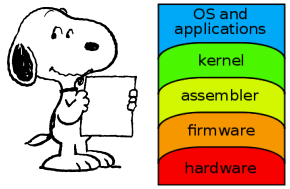


- ✓ **Wszystkie wątki mają dostęp do wspólnej pamięci i danych dzielonych**
- ✓ **Dane mogę być prywatne i wspólne**
- ✓ **Dane prywatne dostępne są tylko dla wątku właściciela**
- ✓ **Transfer danych odbywa się bardziej przejrzyście**
- ✓ **Synchronizacja jest wciąż potrzebna lecz jest przeważnie ukryta**

OpenMP™

http://www.openmp.org

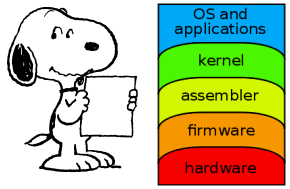
Realizacja OpenMP (wątki)



Forum 1992

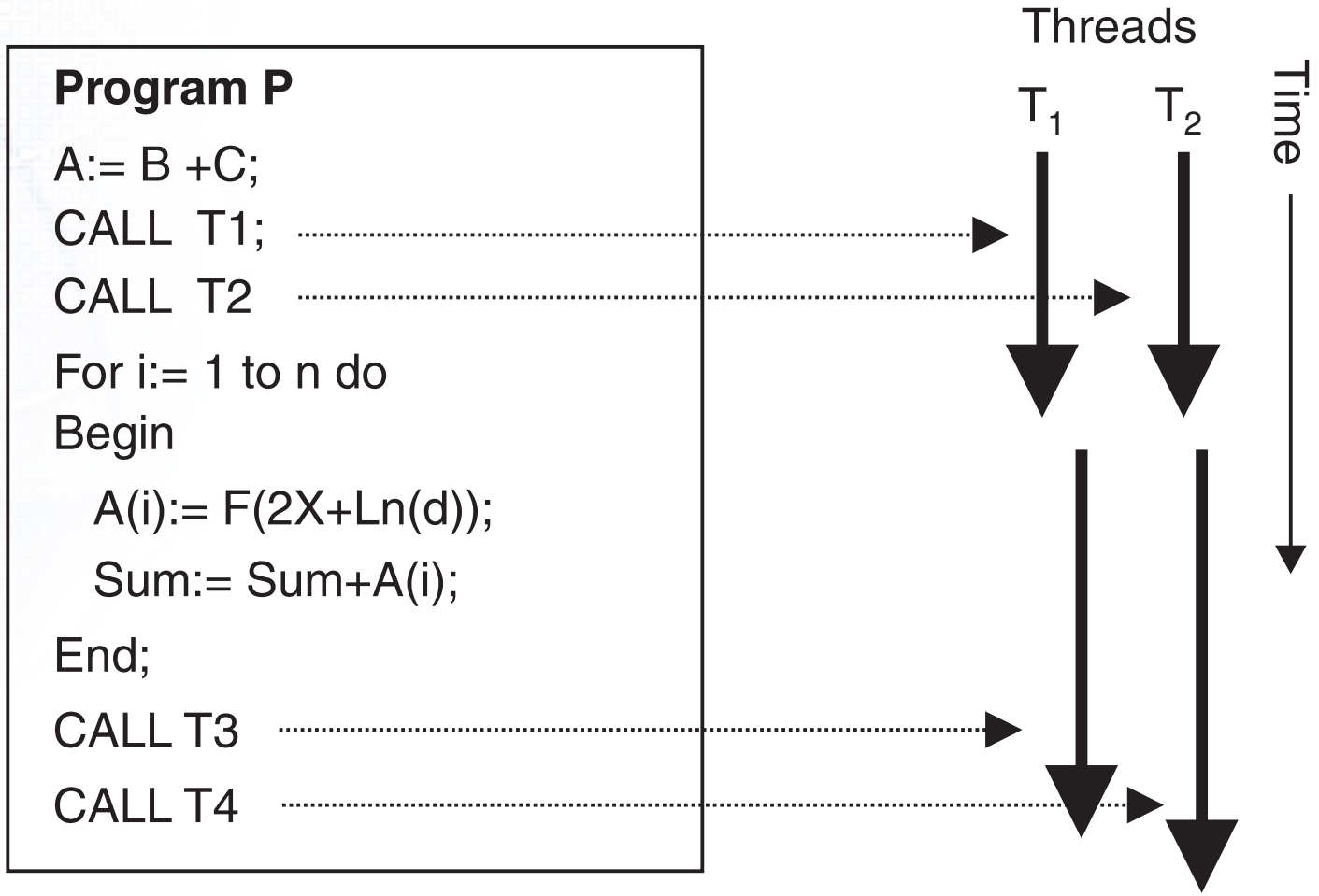
- OpenMP = Application Program Interface
- Przenośna, skalowalna
- Wspomaga C/C++ i Fortran
- Działa we wszystkich architekturach systemów
- Posiada różne konstrukcje i dyrektywy określające obszary równoległe, podział pracy, synchronizację i dane

Wątki

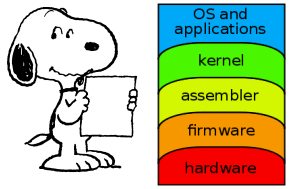


Program P

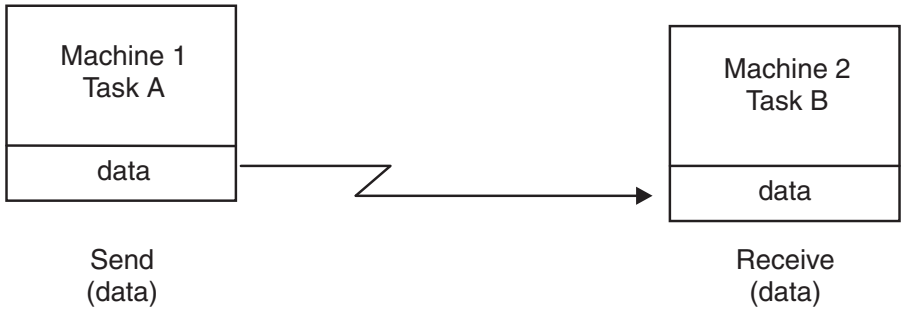
```
A:= B +C;
CALL  T1;
CALL  T2
For i:= 1 to n do
Begin
   A(i):= F(2X+Ln(d));
   Sum:= Sum+A(i);
End;
CALL T3
CALL T4
```



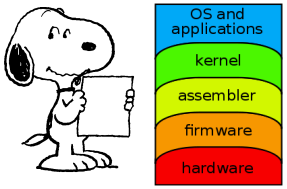
Message Passing Interface (MPI)

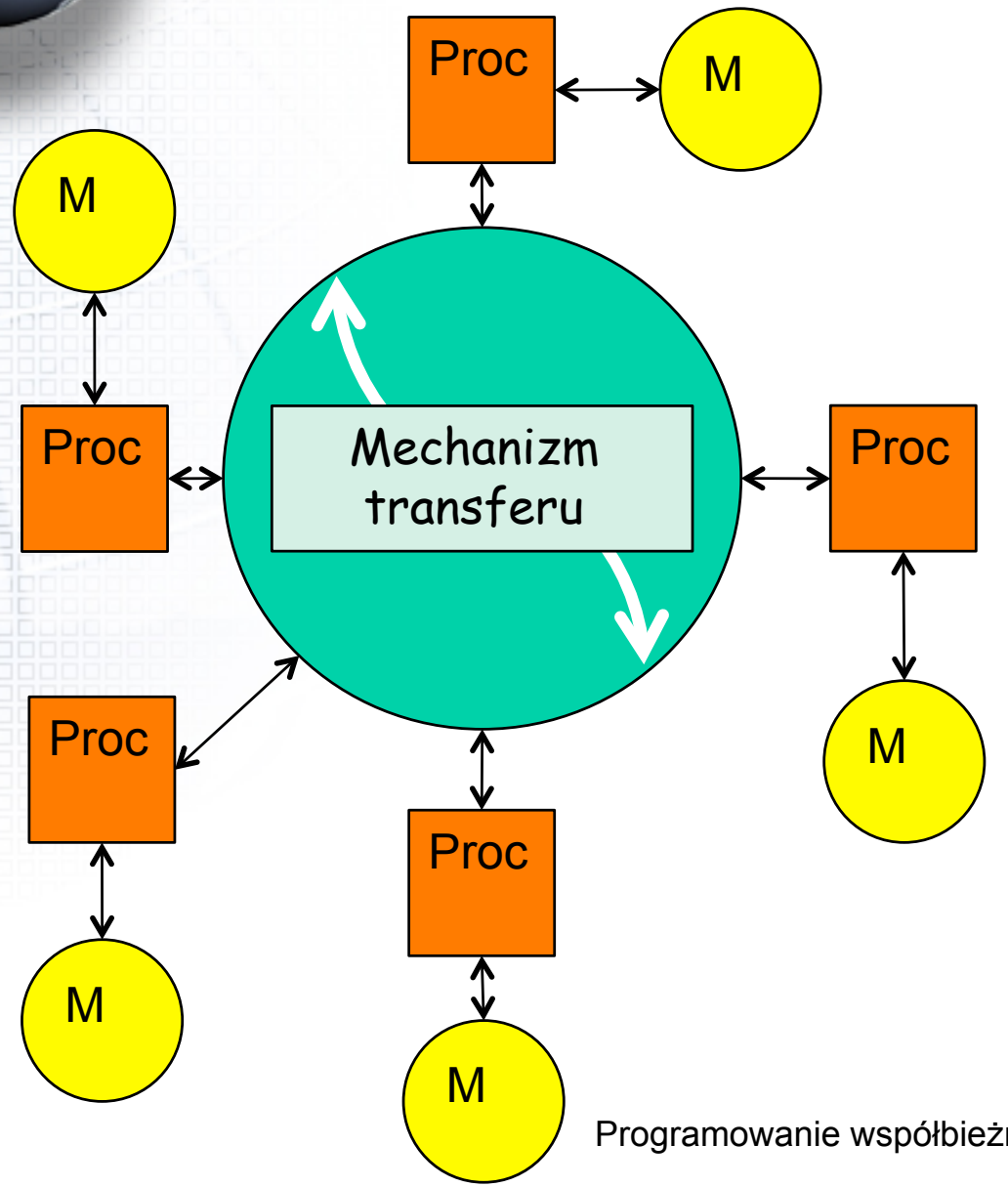


- MPI – 1994
- MPI-2 – 1996
- Standard
- Model typu
  WYŚLIJ ←→ ODBIERZ (dane, komunikat)
  ang. Send(data) ←→ Recive(data)



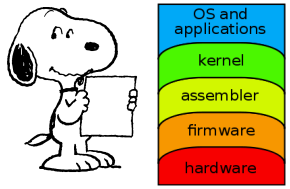
MPI - cechy





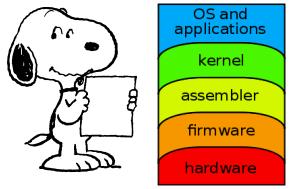
- ✓ **Każdy procesor ma dostęp tylko do własnej pamięci (dane prywatne)**
- ✓ **Transfer danych i operacje synchronizacji programuje się jawnie**
- ✓ **Wszystkie dane są prywatne**
- ✓ **Współdzielenie danych = wymiena buforów**

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, procesu itd)**

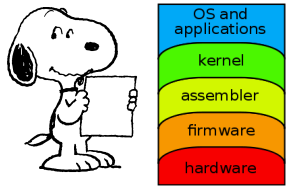
MPI – przykład: Wyślij N liczb int




```
#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
```

Tutorial IWOMP 2010 – CCS Un. of Tsukuba, June 14, 2010

MPI - komunikacja



Każdy proces ma swój numer: 0, 1, ...

Proces 0

```
you = 1
him = 0
me  = 0

MPI_Send
```

```
N liczb całkowitych
odbiorca = you = 1
tag = 1957
```

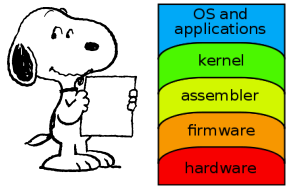
```
N liczb całkowitych
nadawca = him = 0
tag = 1957
```

Process 1

```
you = 1
him = 0
me  = 1

MPI_Recv
```

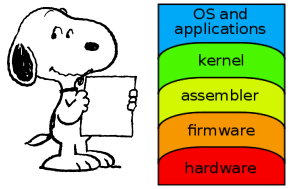
MPI za/przeciw



- **Zalety MPI**
  - **Elastyczność – może działać na dowolnej wielkości klastrze**
  - **Bezpośredniość – w programie wstawia się wywołania proc. MPI**
  - **Dostępność – istnieje kilka wdrożeń**
  - **Powszechność – model bardzo popularny**

- **Wady MPI**
  - **Reorganizacja kodu – sporo roboty**
  - **Łatwo o błędy – dużo detali do sprawdzania**
  - **Trudności w odnajdywaniu błędow – detale**
  - **Wymaga więcej zasobów – potrzebna większa pamięć**
  - **Specjalnej uwagi wymaga we/wy**

Data Parallel Model (DPM)



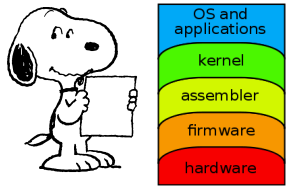
Operacje na zbiorach danych (np na tablicach)

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

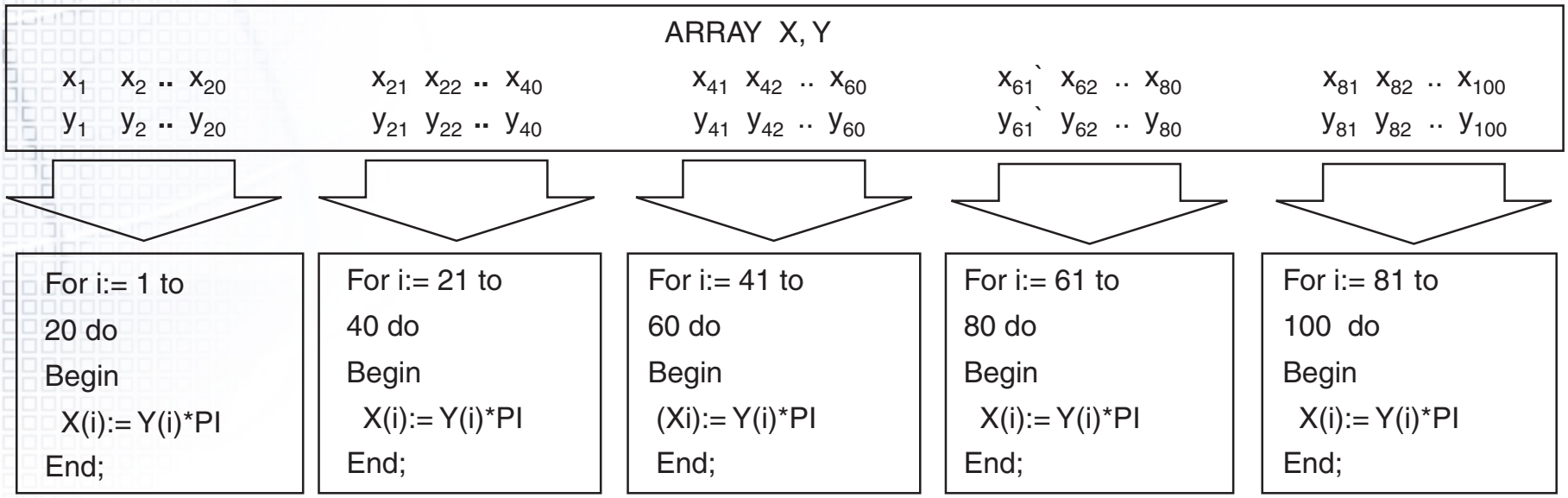
Przykład.

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

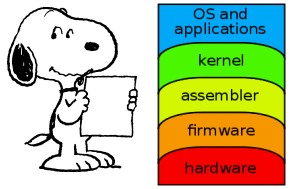
DPM, przykład



Przykład (data parallel model), 4 procesory



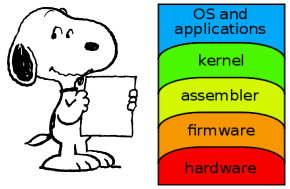
DPM, przykład



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
```

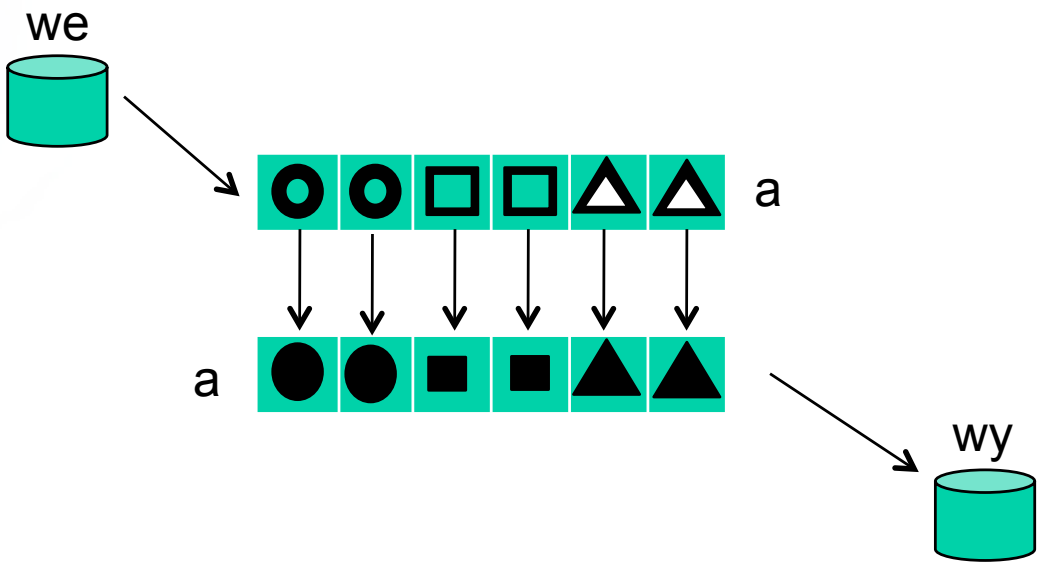
Przykład (1 procesor)



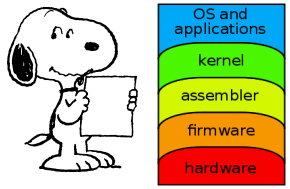
1. Czytaj a() z pliku

2. wykonaj obliczenia dla a()
   od a(i=1) do a(i=6)

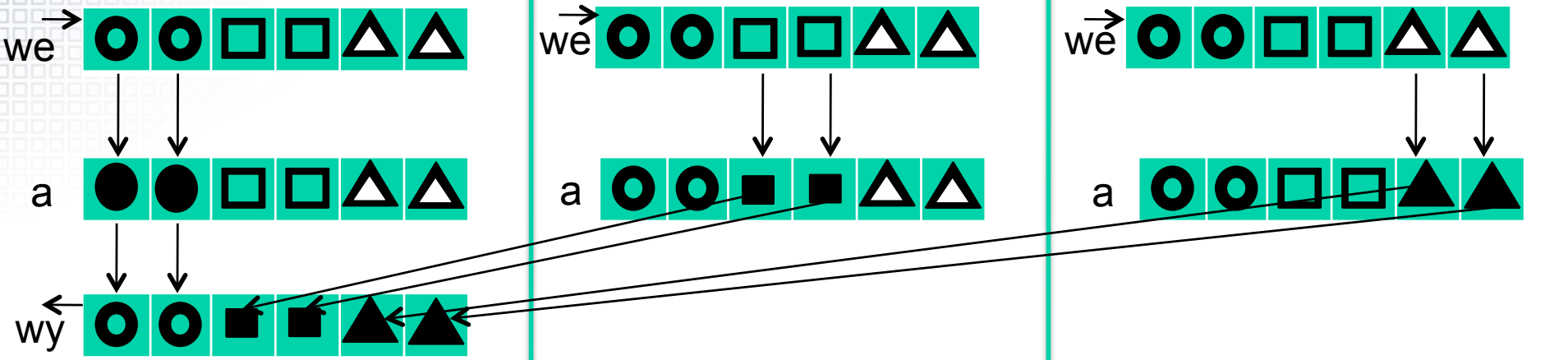
3. zapisz a() do pliku



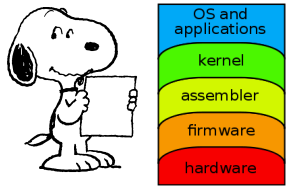
Przykład (3 procesory)



Proces 0	Proces 1	Proces 2
1. Czytaj a() z pliku 2. Pobierz „rank”	1. Czytaj a() z pliku 2. Pobierz „rank”	1. Czytaj a() z pliku 2. Pobierz „rank”
3. If rank==0 then is=1, ie=2 4. If rank==1 then is=3, ie=4 5. If rank==2 then is=5, ie=6	3. If rank==0 then is=1, ie=2 4. If rank==1 then is=3, ie=4 5. If rank==2 then is=5, ie=6	3. If rank==0 then is=1, ie=2 4. If rank==1 then is=3, ie=4 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

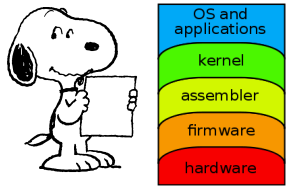


Algorytmy współbieżne (AW)



- AW to algorytmy, które wykonują się na równoległych maszynach
- Miarą wydajności AW jest szybkość z jaką można wykonać zadanie na tylu procesorach ile potrzeba
- Ważny jest czas komunikacji między procesorami

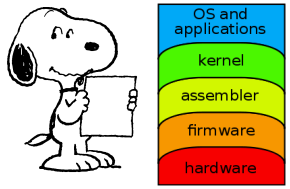
Algorytmy współbieżne (AW) cd.



AW dla danego problemu polega na

- Wykryciu i wykorzystaniu równoległości w danym algorytmie sekwencyjnym
- Stworzeniu nowego algorytmu równoległego
- Adaptacji istniejącego algorytmu równoległego dla zadanego problemu

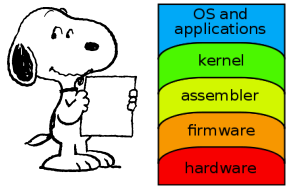
Algorytmy współbieżne (AW) cd.



Główne zastosowania algorytmów równoległych

- Problemy programowania liniowego
- Sortowanie
- Szybka transformacja Fouriera (FFT)
- Przeszukiwanie i grafy
- Operacje macierzowe

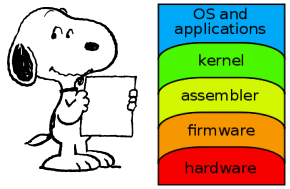
Obliczenia rozproszone



Obliczenia rozproszone są z założenia związane z fizycznym rozproszeniem podzadań – np. programy pracujące w bankowości muszą działać na pewnym obszarze, w automatach, stanowiskach kasowych, bankomatach itp i komunikować się z głównym systemem bankowym.

Programy te działają dla wygody użytkowników (poprzez np. sieci komputerowe)

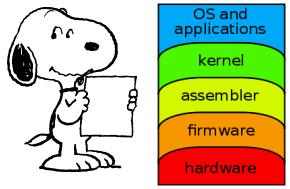
Obliczenia rozproszone (cd)



Problemy:

- Czas (lokalny), synchronizacja zegarów.
- Niepełne informacje o innych węzłach
- Przekrywanie się informacji (wielu pracujących nad tym samym)

Klastry komputerowe



- Klastry komputerowe = razem pracujące procesory, komputery ...
- **Beowulf** – rodzaj klastra komputerowego zbudowanego przy założeniu uzyskania maksymalnej mocy obliczeniowej jak najmniejszym kosztem. Najczęściej budowany ze zwykłych, masowo produkowanych komputerów klasy PC, połączonych siecią Ethernet. Komputery tworzące taki klaster pracują zazwyczaj pod Linuksem, a do obliczeń równoległych używa się bibliotek MPI, LAM lub PVM.
- Pierwszy: NASA, 1994, Donald Becker