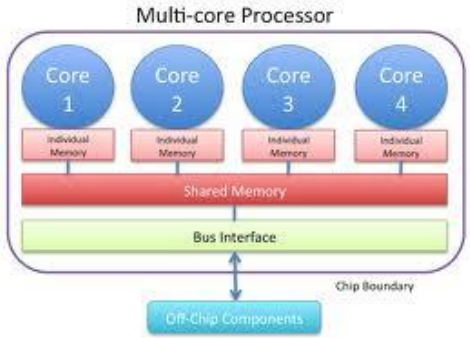
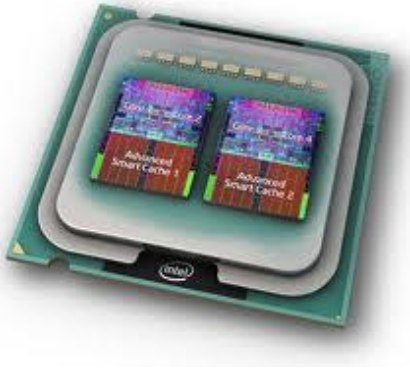
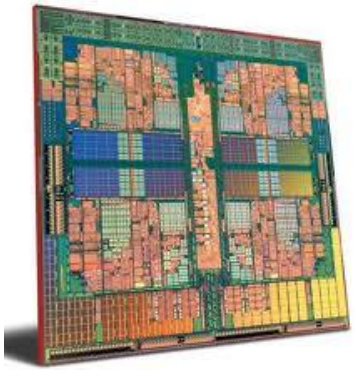
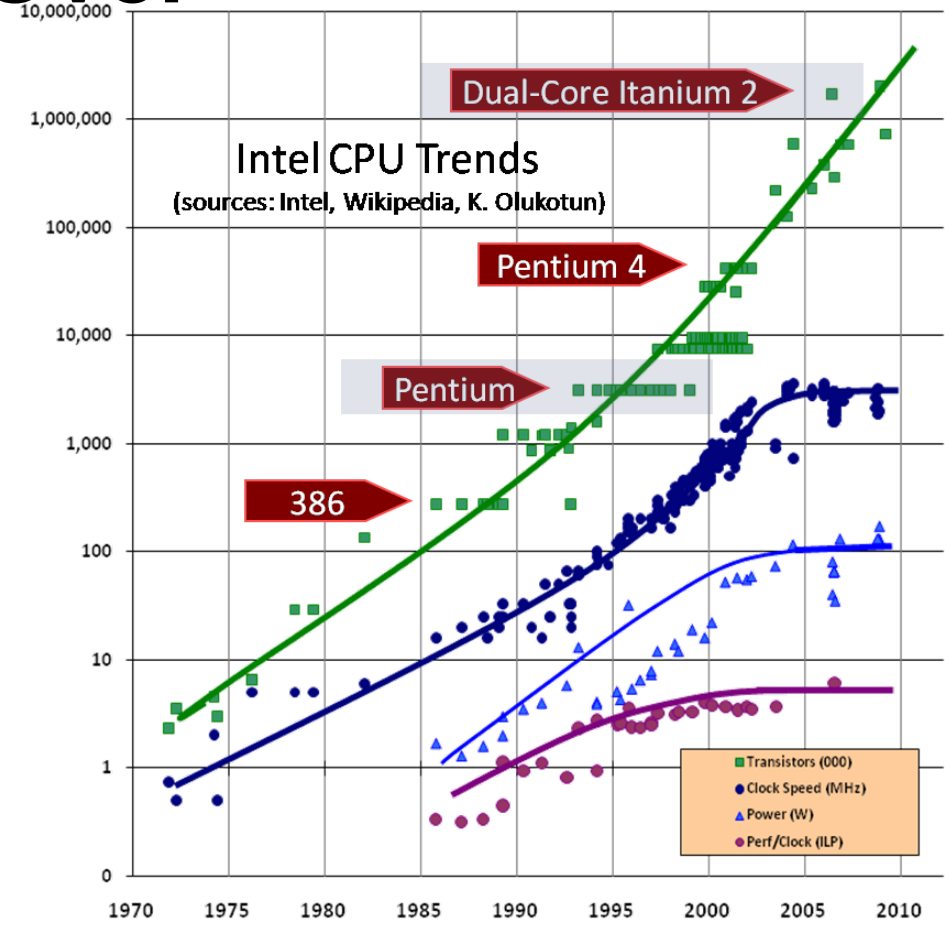
MPI
Message Pasing Interface

“The Free Lunch Is Over”









Programowanie równoległe

Taksonomia Flynna:

- **SISD** (Single Instruction, Single Data) - komputery skalarne
- **SIMD** (Single Instruction, Multiple Data) - wiele strumieni danych przez jeden program - tzw. komputery wektorowe. SIMT - "Single Instruction, Multiple Threads"
- **MISD** (Multiple Instruction, Single Data)
- **MIMD** (Multiple Instruction, Multiple Data) - wiele programów, każdy przetwarza własne dane: klastry.

MPI Message Pasing Interface

Message Passing Interface (**MPI**) (z ang. *Interfejs Transmisji Wiadomości*) – protokół komunikacyjny będący standardem przesyłania komunikatów pomiędzy procesami programów równoległych działających na jednym lub więcej komputerach. Interfejs ten wraz z protokołem oraz semantyką specyfikuje, jak jego elementy winny się zachowywać w dowolnej implementacji.

Celami MPI są wysoka jakość, skalowalność oraz przenośność.

Standard MPI implementowany w postaci bibliotek, dostępnych z C,C++, Ada, Fortran.

Implementacje: MPICH, PVM, OpenMP

mpi4py

MPI for Python package.

MPI for Python provides bindings of the *Message Passing Interface* (MPI) standard.

MPI - hello world!

Moduł z interfejsem do MPI

Komunikator - objekt pośredniczący w wymianie danych między procesami

size - liczba procesów w “świecie”

rank - unikalny numer procesu

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
print "OK, rank= ",rank,size
```

Uruchamiamy na przykład
3 kopie:

```
$ mpiexec  -n 3 python  hello.py
```

Przkazywanie komunikatów: MPI

Komunikacja **point-to-point**:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
   data = {'a': 7, 'b': 3.14}
   comm.send(data, dest=1)

elif rank == 1:
   data = comm.recv(source=0)
   print "OK, rank= ",rank,"dane: ",data
```

Pierwszy proces wysyła dane:

Drugi odbiera i drukuje

mpi4py:

Dwa rodzaje funkcji:

```
comm.send(a, dest=1)
comm.recv(a, source=0)
```

Małe litery: przesyłanie objektów python-owych (serializacja, pickling), wolniejsze

```
comm.Send(a, dest=1)
comm.Recv(a, source=0)
```

Duże litery: przesyłanie buforów danych, np. ciągła tablica numpy. szybsze

Przkazywanie komunikatów: MPI

Komunikacja **point-to-point**, przesyłanie tablicy numpy

```
import numpy as np
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

a = np.zeros((2,2))
if rank == 0:
   a[:] = 2
   comm.send(a[0,:], dest=1)

elif rank == 1:
   a[0,:] = comm.recv(source=0)
   print "OK,",np.sum(a)
```

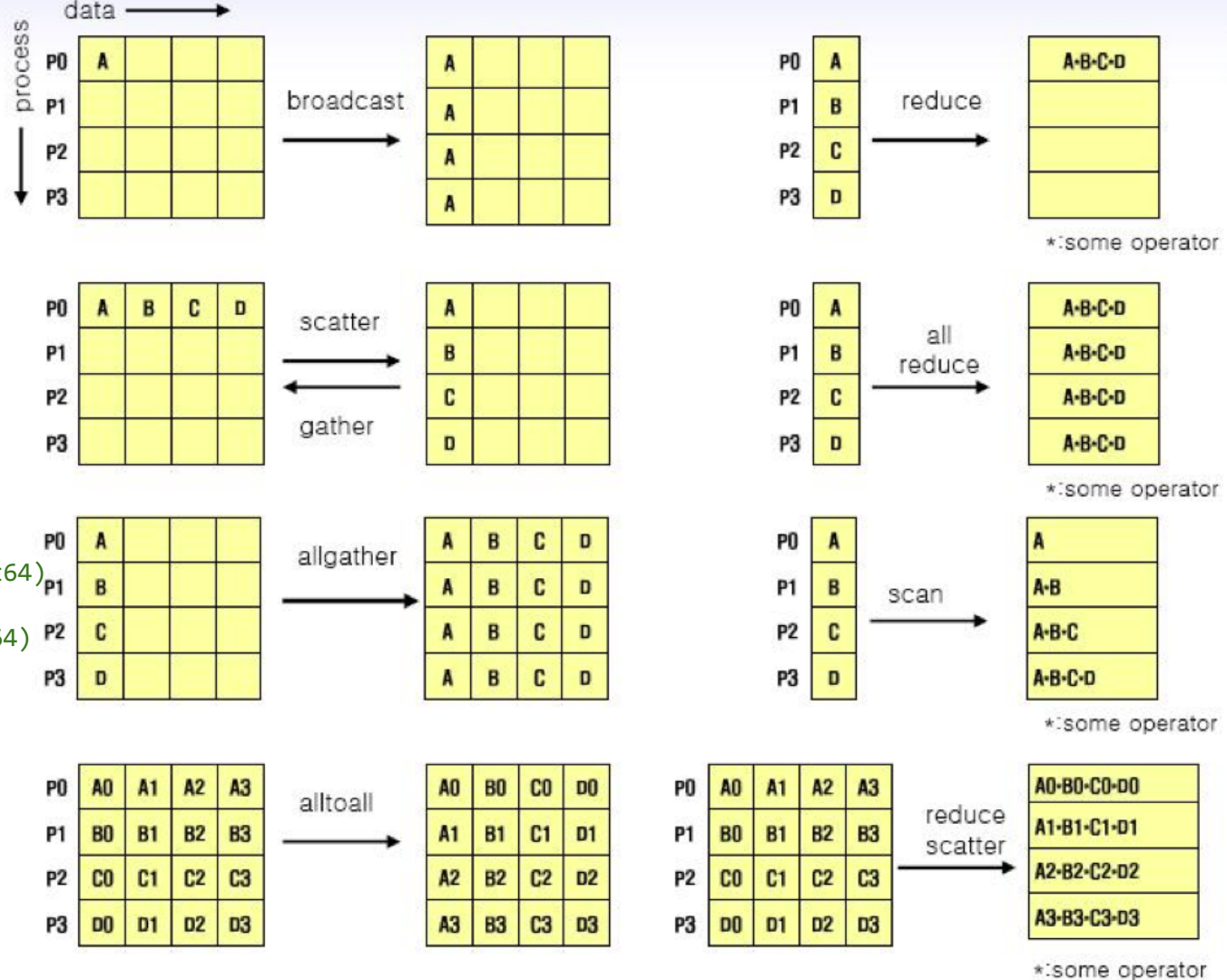
Wysyłamy tylko pierwszy rząd

Odbieramy też tylko pierwszy rząd

Komunikacja kolektywna

przykład:

```
from mpi4py import MPI
import numpy as np
comm = MPI.COMM_WORLD

N_local = 5
N = N_local*comm.size

if comm.rank == 0:
    A = np.arange(N,dtype=np.float64)
else:
    A = np.empty(N,dtype=np.float64)

A_local = np.empty(N_local)

comm.Scatter( A, A_local )
A_local = A_local+100.0
comm.Gather( A_local, A)

if comm.rank==0:
    print A
```



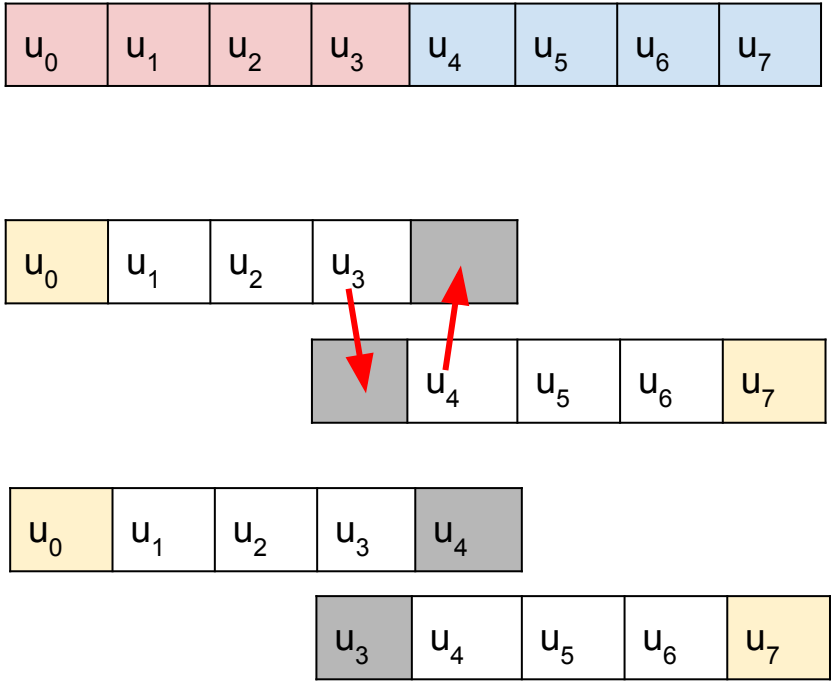
Projekt 1: całkowanie 1d

Napisać program całkujący równolegle funkcję f(x) w przedziale (a,b) wykorzystujący N procesów i komunikacje MPI.

Wykorzystać metodę “Reduce”:

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.rank

A = np.random.rand(1)
suma = np.zeros(1)
comm.Reduce(A,suma, op = MPI.SUM)
if rank==0:
    print suma
```

Projekt 2: równanie dyfuzji 1d



Napisać program rozwiązujący na 2 procesach jednowymiarowe równanie dyfuzji. Każdy process

Wykorzystać metody:

- Send,Recv - do wymiany brzegu
- Gather - do połaczenia wyniku

Projekt 3:
Monte-Carlo

Napisać program obliczający liczbę pi metodą Monte Carlo korzystając z N procesów komunikujących się za pomocą MPI

Projekt 4:
Fraktal

Napisać program generujący zbiór Julii na N procesach. Użyć MPI do zebrania danych.

$$z_0 = p$$

$$z_{n+1} = z_n^2 + c$$

$$f(z) = z - \frac{f(z)}{f'(z)} = \frac{1 + (n-1)z^n}{nz^{n-1}}.$$