Mpi4Py

Terdapat 2 pendekatan dalam paralel programming yaitu:

a) Shared Memory

Pada pendekatan ini, prosesor berbagi akses pada memory yang sama. OpenMP adalah salah satu contohnya. Tidak ada OpenMP untuk python.

b) Distributed memory

Setiap prosesor (CPU atau core) mengakses memori dan cpunya sendiri-sendiri. Jika prosesor ingin mengakses data yang berada pada memori yang dimiliki oleh prosesor lain, dua prosesor tersebut harus melakukan pertukaran pesan ("message"). Contohnya model ini adalah MPI (message passing interface). Pada python hal ini dilakukan dengan modul mpi4py.

Dasar Pemrograman mpi4py

1. Hello world

```
1  from mpi4py import MPI
2
3  comm = MPI.COMM_WORLD
4  rank = comm.Get_rank()
5  size = comm.Get_size()
6  print "hello world from process %d/%d\n" %(rank,size)
```

Line 1: import MPI
Line 2: buat object comm dari class MPI
Line 3: memperoleh ranking dari setiap proses
Line 4: memperoleh ukuran (banyaknya proses dalam communicator)
Line 5: print ranking dan ukuran

Dua pertanyaan penting yang muncul dalam paralel programming adalah: berapa banyak proses yang berpartisipasi dalam komputasi? dan proses manakah saya? MPI menggunakan size dan rank untuk menjawab pertanyaan diatas. **Size** menunjukkan banyaknya proses keseluruhan dan **rank** menunjukkan angka 0 s.d size-1, sebagai identifikasi proses yang sedang memanggil.

a) proses dapat dikelompokkan menjadi *groups*
b) setiap message dikirim dalam *context*
c) *communicator = group+context*
d) Proses diidentifikasi dengan menggunakan *ranks*
e) *COMM_WORLD* adalah group default

Untuk mengeksekusi kode di atas secara paralel digunaan program mpiexec.exe
4 program berjalan secara parallel, masing-masing mengeksekusi hello_mpi.py
$ mpiexec -n 4 python hello_mpi.py
hello world from process 0/4
hello world from process 2/4
hello world from process 1/4
hello world from process 3/4

Jika dijalankan tanpa mpiexec hanya dijalankan 1 proses
$ python hello_mpi.py
hello world from process 0/1

2. Point-to-point communication

```
1  from mpi4py import MPI
2
3  comm = MPI.COMM_WORLD
4  rank = comm.Get_rank()
5  size = comm.Get_size()
6  if rank == 0:
7      for i in range(1, size):
8          sendMsg = "Hello, Rank %d\n" %i
9          comm.send(sendMsg, dest=i)
10 else:
11     recvMsg = comm.recv(source=0)
12     print recvMsg
```

Line 1: import MPI
Line 3: buat object comm dari class MPI
Line 4: memperoleh ranking dari setiap proses
Line 5: memperoleh ukuran (banyaknya proses)
Line 6-9: jika saya adalah proses 0 (rank 0) maka saya akan mengirim pesan ke proses 1, 2, 3
Line 10-12: jika saya bukan proses 0 maka saya akan menerima pesan dan menampilkannya

MPI BASIC Send (**BLOCKING**)

a) Message akan dikirim ke **dest** (menggunakan rank).
b) Ketika fungsi ini return maka data telah dikirim ke sistem. Message mungkin sudah diterima atau belum diterima oleh target.

MPI BASIC Recv (**BLOCKING**)

a) proses akan menunggu hingga message diterima.
b) source menunjukkan proses pengirim message.

Menjalankan 4 proses secara parallel dengan rangking 0-3. Rangking 0 akan menampilkan hasilnya. Rangking 1-3 mengirim pesan.
$ mpiexec -n 4 python p2p_mpi.py

Hello, Rank 1
Hello, Rank 2
Hello, Rank 3

3. Broadcast-collective communication

```
1  from mpi4py import MPI
2
3  comm = MPI.COMM_WORLD
4  rank = comm.Get_rank()
5  size = comm.Get_size()
6  if rank == 0:
7      comm.bcast("Hello from Rank 0",root=0)
8  else:
9      msg=comm.bcast("",root=0)
10     print "Rank %d received: %s" %(rank, msg)
```

Line 6-7: jika saya proses dengan rank 0 maka saya membroadcast pesan “Hello from rank 0”
Line 90-10: jika saya bukan rank 0 maka saya menerima dan menampilkan pesan

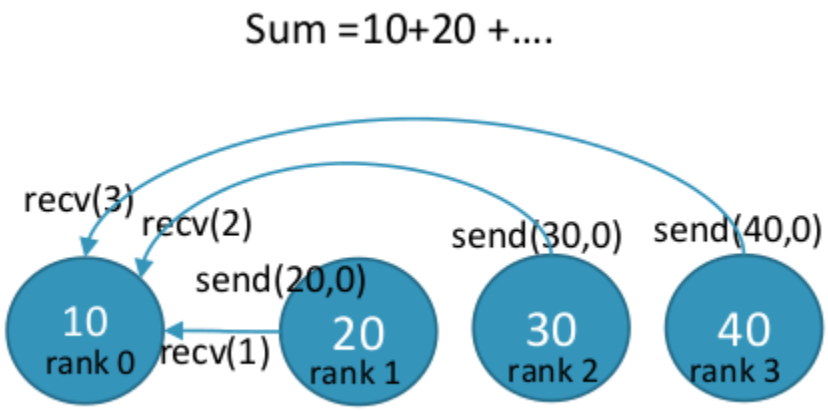
Menjalankan 4 proses dengan ranking 0-3. Rank 0 broadcast pesan ke semua rangking.
$mpiexec.exe -n 4 python .\bcast_mpi.py
Rank 2 received: Hello from Rank 0
Rank 3 received: Hello from Rank 0
Rank 1 received: Hello from Rank 0

4. Sum-p2p

```
1  from mpi4py import MPI
2
3  comm = MPI.COMM_WORLD
4  rank=comm.Get_rank()
5  size=comm.Get_size()
6  val = (rank+1)*10
7  print "Rank %d has value %d" %(rank, val)
8  if rank == 0:
9      sum = val
10     for i in range(1,size):
11         sum += comm.recv(source=i)
12     print "Rank 0 worked out the total %d" %(sum)
13 else:
14     comm.send(val, dest=0)
```

Line 6: val adalah angka yang akan dijumlahkan. Val tergantung dari ranking setiap proses
Line 7: menampilkan proses dan val yang dimilikinya

Line 8-12: proses rank 0 bertugas mengumpulkan val, menjumlahkannya totalnya kemudian menampilkannya
Line 13: proses lain hanya bertugas mengirim



Catatan: rank 0 menerima dari rank 1, 2, 3 secara berurutan (sequential). Setiap proses akan segera menjalankan “send” pada saat proses dieksekusi akan tetapi “send” akan benar-benar selesai ketika “recv” yang bersesuaian dipanggil oleh rank 0. **Cara sekuensial ini harus dihindari.**

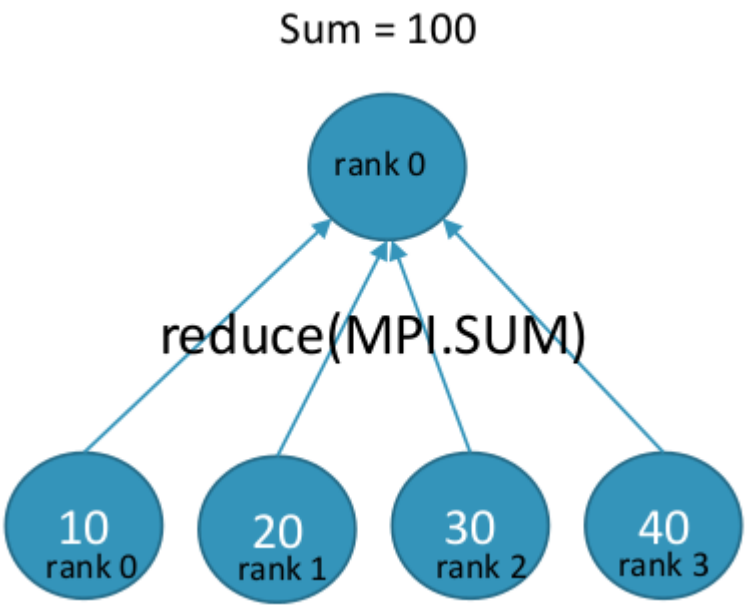
Menjalankan 4 buah proses dengan ranking 0-3. Rank 1 mengirim 20, rank 2 mengirim 30, rank 3 mengirim 40. Pada rank 0 telah ada nilai 10, rank 0 bertugas menjumlahkan semua pesan yang diterimanya.

```
$mpiexec.exe -n 4 python .\sum_p2p.py
Rank 3 has value 40
Rank 2 has value 30
Rank 1 has value 20
Rank 0 has value 10
Rank 0 worked out the total 100
```

5. Sum-collective reduce

```
1  from mpi4py import MPI
2
3  comm = MPI.COMM_WORLD
4  rank = comm.Get_rank()
5  size = comm.Get_size()
6  val = (rank+1)*10
7  print "Rank %d has value %d" %(rank, val)
8  sum = comm.reduce(val, op=MPI.SUM, root=0)
9  if rank==0:
10     print "Rank 0 worked out the total %d" %sum
```

Line 8: menggunakan fungsi native dari MPI untuk mempercepat pemrosesan.

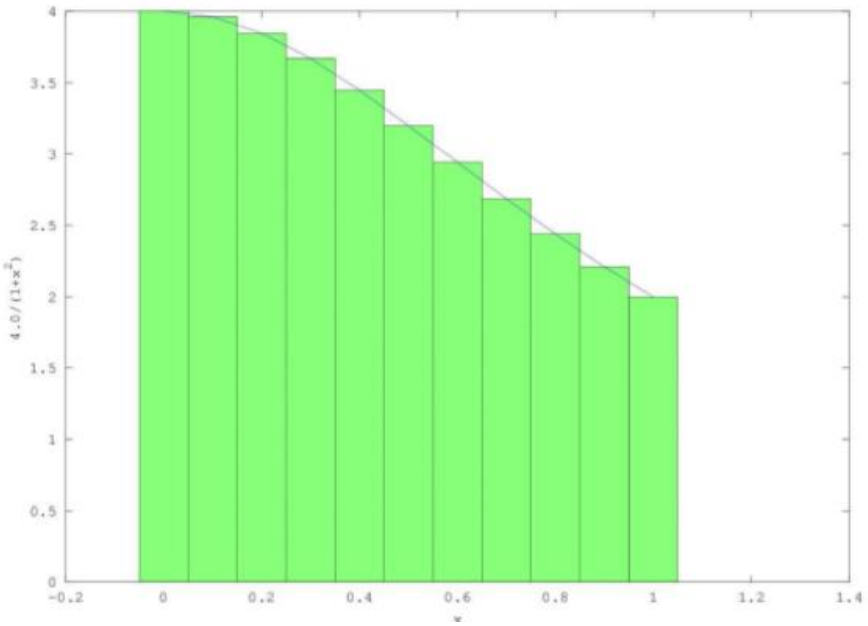


Menghitung PI

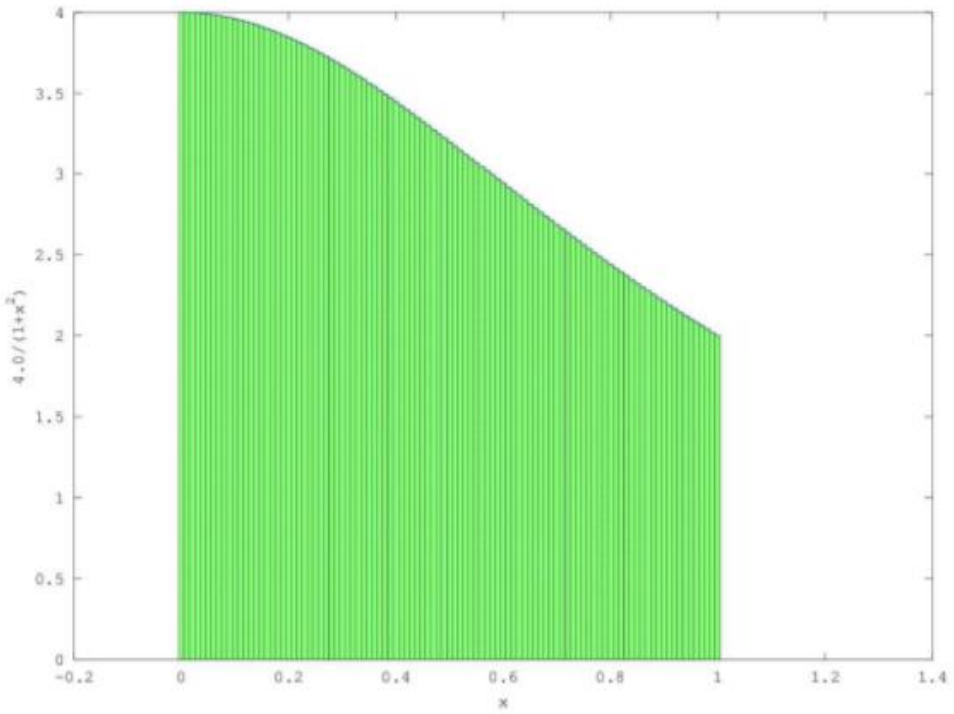
PI (3.1415…) dapat dihitung menggunakan integrasi F(x):

$$F(x) = \frac{4.0}{(1 + x^2)}$$

Integrasi F(x) tersebut dapat diaproksimasi dengan perhitungan discrete.



Dengan menaikkan banyaknya kotak (*steps*) maka aproksimasi akan mendekati nilai pi sebenarnya



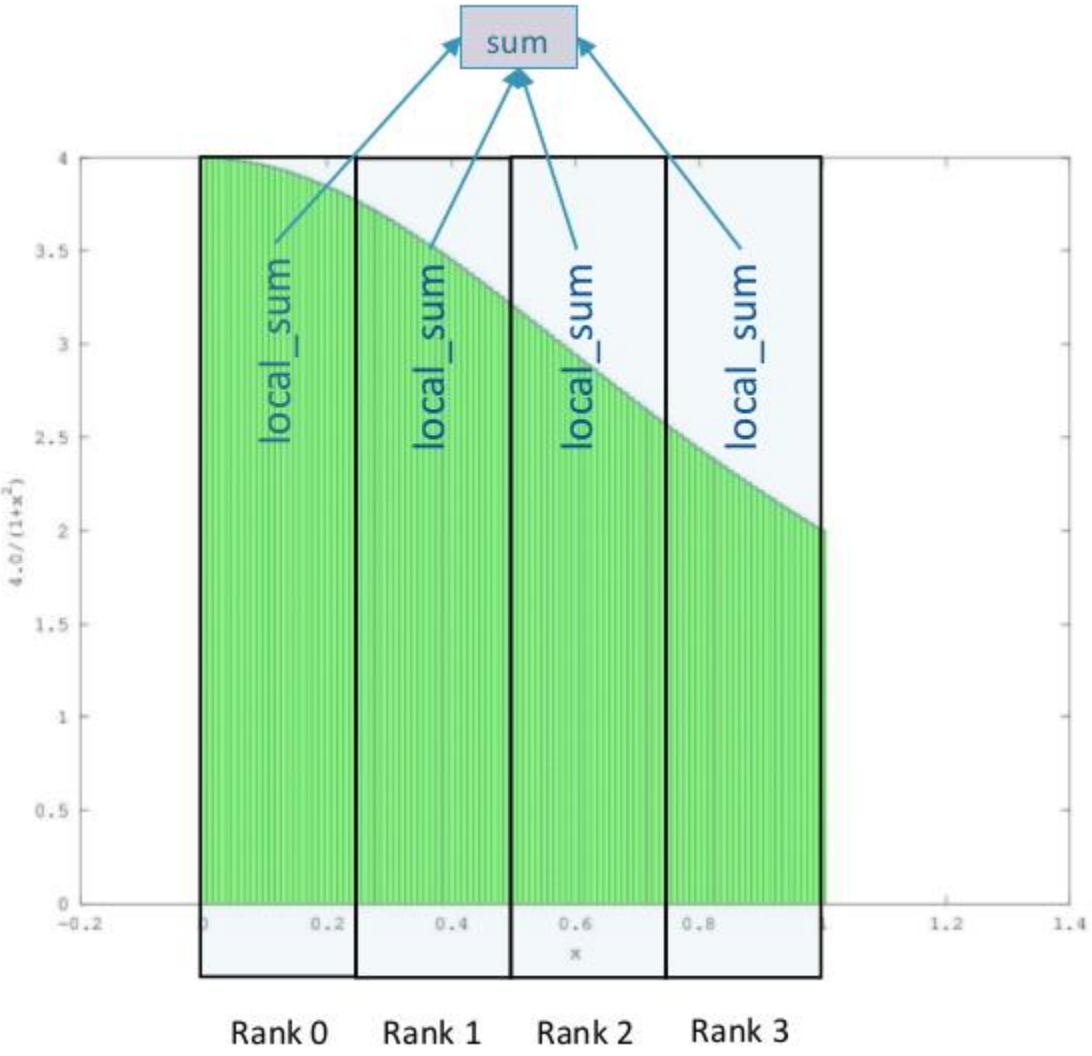
Menghitung PI cara serial

```
import time

def Pi(num_steps):
    start = time.time()
    step = 1.0/num_steps
    sum = 0
    for i in xrange(num_steps):
        x= (i+0.5)*step
        sum = sum + 4.0/(1.0+x*x)
    pi = step * sum
    end = time.time()
    print "Pi with %d steps is %f in %f secs" %(num_steps, pi, end-start)

if __name__ == '__main__':
    Pi(100000)
```

Metode ini terlalu lama dan memakan banyak resource. Akan kita gunakan MPI untuk membuatnya menjadi lebih cepat.

Solusi: *decompose then collect*
Ide dasar penyelesaian, Misal num_step =100 dan kita akan melakukan paralelisasi dengan 4 proses. Kita akan memberikan:

a) step 0-24 ke rank 0
b) step 25-49 ke rank 1
c) step 50-74 ke rank 2
d) step 75-99 ke rank 3



Tugas:
Ubah serial pi menjadi paralel pi menggunakan ide di atas!