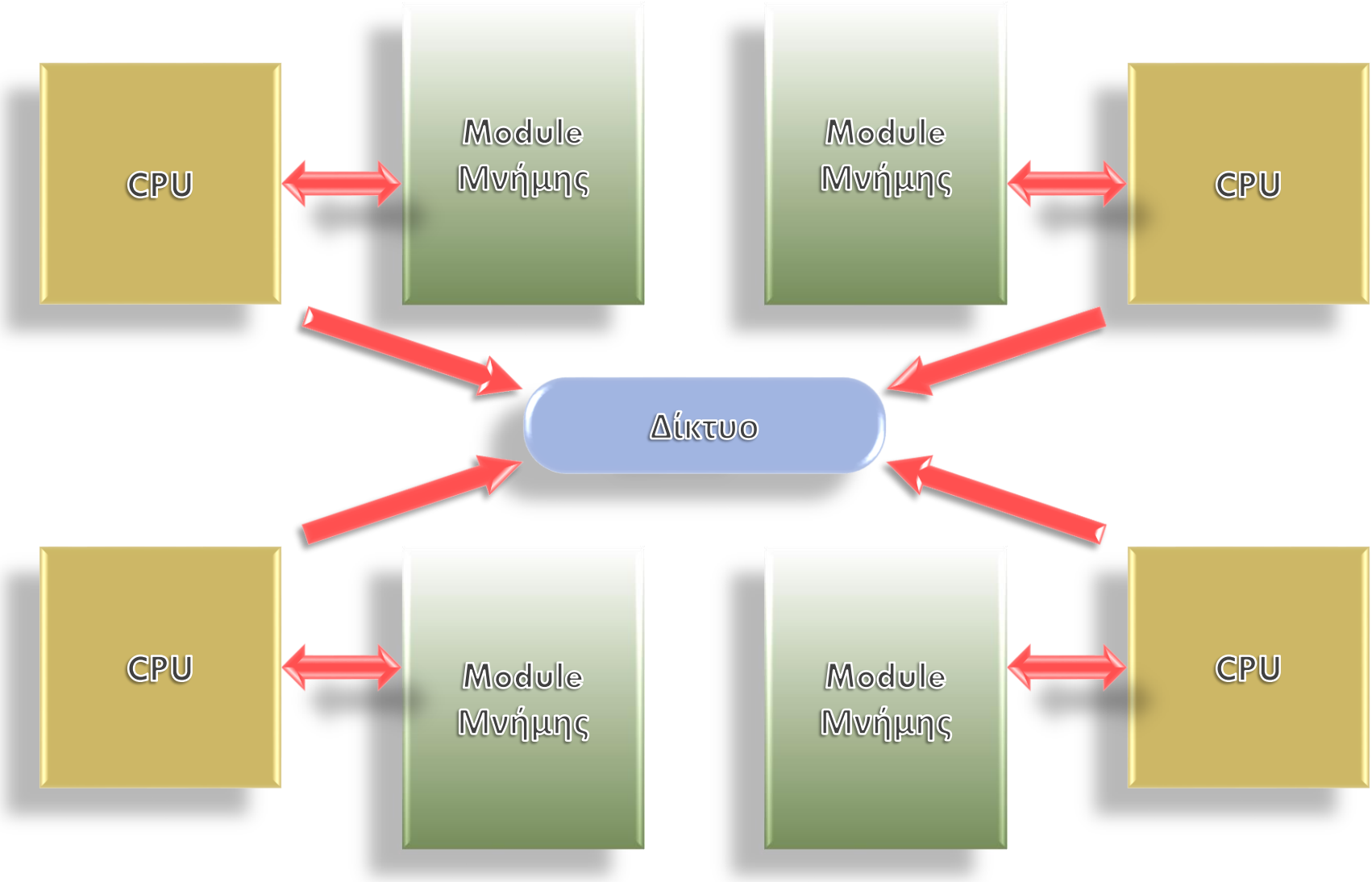
Ιωάννης Ε. Βενέτης

ΑΛΓΟΡΙΘΜΙΚΕΣ ΤΕΧΝΙΚΕΣ ΚΑΙ ΕΦΑΡΜΟΓΕΣ

21/01/2019

MPI (I)

Κατανεμημένη Μνήμη



Συστάδες (Clusters)

- Τυπικές συστάδες σήμερα
  - Κάθε κόμβος περιλαμβάνει μια κάρτα δικτύου τύπου Ethernet (ή ιδιαίτερου τύπου, π.χ. Myrinet, Infiniband)
  - Όλοι οι κόμβοι συνδέονται μέσω ενός δικτύου
  - Τα μηνύματα μεταδίδονται με χρήση πακέτων μέσω του δικτύου
- Όπως ένα σύστημα με αρχιτεκτονική κατανεμημένης μνήμης
  - Με πιο τυπικά χαρακτηριστικά στα υποσυστήματα

Πέρασμα μηνυμάτων (Message Passing)

- Αξιοποιούνται διεργασίες για την εκτέλεση μιας εφαρμογής
  - Κάθε διεργασία έχει τον δικό της χώρο διευθύνσεων μνήμης
  - Κάθε κόμβος έχει άμεση πρόσβαση μόνο στην τοπική μνήμη
- Δεδομένα
  - Διαχωρίζονται έτσι ώστε κάθε διεργασία να έχει ένα τμήμα των δεδομένων
  - Δεν υπάρχουν κοινά δεδομένα
  - Πρόσβαση σε δεδομένα που κατέχει άλλη διεργασία γίνεται με ρητές κλήσεις σε συναρτήσεις αποστολής / λήψης δεδομένων
  - Ο συντονισμός/συγχρονισμός γίνεται έμμεσα μέσω των συναρτήσεων αυτών

Single Program Multiple Data (SPMD)

- Κάθε διεργασία εκτελεί το ίδιο εκτελέσιμο αρχείο
  - Μπορεί να βρίσκονται όμως σε διαφορετικό σημείο εκτέλεσης του προγράμματος κάθε στιγμή
  - Διαχωρίζονται με βάση τον βαθμό (rank)
    - Αντίστοιχο με τον αριθμό νήματος στο OpenMP
    - Μπορούν να επεξεργάζονται υποσύνολα των δεδομένων ή να διαφοροποιούν την ροή εκτέλεσης τους

Πέρασμα μηνυμάτων (Message Passing)

- Πρώιμα θεωρητικά συστήματα
  - CSP: Communicating Sequential Processes
  - Αποστολή και αντίστοιχη λήψη σε άλλο επεξεργαστή
    - Και οι δύο επεξεργαστές περιμένουν
    - Το μοντέλο OCCAM στο σύστημα Transputer χρησιμοποιούσε το συγκεκριμένο μοντέλο
    - Προβλήματα απόδοσης λόγω της μη απαραίτητης αναμονής σε κάθε αποστολή και λήψη
- Σύγχρονα συστήματα
  - Οι λειτουργίες αποστολής δεν περιμένουν την λήψη των δεδομένων στους παραλήπτες

Προτυποποίηση περάσματος μηνυμάτων

- Πρώτες προσπάθειες
  - nxlib (Σε συστήματα Hybercube της Intel)
  - Παραλλαγές του συστήματος ncube
  - PVM
  - Κάθε εταιρεία είχε το δικό της πρότυπο
- MPI standard:
  - Σύμπραξη εταιρειών και ακαδημαϊκής κοινότητας
  - Με σκοπό την προτυποποίηση των μέχρι τότε χρησιμοποιούμενων πρακτικών
  - Κατέληξε με την έκδοση ενός ιδιαίτερα μεγάλου προτύπου
  - Δημοφιλές, χάρη στην υποστήριξη εταιρειών

MPI (Message Passing Interface)

- Είναι πρότυπο, όχι συγκεκριμένη υλοποίηση
- Βιβλιοθήκη ανταλλαγής μηνυμάτων
- Σχεδίαση σε στρώματα (layers)
- Σε υψηλό επίπεδο, παρέχει διεπαφή (interface) στον προγραμματιστή
- Σε χαμηλό επίπεδο, επικοινωνεί με το δίκτυο διασύνδεσης
- Υποστηρίζει C, C++, Fortran 77 και Fortran 90

Υλοποιήσεις MPI

- Open MPI: http://www.open-mpi.org
- MPICH: http://www-unix.mcs.anl.gov/mpi/mpich
- MPICH2: http://www-unix.mcs.anl.gov/mpi/mpich2
- MPICH-GM: http://www.myri.com/scs
- LAM/MPI: http://www.lam-mpi.org
- LA-MPI: http://public.lanl.gov/lampi
- SCI-MPICH: http://www.lfbs.rwth-aachen.de/users/joachim/SCI-MPICH
- MPI/Pro: http://www.mpi-softtech.com

Ένα απλό υποσύνολο του MPI

- Οι παρακάτω έξι συναρτήσεις επιτρέπουν την συγγραφή μεγάλου πλήθους προγραμμάτων:
  - MPI_Init
  - MPI_Finalize
  - MPI_Comm_size
  - MPI_Comm_rank
  - MPI_Send
  - MPI_Recv

Αρχικοποίηση/Τερματισμός διεργασιών

- int MPI_Init(int *argc, char ***argv);
  - Αρχικοποιεί τον υπολογισμό
  - Πρέπει να καλείται από κάθε διεργασία MPI
    - Καλείται ακριβώς μια φορά
    - Καλείται πριν από κάθε άλλη συνάρτηση MPI
  - Αρχικοποιεί την χρησιμοποιούμενη υλοποίηση του MPI
- int MPI_Finalize(void);
  - Τερματίζει τον υπολογισμό
  - Πρέπει να καλείται από κάθε διεργασία MPI
    - Καλείται ακριβώς μια φορά
  - Κάνει εκκαθάριση στην χρησιμοποιούμενη υλοποίηση του MPI

Προσδιορισμός διεργασιών

- int MPI_Comm_size(MPI_Comm comm, int *size);
  - Αποθηκεύει στην “size” το πλήθος των διεργασιών που συμμετέχουν στο MPI πρόγραμμα
- int MPI_Comm_rank(MPI_Comm comm, int *rank);
  - Αποθηκεύει στο “rank” τον βαθμό της διεργασίας

Communicators

- Παράμετρος “comm” στις παραπάνω συναρτήσεις
- Ορίζει ένα “πεδίο επικοινωνίας” (communication domain)
  - Υποσύνολο διεργασιών της εφαρμογής MPI που μπορούν να επικοινωνούν μεταξύ τους
- Θα χρησιμοποιήσουμε το “MPI_COMM_WORLD”
  - Σταθερά που ορίζει η εκάστοτε υλοποίηση
  - Ορίζει έναν communicator που περιέχει όλες τις διεργασίες της εφαρμογής MPI

Ένα πρώτο, απλό παράδειγμα

```
#include "mpi.h"
#include <stdio.h>
int main(int  argc, char *argv[]) {
    int rank, size;

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    printf(“Hello world! I'm %d of %d\n”, rank, size);

    MPI_Finalize();

    return(0);
}
```

Μεταγλώττιση/εκτέλεση προγράμματος MPI

- Μεταγλώττιση
  - mpicc –O3 –Wall –o my_prog my_prog.c
- Εκτέλεση:
  - mpirun -np 4 ./my_prog
- Επικοινωνεί με μια διεργασία-δαίμονα σε κάθε κόμβο του συστήματος.
- Έχει ως αποτέλεσμα την δημιουργία και εκτέλεση μιας διεργασίας σε κάθε κόμβο.

Διαφοροποίηση εκτέλεσης με βάση το rank

```
#include "mpi.h"
#include <stdio.h>
int main(int  argc, char *argv[]) {
    int rank, size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (rank == 2) {
        printf("I am process 2 and I am different!\n");
    } else {
        printf("Hello world! I'm %d of %d\n", rank, size);
    }
    MPI_Finalize();

    return(0);
}
```

Επικοινωνία στο MPI

- Μπορεί να είναι point-to-point ή συλλογική (collective)
- Μπορεί να είναι synchronous, buffered ή ready (ανάλογα με το τι θεωρείται ως συνθήκη επιτυχίας)
- Μπορεί να είναι blocking ή non-blocking (ανάλογα με το πότε επιστρέφει η συνάρτηση επικοινωνίας)

Point-to-point communication

- Επιτρέπει την ανταλλαγή δεδομένων μεταξύ συγκεκριμένων διεργασιών ενός προγράμματος MPI

Αποστολή μηνύματος στο MPI

- int MPI_Send(void *buf, int count,
  MPI_Datatype datatype, int dest, int tag,
  MPI_Comm comm);
  - “buf”: Η διεύθυνση των δεδομένων προς αποστολή
  - “count”: Το πλήθος των στοιχείων που θα αποσταλούν
  - “datatype”: Ο τύπος δεδομένων κάθε στοιχείου
  - “dest”: Ο βαθμός της διεργασίας-παραλήπτης
  - “tag”: Ετικέτα του μηνύματος
  - “comm”: Communicator

Λήψη μηνύματος στο MPI

- int MPI_Recv(void *buf, int count,
  MPI_Datatype datatype, int source, int tag,
  MPI_Comm comm, MPI_Status *status);
  - “buf”: Η διεύθυνση αποθήκευσης των δεδομένων που θα παραληφθούν
  - “count”: Το πλήθος των στοιχείων που θα παραληφθούν
  - “datatype”: Ο τύπος δεδομένων κάθε στοιχείου
  - “source”: Ο βαθμός της διεργασίας-αποστολέας
  - “tag”: Ετικέτα του μηνύματος
  - “comm”: Communicator
  - “status”: Δείχνει την κατάσταση του μηνύματος

Τύποι δεδομένων στο MPI

- Το MPI παρέχει τους δικούς του τύπους δεδομένων
  - Λόγοι φορητότητας (portability)
  - Επιτρέπει ορισμό νέων τύπων δεδομένων
    - Derived data types
- Οι προκαθορισμένοι τύπου δεδομένων του MPI αντιστοιχούν στους προκαθορισμένους τύπους της γλώσσας
  - MPI_SHORT → short int
  - MPI_INT → int
  - MPI_LONG → long int
  - MPI_FLOAT → float
  - MPI_DOUBLE → double
  - MPI_CHAR → char
  - …

Υπολογισμός της παράστασης $1^2+2^2+...+N^2$ (1/3)

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[]) {
   int my_rank, p, i, res, finres, start, end, num, N;
   int source, target;
   int tag1 = 50, tag2 = 60;
   MPI_Status status;

   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Comm_size(MPI_COMM_WORLD, &p);

   if (my_rank == 0)  {
         printf(“Enter last number: ”);
         scanf("%d", &N);
         for (target = 1; target < p; target++) {
                  MPI_Send(&N, 1, MPI_INT, target, tag1, MPI_COMM_WORLD);
         }
   } else {
         MPI_Recv(&N, 1, MPI_INT, 0, tag1, MPI_COMM_WORLD, &status);
   }
```

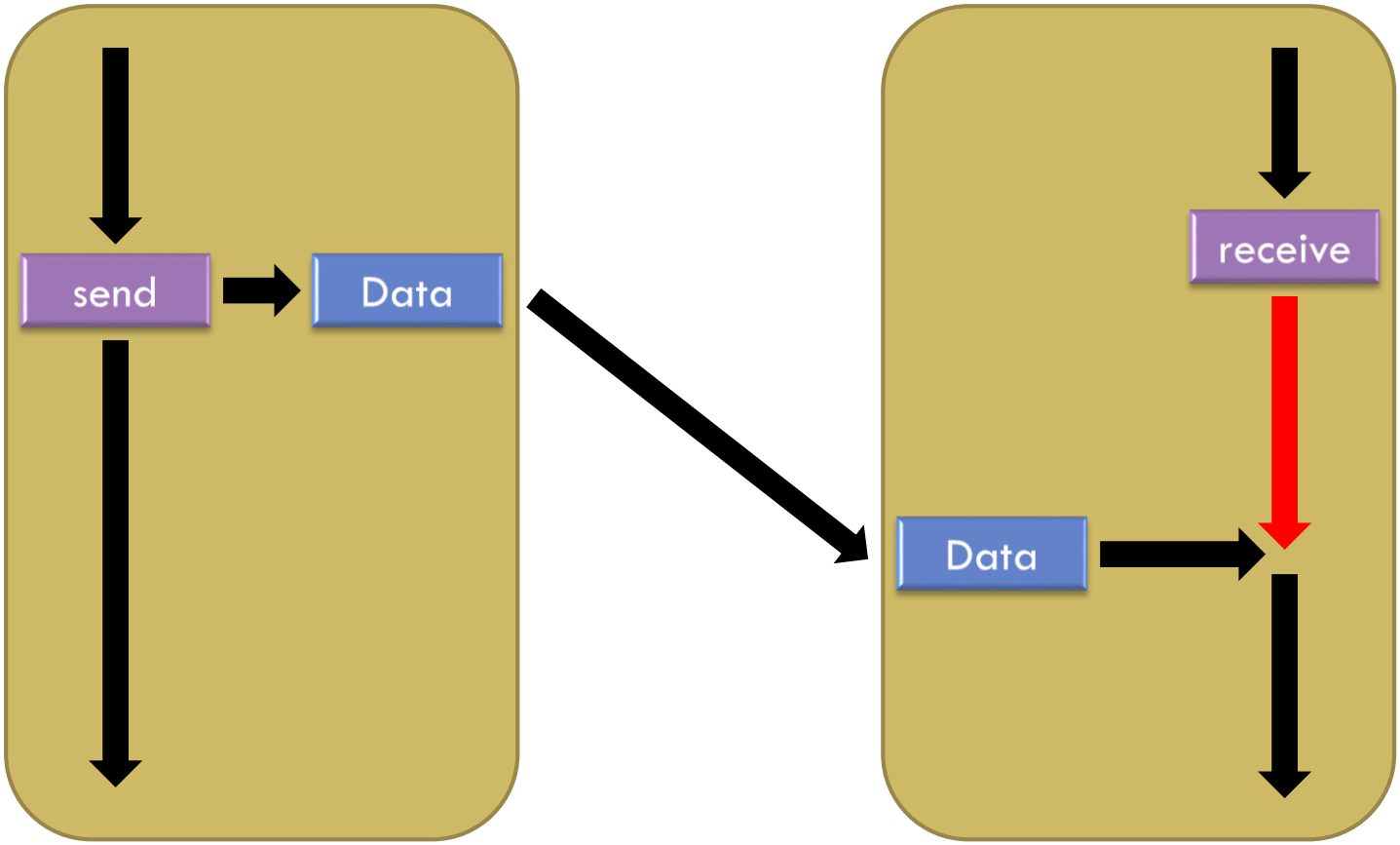
Υπολογισμός της παράστασης $1^2+2^2+...+N^2$ (2/3)

```
res   = 0;
num   = N / p;
start = (my_rank * num) + 1;
end   = start + num;

for (i = start; i < end; i++) {
      res += (i * i);
}
```

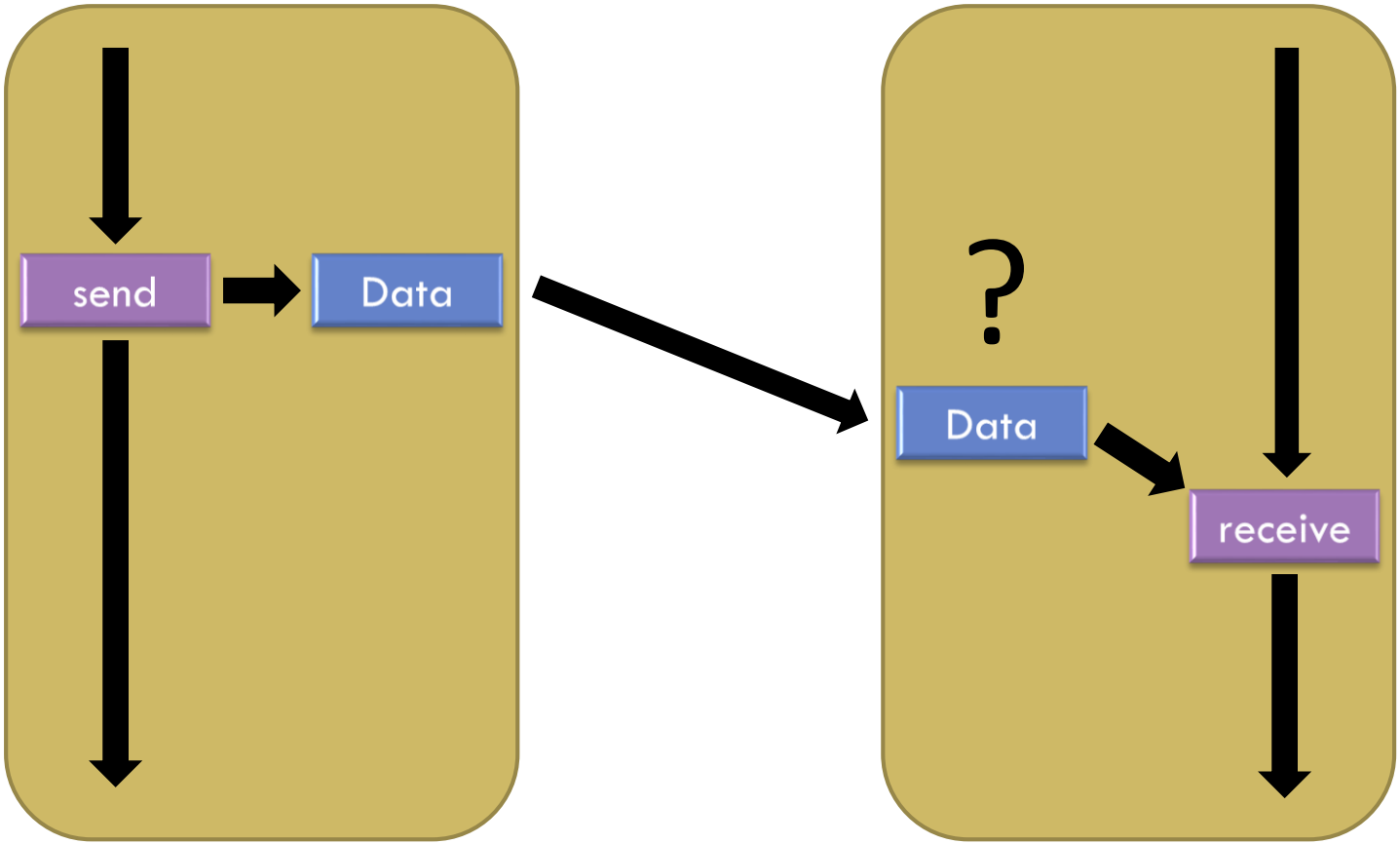
Υπολογισμός της παράστασης $1^2+2^2+...+N^2$ (3/3)

```
    if (my_rank != 0)  {
          MPI_Send(&res, 1, MPI_INT, 0, tag2, MPI_COMM_WORLD);
    } else {
          finres = res;

          printf("\nResult of process %d: %d\n", my_rank, res);

          for (source = 1; source < p; source++)  {
                    MPI_Recv(&res, 1, MPI_INT, source, tag2, MPI_COMM_WORLD, &status);
                    finres += res;
                    printf("\nResult of process %d: %d\n", source, res);
          }
          printf("\n\n\nFinal result: %d\n", finres);
    }

    MPI_Finalize();

    return(0);
}
```

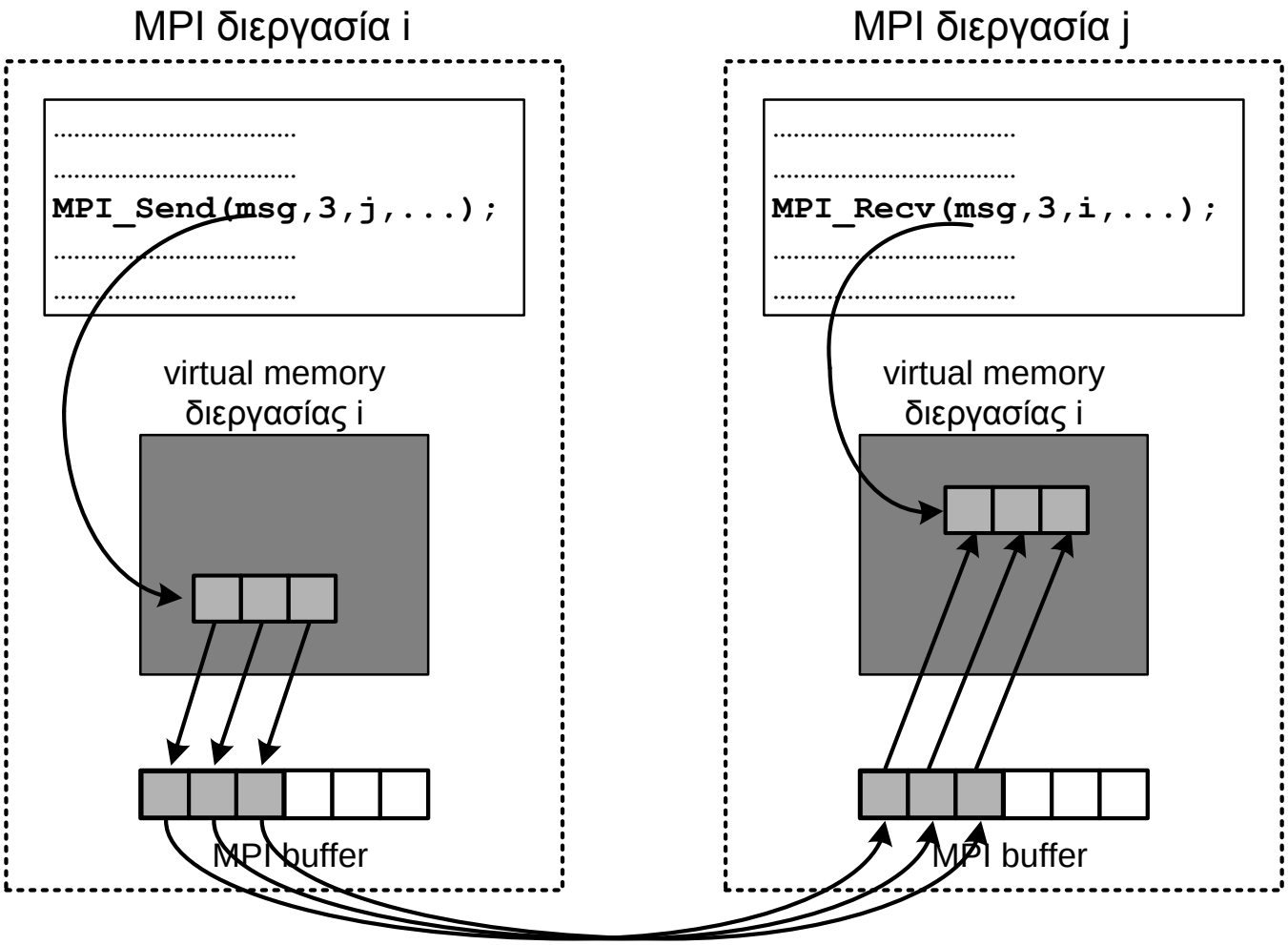
Σχετική ταχύτητα εκτέλεσης: Λήψη εκτελείται πριν την αποστολή δεδομένων



Σχετική ταχύτητα εκτέλεσης: Αποστολή εκτελείται πριν την λήψη δεδομένων



Αποστολή και λήψη στο MPI



Άθροισμα τετραγώνων στοιχείων διανύσματος (1/3)

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
    int my_rank, p, i, res, finres, num, N;
    int source, target;
    int tag1=50, tag2=60, tag3=70;
    int data[100];
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
```

Άθροισμα τετραγώνων στοιχείων διανύσματος (2/3)

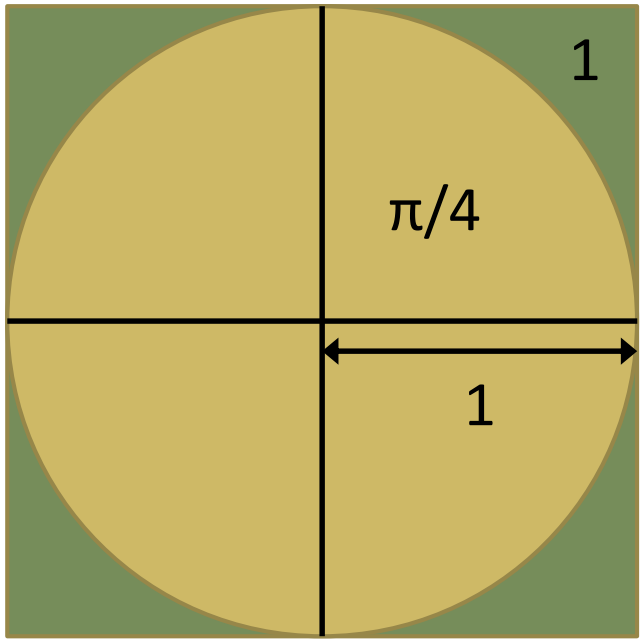
```
if (my_rank == 0)  {
      printf(“Enter size of vector: ");
      scanf("%d", &N);

      printf(“Enter %d vector elements: ”, N);
      for (i = 0; i < N; i++) {
               scanf(“%d”, &data[i]);
      }

      for (target = 1; target < p; target++) {
               MPI_Send(&N, 1, MPI_INT, target, tag1, MPI_COMM_WORLD);
      }

      num = N / p;
      i   = num;
      for (target = 1; target < p; target++)  {
               MPI_Send(&data[i], num, MPI_INT, target, tag2, MPI_COMM_WORLD);
               i += num;
      }
} else {
      MPI_Recv(&N, 1, MPI_INT, 0, tag1, MPI_COMM_WORLD, &status);
      num = N / p;
      MPI_Recv(&data[0], num, MPI_INT, 0, tag2, MPI_COMM_WORLD, &status);
}
```

Άθροισμα τετραγώνων στοιχείων διανύσματος (3/3)

```
    res = 0;
    for (i = 0; i < num; i++) {
         res += (data[i] * data[i]);
    }

    if (my_rank != 0) {
          MPI_Send(&res, 1, MPI_INT, 0, tag3, MPI_COMM_WORLD);
    } else {
          finres = res;

          printf("\nResult of process %d: %d\n", my_rank, res);

          for (source = 1; source < p; source++) {
                    MPI_Recv(&res, 1, MPI_INT, source, tag3, MPI_COMM_WORLD, &status);
                    finres += res;
                    printf("\nResult of process %d: %d\n", source, res);
          }
          printf("\n\n\nFinal result: %d\n", finres);
    }

    MPI_Finalize();

    return(0);
}
```

Υπολογισμός π με μέθοδο Monte Carlo (1/3)

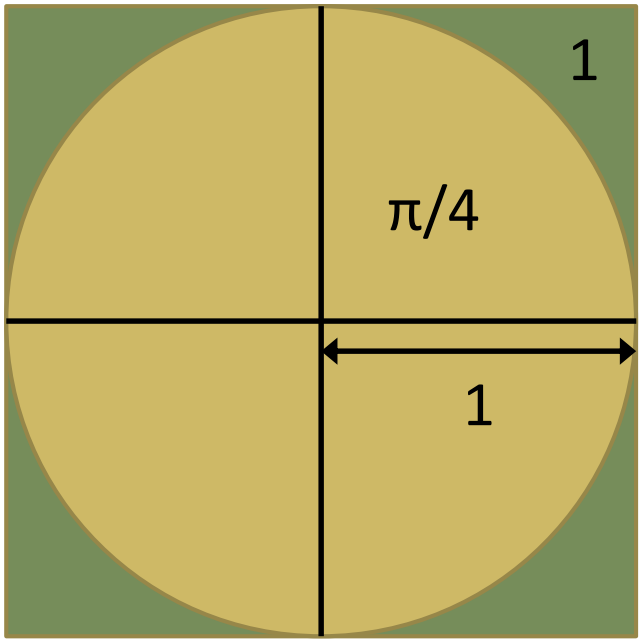
- Ιδέα
  - Φτιάξε τετράγωνο με πλευρά 1 και τεταρτημόριο με ακτίνα 1
    - Εμβαδό τετραγώνου = 1
    - Εμβαδό τεταρτημορίου = π/4
  - Τοποθέτησε N τυχαία σημεία
    - Έστω ότι M από αυτά βρίσκονται εντός του τεταρτημορίου
  - Τότε:



$$\frac{\text{Εμβαδό κύκλου}}{\text{Εμβαδό τετραγώνου}} = \frac{\pi/4}{1} \approx \frac{M}{N} \Rightarrow \pi \approx 4 \cdot \frac{M}{N}$$

Υπολογισμός π με μέθοδο Monte Carlo (2/3)

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main(int argc, char* argv[])
{
    int i, rank, np, c, local_c, local_N, N=1000000;
    double x, y;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    srand48(rank);
    local_c = 0;
    local_N = N / np;

    for (i = 0; i < local_N; i++) {
          x = drand48();
          y = drand48();
          if (((x * x) + (y * y)) <= 1.0) {
                    local_c++;
          }
    }
```



Αποφεύγει την δημιουργία των ίδιων ψευδο-τυχαίων αριθμών σε όλους τους επεξεργαστές

Υπολογισμός π με μέθοδο Monte Carlo (3/3)

```
    if (my_rank != 0)  {
          MPI_Send(&local_c, 1, MPI_INT, 0, tag2, MPI_COMM_WORLD);
    } else {
          c = local_c;

          for (source = 1; source < p; source++)  {
                    MPI_Recv(&local_c,1,MPI_INT,source,tag2,MPI_COMM_WORLD,&status);
                    c += local_c;
          }
          printf("Estimate of Pi = %24.16f\n", (4.0 * c) / N);
    }

    MPI_Finalize();

    return(0);
}
```