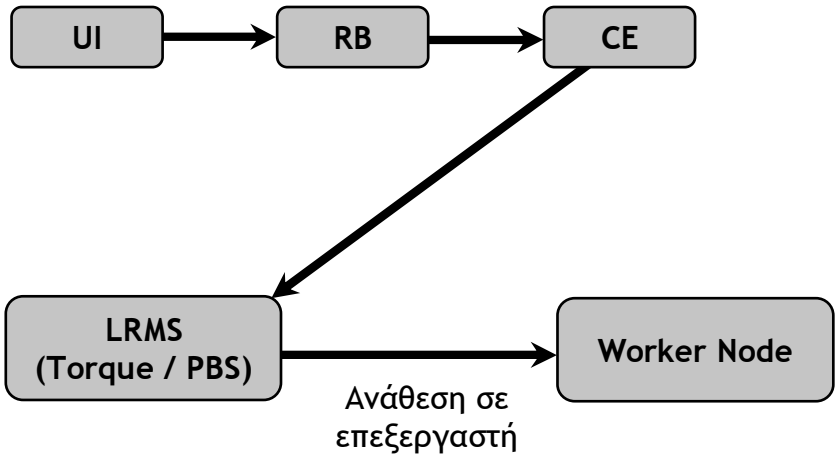
Το Message Passing Interface (MPI) και η υποστήριξή του στο EGEE Grid

Vangelis Koukis
HG-01-GRNET and HG-06-EKT admin team
vkoukis@cslab.ece.ntua.gr

UoA, 2006/10/24

Πορεία μιας σειριακής εργασίας στο Grid

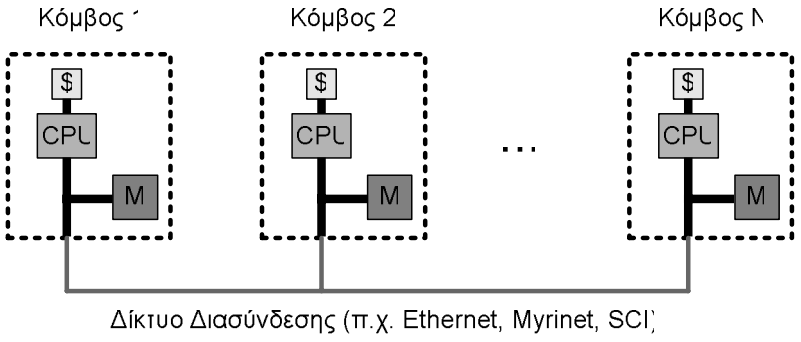


Ανάγκη για υποστήριξη MPI στο Grid

- Μεγάλη εγκατεστημένη υπολογιστική ισχύς: Πώς την εκμεταλλευόμαστε;
  - 1000άδες επεξεργαστών
  - Πολλές ανεξάρτητες (σειριακές) δουλειές, για ανεξάρτητη επεξεργασία διαφορετικού υποσυνόλου των δεδομένων εισόδου
- Και αν υπάρχουν εξαρτήσεις;
  - Αν το πρόβλημα δεν είναι "Embarassingly Parallel";

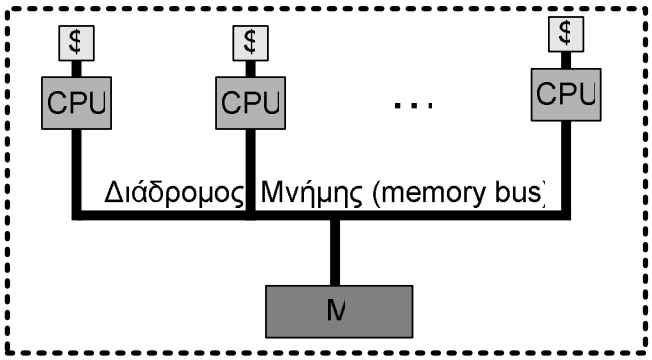
Παράλληλες Αρχιτεκτονικές

- Αρχιτεκτονική κατανεμημένης μνήμης (distributed memory systems, π.χ. Cluster)



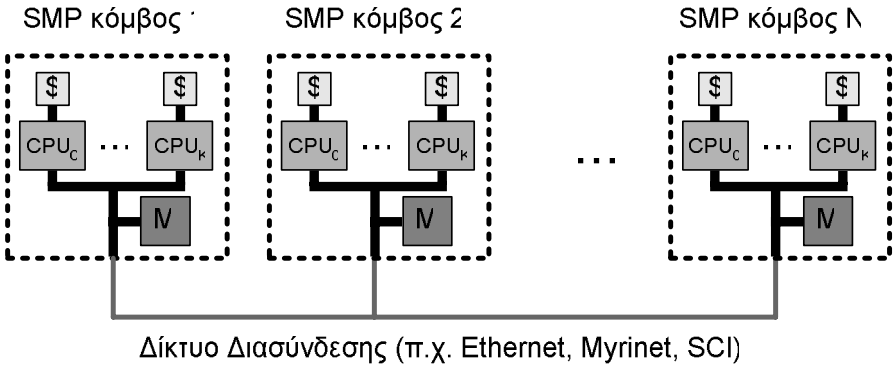
Παράλληλες Αρχιτεκτονικές (2)

- Αρχιτεκτονική μοιραζόμενης μνήμης (shared memory systems, π.χ. SMP)

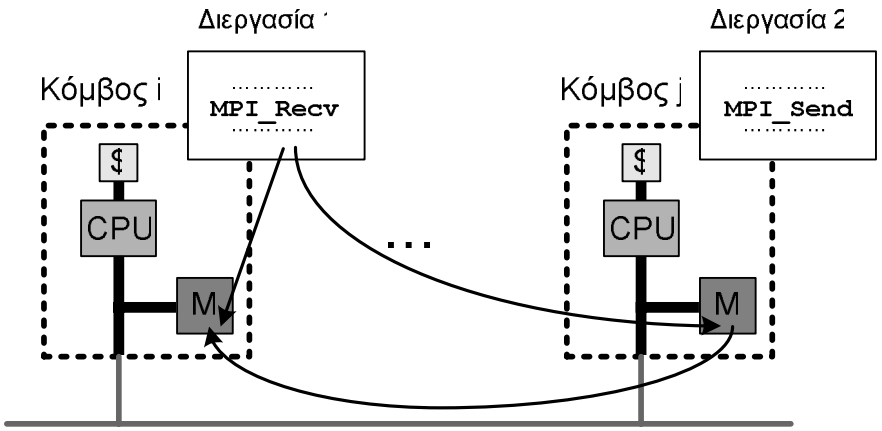


Παράλληλες Αρχιτεκτονικές (3)

- Υβριδική αρχιτεκτονική (π.χ. SMP cluster)



Παράλληλες Αρχιτεκτονικές (4)



Τι είναι το MPI;

- Είναι πρότυπο, όχι συγκεκριμένη υλοποίηση
- Βιβλιοθήκη ανταλλαγής μηνυμάτων
- Σχεδίαση σε στρώματα (layers)
- Σε υψηλό επίπεδο, παρέχει συγκεκριμένη προγραμματιστική διεπαφή (interface)
- Σε χαμηλό επίπεδο, επικοινωνεί με το δίκτυο διασύνδεσης
- Υποστηρίζει C, C++, Fortran 77 και F90

Υλοποιήσεις MPI

- 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
- Open MPI
  http://www.open-mpi.org
- SCI-MPICH
  http://www.lfbs.rwth-aachen.de/users/joachim/SCI-MPICH
- MPI/Pro
  http://www.mpi-softtech.com
- MPICH-G2
  http://www3.niu.edu/mpi

Single Program, Multiple Data (SPMD)

- Πολλές διεργασίες, όλες εκτελούν το ίδιο πρόγραμμα
- Διακρίνονται με βάση το βαθμό (rank) που αποδίδεται σε κάθε μία διεργασία
  - Επεξεργάζεται διαφορετικό υποσύνολο δεδομένων
  - Διαφοροποιεί τη ροή εκτέλεσής της
- Επιδίωξη παράλληλου προγραμματισμού
  - Μεγιστοποίηση παραλληλίας
  - Αποδοτική αξιοποίηση πόρων συστήματος (π.χ. μνήμη)
  - Ελαχιστοποίηση όγκου δεδομένων επικοινωνίας
  - Ελαχιστοποίηση αριθμού μηνυμάτων
  - Ελαχιστοποίηση συγχρονισμού

Διεργασίες και Communicators

- Σε κάθε διεργασία αποδίδεται ένα μοναδικό rank στο εύρος 0...P-1, όπου P το συνολικό πλήθος διεργασιών στον συγκεκριμένο communicator
- Σε γενικές γραμμές, ο communicator ορίζει ένα σύνολο από διεργασίες που μπορούν να επικοινωνούν μεταξύ τους (π.χ. MPI_COMM_WORLD)
- Προσοχή: Αναφερόμαστε πάντα σε διεργασίες, όχι σε επεξεργαστές

Τυπική δομή κώδικα MPI

```
#include <mpi.h>

int main(int argc, char *argv[])
{
      ...
      /* Πρώτη κλήση MPI */
      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
      MPI_Comm_size(MPI_COMM_WORLD, &size);
      ...
      /* Τελευταία κλήση MPI */
      MPI_Finalize();
}
```

Βασικές Συναρτήσεις στο MPI

- MPI_Init(argc,argv)
  - Αρχικοποίηση
- MPI_Comm_rank(comm,rank)
  - Εύρεση του rank της διεργασίας στον comm
- MPI_Comm_size(comm,size)
  - Εύρεση πλήθους διεργασιών size σε comm
- MPI_Send(sndbuf,count,datatype,dest,tag,comm)
  - Αποστολή μηνύματος σε διεργασία dest
- MPI_Recv(rcvbuf,count,datatype,source,tag, comm,status)
  - Λήψη μηνύματος από διεργασία source
- MPI_Finalize()
  - Τερματισμός

Βασικές Συναρτήσεις στο MPI (2)

```
int MPI_Init(int* argc, char*** argv)
```

- Αρχικοποίηση περιβάλλοντος MPI
- Παράδειγμα:

```
int main(int argc,char *argv[])
{
      …
      MPI_Init(&argc,&argv);
      …
}
```

Βασικές Συναρτήσεις στο MPI (3)

```
int MPI_Comm_rank (MPI_Comm comm, int* rank)
```

- Καθορισμός *rank* καλούσας διεργασίας που ανήκει στον communicator *comm*
- Παράδειγμα:

```
int rank;

MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

Βασικές Συναρτήσεις στο MPI (4)

```
int MPI_Comm_size (MPI_Comm comm, int* size)
```

- Καθορισμός πλήθους διεργασιών *size* που ανήκουν στον communicator *comm*
- Παράδειγμα:

```
int size;

MPI_Comm_size(MPI_COMM_WORLD,&size);
```

Βασικές Συναρτήσεις στο MPI (5)

```
int MPI_Send(void *buf, int count, int dest,
int tag, MPI_Datatype datatype, MPI_Comm
comm)
```

- Αποστολή μηνύματος *buf* από καλούσα διεργασία σε διεργασία με rank *dest*
- Ο πίνακας *buf* έχει *count* στοιχεία τύπου *datatype*
- Παράδειγμα:

```
int message[20],dest=1,tag=55;

MPI_Send(message, 20, dest, tag, MPI_INT,
     MPI_COMM_WORLD);
```

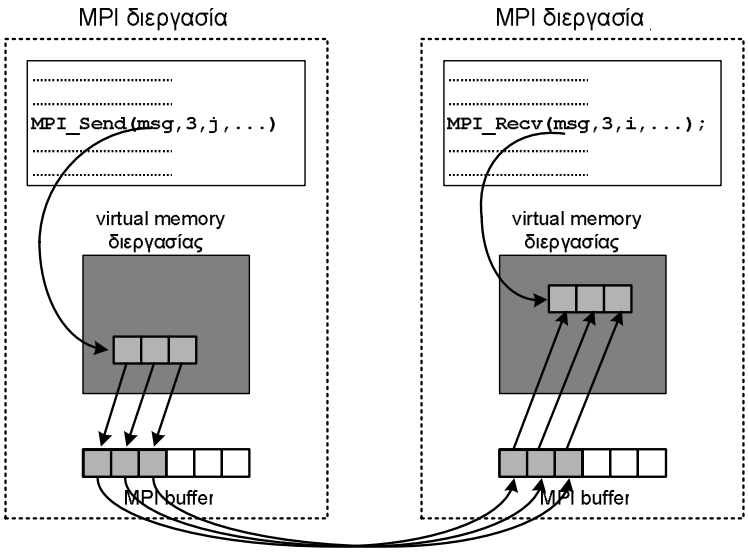
Βασικές Συναρτήσεις στο MPI (6)

```
int MPI_Recv(void *buf, int count, int
source, int tag, MPI_Datatype datatype,
MPI_Comm comm, MPI_Status *status)
```

- Λήψη μηνύματος από διεργασία με rank *source* και αποθήκευση στον *buf*
- Λαμβάνονται το πολύ *count* δεδομένα τύπου *datatype* (ακριβής αριθμός με MPI_Get_count)
- Wildcards
  - MPI_ANY_SOURCE, MPI_ANY_TAG
- Παράδειγμα:

```
int message[50],source=0,tag=55;
MPI_Status status;
MPI_Recv(message, 50, source, tag,
     MPI_INT, MPI_COMM_WORLD, &status);
```

Βασικές Συναρτήσεις στο MPI (7)



Βασικές Συναρτήσεις στο MPI (8)

```
int MPI_Finalize()
```

- Τερματισμός περιβάλλοντος MPI
- Πρέπει να αποτελεί την τελευταία κλήση MPI του προγράμματος

Παράδειγμα

```
/* Παράλληλος υπολογισμός της παράστασης f(0)+f(1)*/
#include <mpi.h>

int main(int argc,char** argv){
   int v0,v1,sum,rank;
   MPI_Status stat;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD,&rank);
   if(rank==1) {
      v1=f(1);
      MPI_Send(&v1,1,0,50,MPI_INT,MPI_COMM_WORLD);
   else if(rank==0){
      v0=f(0);
      MPI_Recv(&v1,1,1,50,MPI_INT,MPI_COMM_WORLD,&stat);
      sum=v0+v1;
   }
   MPI_Finalize();
}
```

Διεργασία 1

Διεργασία 0

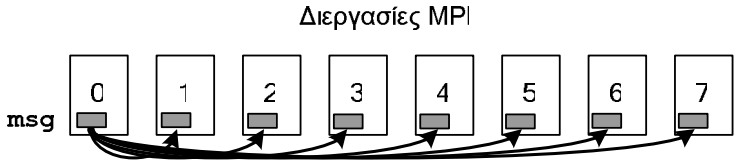
Είδη Επικοινωνίας

- Point-to-point ή Συλλογική (Collective)
- Synchronous, buffered ή ready
  - ➨ανάλογα με το τι θεωρείται ως συνθήκη επιτυχίας)
- Blocking ή non-blocking
  - ➨ανάλογα με το πότε επιστρέφει η συνάρτηση επικοινωνίας

Συλλογική Επικοινωνία

Παράδειγμα: Αποστολή του msg στις διεργασίες 1-7 από τη 0

```
if (rank == 0)
  for (dest = 1; dest < size; dest++)
    MPI_Send(msg,count,dest,tag,MPI_FLOAT,MPI_COMM_WORLD);
```



Γενικά: Για *p* διεργασίες έχουμε *p* - 1 βήματα επικοινωνίας

Συλλογική Επικοινωνία (2)

Παράδειγμα: Αποστολή του msg στις διεργασίες 1-7 από τη 0

```
MPI_Bcast(msg,count,MPI_FLOAT,0,MPI_COMM_WORLD);
```

Γενικά: Για *p* διεργασίες έχουμε $\lceil \log_2 p \rceil$ βήματα επικοινωνίας

Συλλογική Επικοινωνία (3)

```
int MPI_Bcast(void* message, int count,
MPI_Datatype datatype, int root, MPI_Comm
comm)
```

- Αποστολή του *message* από τη διεργασία με rank *root* προς όλες τις διεργασίες του communicator *comm*
- Το message περιέχει *count* δεδομένα τύπου *datatype*
- Καλείται από όλες τις διεργασίες του *comm*

Συλλογική Επικοινωνία (4)

```
int MPI_Reduce(void* operand, void*
result, int count, MPI_Datatype datatype,
MPI_Op op, int root, MPI_Comm comm)
```

- Τα δεδομένα *operand* συνδυάζονται με εφαρμογή του τελεστή *op*, και το αποτέλεσμα αποθηκεύεται στη διεργασία *root* στο *result*
- Πρέπει να κληθεί από όλες τις διεργασίες του *comm*
- MPI_Op: MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD κλπ.
- Αντίστοιχα και MPI_Allreduce

Συλλογική Επικοινωνία (5)

```
/* Παράλληλος υπολογισμός της παράστασης f(0)+f(1)*/
#include <mpi.h>

int main(int argc,char *argv[]){
   int sum,rank;
   MPI_Status stat;

   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD,&rank);
   /* Υπολογισμός τιμών στον f[] */
   MPI_Reduce(&f[rank],&sum,1,MPI_INT,MPI_SUM,0,
             MPI_COMM_WORLD);
   MPI_Finalize();
}
```

Συλλογική Επικοινωνία (6)

```
int MPI_Barrier(MPI_Comm comm)
```

- Συγχρονισμός διεργασιών του communicator *comm*
- Η εκτέλεση τους συνεχίζεται μόνον όταν *όλες* έχουν εκτελέσει την κλήση
- Περιορίζει την παραλληλία

Συλλογική Επικοινωνία (7)

```
int MPI_Gather(void* sendbuf, int sendcnt,
MPI_Datatype sendtype, void* recvbuf, int
recvcount, MPI_Datatype recvtype, int root,
MPI_Comm comm)
```

- Συνενώνονται στη διεργασία *root* οι πίνακες *sendbuf* των υπολοιπών (κατά αύξουσα σειρά rank)
- Το αποτέλεσμα αποθηκεύεται στον πίνακα *recvbuf*, ο οποίος έχει νόημα μόνο στη διεργασία *root*
- Αντίστοιχα και MPI_Allgather
- Αντίστροφη: MPI_Scatter

Synchronous – Buffered - Ready

- Αναφέρονται σε λειτουργία αποστολής, διαφοροποιούνται ως προς λειτουργία λήψης
- Υπάρχουν τόσο σε blocking, όσο και σε non-blocking μορφή
- Το απλό MPI_Send μπορεί να είναι είτε synchronous είτε buffered: εξαρτάται από υλοποίηση

Synchronous – Buffered – Ready (2)

- `int MPI_Ssend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)`
  - Επιτυγχάνει μόνο όταν πάρει επιβεβαίωση λήψης από δέκτη - ασφαλές
- `int MPI_Bsend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)`
  - Επιτρέφει αμέσως, αντιγράφοντας το μήνυμα σε system buffer για μελλοντική μετάδοση - σφάλμα σε έλλειψη πόρων
- `int MPI_Rsend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)`
  - Επιστρέφει αμέσως, αλλά επιτυγχάνει μόνο αν έχει προηγηθεί αντίστοιχο receive από το δέκτη - αβέβαιο

Synchronous – Buffered – Ready (3)

MPI_Bsend	MPI_Ssend	MPI_Rsend
Τοπικό	Μη τοπικό	Τοπικό
2 αντιγραφές στη μνήμη	1 αντιγραφή στη μνήμη	1 αντιγραφή στη μνήμη
Αποτυγχάνει ελλείψει πόρων	Δεν αποτυγχάνει ελλείψει πόρων	Δεν αποτυγχάνει ελλείψει πόρων
Δεν αποτυγχάνει αν δεν έχει προηγηθεί λήψη	Δεν αποτυγχάνει αν δεν έχει προηγηθεί λήψη	Αποτυγχάνει αν δεν έχει προηγηθεί λήψη

Non – Blocking Communication

- Άμεση επιστροφή
- Δεν είναι ασφαλές να επαναχρησιμοποιηθούν οι buffers επικοινωνίας πριν ελεγχθεί η επιτυχία
- Δύο δυνατότητες για έλεγχο επιτυχίας της επικοινωνίας

```
➡ int MPI_Test (MPI_Request* request,int*
  flag, MPI_Status* status)
➡ int MPI_Wait (MPI_Request*
  request,MPI_Status* status)
```

Non – Blocking Communication (2)

- Κάθε blocking συνάρτηση έχει την αντίστοιχη non-blocking:

➡MPI_Isend (για MPI_Send)
➡MPI_Issend (για MPI_Ssend)
➡MPI_Ibsend (για MPI_Bsend)
➡MPI_Irsend (για MPI_Rsend)
➡MPI_Irecv (για MPI_Recv)

Non – Blocking Communication (3)

- Ποιο είναι το όφελος;
  - ➡Επικάλυψη υπολογισμού – επικοινωνίας

Blocking	Non-blocking
`MPI_Recv();`	`MPI_Irecv();`
`MPI_Send();`	`MPI_Isend();`
`Compute();`	`Compute();`
	`Waitall();`

Τύποι Δεδομένων MPI

MPI_CHAR: 8-bit χαρακτήρας
MPI_DOUBLE: 64-bit κινητής υποδιαστολής
MPI_FLOAT: 32-bit κινητής υποδιαστολής
MPI_INT: 32-bit ακέραιος
MPI_LONG: 32-bit ακέραιος
MPI_LONG_DOUBLE: 64-bit κινητής υποδιαστολής
MPI_LONG_LONG: 64-bit ακέραιος
MPI_LONG_LONG_INT: 64-bit ακέραιος
MPI_SHORT: 16-bit ακέραιος
MPI_SIGNED_CHAR: 8-bit προσημασμένος χαρακτήρας
MPI_UNSIGNED: 32-bit απρόσημος ακέραιος
MPI_UNSIGNED_CHAR: 8-bit απρόσημος χαρακτήρας
MPI_UNSIGNED_LONG: 32-bit απρόσημος ακέραιος
MPI_UNSIGNED_LONG_LONG: 64-bit απρόσημος ακέραιος
MPI_UNSIGNED_SHORT: 16-bit απρόσημος ακέραιος
MPI_WCHAR: 16-bit απρόσημος χαρακτήρας

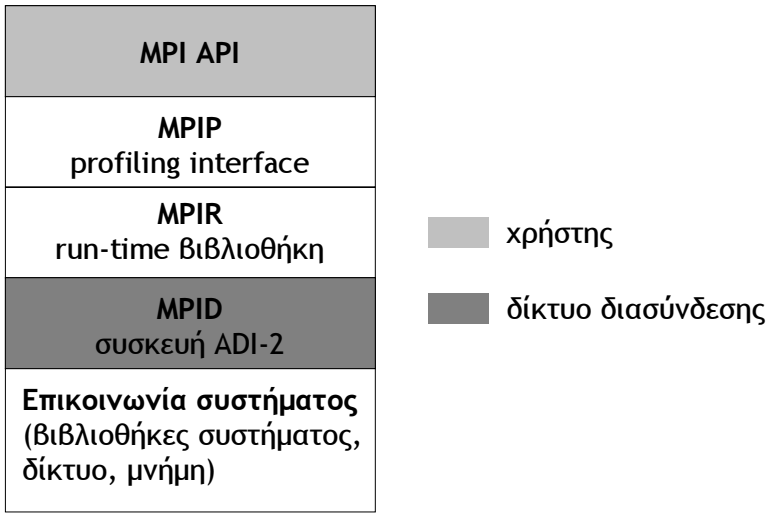
Τύποι Δεδομένων MPI (2)

- Ομαδοποίηση δεδομένων επικοινωνίας:
- Παράμετρος count (για ομοιογενή δεδομένα σε συνεχόμενες θέσεις μνήμης)
- MPI_Type_struct (derived datatype)
- MPI_Pack(), MPI_Unpack() (για ετερογενή δεδομένα)

Το πρότυπο MPI-2

- Παράλληλη είσοδος-έξοδος (Parallel I/O)
- Δυναμική διαχείριση διεργασιών (dynamic process management)
- Απομακρυσμένες λειτουργίες πρόσβαση στη μνήμη (remote memory operations)
  - ➡One-sided operations

Η υλοποίηση MPICH



Η Υλοποίηση MPICH (2)

- Ανά διεργασία, 1 send message queue, 2 receive queues
  - posted + unexpected
- Επιλογή device βάσει του destination rank
  - p4, shmem
- Επιλογή πρωτοκόλλου βάσει του message size
  - Short < 1024 bytes, rendezvous > 128000 bytes, eager ενδιάμεσα
- Έλεγχος ροής - Flow control
  - 1MB buffer space για eager πρωτόκολλο ανά ζεύγος διεργασιών

Εκτέλεση προγράμματος MPI (1)

- Παραδοσιακός τρόπος: Σε Cluster
- Linux cluster 16 κόμβων (kid1…kid16)
- Μεταγλώττιση και εκτέλεση
  - Κατάλληλο PATH για την υλοποίηση
    - `export PATH=/usr/local/bin/mpich-intel:…:$PATH`
  - Μεταγλώττιση με τις κατάλληλες βιβλιοθήκες
    - `mpicc test.c –o test –O3`
  - Εκτέλεση
    - `mpirun –np 16 test`

Επίδειξη!

- Hello World με υποβολή ενός 16-process MPICH job σε dedicated cluster (kids)

Εκτέλεση προγράμματος MPI (2)

- Σε ποια μηχανήματα εκτελούνται οι διεργασίες;
  - Machine file

```
$ cat <<EOF >machines
kid5
kid7
kid8
kid10
EOF

$ mpiCC test.cc -o test -O3 -static -Wall
$ mpirun -np 4 -machinefile machines test
```

Εκτέλεση προγράμματος MPI (3)

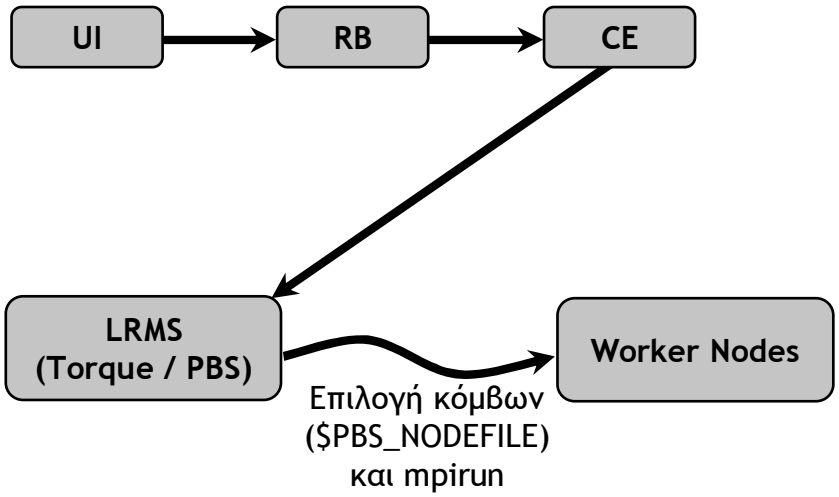
- Λεπτομέρειες Υλοποίησης
  - Πώς δημιουργούνται οι απαραίτητες διεργασίες; Implementation-specific
    - rsh / ssh χωρίς password, οι κόμβοι του cluster εμπιστεύονται ο ένας τον άλλο
    - Με χρήση daemons, ("lamboot" για το LAM/MPI)
- Τι γίνεται με το file I/O;
  - Shared storage ανάμεσα στους cluster nodes
    - NFS στην απλούστερη περίπτωση
    - Κάποιο παράλληλο fs, πχ. PVFS, GFS, GPFS

Ένταξη στο περιβάλλον του Grid

- Υποβολή εργασιών τύπου MPICH

```
Type = "job";
JobType = "MPICH";
NodeNumber = 64;
Executable = “mpihello";
StdOutput = "hello.out";
StdError = "hello.err";
InputSandbox = {"hello"};
OutputSandbox = {"hello.out","hello.err"};
#RetryCount = 7;
#Requirements = other.GlueCEUniqueID ==
"ce01.isabella.grnet.gr:2119/jobmanager-pbs-short"
```

Πορεία της εργασίας MPI στο Grid



Επίδειξη!

- Hello World με υποβολή ενός 4-process MPICH job στο HG-01-GRNET

Απορίες – Προβλήματα - Λεπτομέρειες

- Ποιος εκτελεί το mpirun;
  - Σε ποιους κόμβους; Πώς επιλέγονται;
- Shared homes / common storage;
- Εκκίνηση/τερματισμός διεργασιών; Accounting;
  - MPICH-specific λύσεις, με rsh / ssh
  - mpiexec για integration με το Torque
  - CPU Accounting για πολλαπλές διεργασίες
- Υποστήριξη πολλών διαφορετικών Interconnects – υλοποιήσεων MPI;
  - Πού γίνεται η μεταγλώττιση του εκτελέσιμου;

Μελλοντικά...

- Η υποστήριξη MPI για το Grid είναι Work In Progress
  - Υποστήριξη για MPICH over TCP/IP (P4 device)
  - Πρόβλημα με άλλα devices, γιατί χρησιμοποιούνται P4-specific hacks
- Χρειάζονται pre/post-processing scripts
  - Μεταγλώττιση επί τόπου του εκτελέσιμου;

Επιπλέον Θέματα

- Επιλογή επεξεργαστών – ανάθεση σε διεργασίες
  - Θέματα latency κατά την ανταλλαγή μηνυμάτων
  - Memory bandwidth
  - Διαθέσιμης μνήμης
- Υβριδικές αρχιτεκτονικές
  - Συνδυασμός MPI με pthreads / OpenMP για καλύτερη προσαρμογή στην υφιστάμενη αρχιτεκτονική

Βιβλιογραφία - Πηγές

- Writing Message-Passing Parallel Programs with MPI (Course Notes – Edinburgh Parallel Computing Center)
- Using MPI-2: Advanced Features of the Message-Passing Interface (Gropp, Lusk, Thakur)
- http://www.mpi-forum.org (MPI standards 1.1 και 2.0)
- http://www.mcs.anl.gov/mpi (MPICH υλοποίηση)
- comp.parallel.mpi (newsgroup)