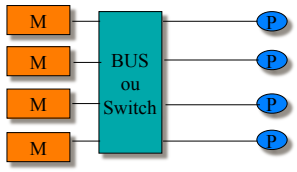
Parallélisme

Passage de messages

Jean-Louis PAZAT
IRISA / INSA

Calculateurs à mémoire partagée



Calculateurs à mémoire partagée

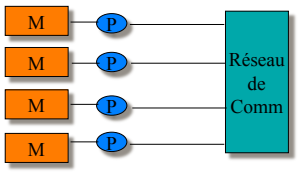
- Avantages
  - conception assez simple
  - utilisation facile
- Inconvénients
  - bande passante mémoire, conflits
  - coût de l'interconnexion
  - nombre de processeurs limité

Calculateurs à mémoire partagée

- Influence sur les langages
  - Notion de processus
    - espace d'adressage protégé
  - Notion de tâches légères
    - espace d'adressage partagé
  - Variables partagées
  - Synchronisations pour empêcher les conflits
    - Base : exclusion mutuelle

4

Calculateurs à mémoire distribuée



5

Calculateurs à mémoire distribuée

- Influence sur le langage
  - notion de processus
  - pas de partage de variables
  - notion de "message"
    - échange de valeurs
    - coopération explicite

6

Réseaux de stations de travail

- Même architecture qu'un calculateur à mémoire distribuée
- Interconnexion de stations de travail
  - réseaux rapides
    - 100Mbit/s, 1 Gbit/s
- Facilement extensibles

Les stations peuvent aussi être des machines parallèles !

7

Plan

- Introduction
- Nommage
- Synchronisation
- Exemples
  - Langage CSP/OCCAM
  - Bibliothèque de passages de messages MPI
- Conclusion

8

Introduction

- Calcul parallèle *vs* distribué/réparti
  - Parallèle :
    - Partage de variables + synchronisations
    - On synchronise pour communiquer
  - Distribué / Réparti :
    - Processus + échange de données ponctuels
      - Instructions explicites pour
        - Envoyer des données à un correspondant
        - Recevoir des données depuis un correspondant
    - On communique pour synchroniser

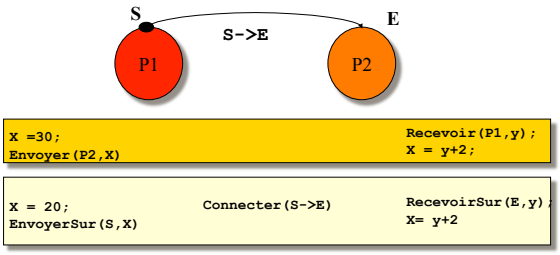
9

Nommage

- Symétrique
  - Emetteur ET recepteurs sont nommés
  - Identificateur du processus correspondant
    ou
  - Identificateur d'un port de communication

10

Nommage



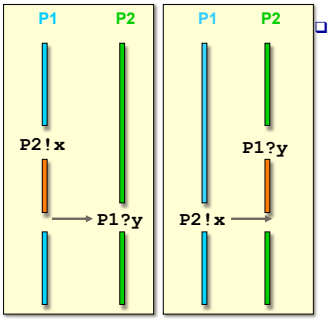
11

Synchronisation

- Emission blocante ou non
  - Jusqu'à émission du message
    ou
  - Jusqu'à réception du message
- Reception blocante ou non
  - Jusqu'à reception du message
    ou
  - Handler prévenant de l'arrivée du message

12

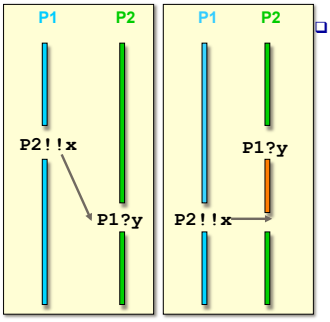
Synchronisation



- Rendez-vous :
  - Émission blocante jusqu'à réception
  - Reception blocante
  - Le premier arrivé attend l'autre
  - Modèle de CSP

13

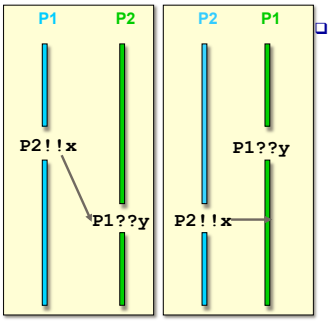
Synchronisation



- Buffers de communication :
  - Émission blocante jusqu'à émission (copie ou envoi)
  - Reception blocante
  - Seul le récepteur attend
  - Modèle de PVM/MPI

14

Synchronisation



- Buffers de communication :
  - Émission blocante jusqu'à émission (copie ou envoi)
  - Reception non blocante
  - Modèle de PVM/MPI

15

Synchronisation

- Emission blocante vs non blocante

```
X =30;
EnvoyerB(P2,X);                  → 30 est envoyé

X = 20;
```

```
X =30;
EnvoyerNB(P2,X);
                        ||       → la valeur courante
X = 20;                             de X est envoyée
```

16

Exemple 1 : CSP

- Langage « historique »
  - Création statique de processus
  - Communication par rendez-vous
  - A été implanté sur des transputers (OCCAM)
  - Commandes :
    - Instructions simples
      - Communications : P!x P?y
      - Instructions habituelles (affectation, …)
    - Instructions composées :
      - Alternative, répétitive (fondées sur les commandes gardées)

17

CSP : commandes gardées

```
Commande gardée :: <Garde> --> <commande>
Garde :: <expr_bool> | <communication>
        |<expr_bool>;<comunication>
```

Exemples :

```
x>=0    P?x        i<=N;P?x
```

Etats :

Garde passable :
booléen vrai et communication possible immédiatement

Garde échouée :
booléen faux ou communication définitivement impossible

Garde *en attente* :
booléen vrai et communication non immédiatement possible mais non définitivement impossible

18

CSP : alternative

`[<commande gardée>{<commande gardée>}*]`

Exemples :

```
[x>0 --> s=1 | x<0 --> s =-1 | x=0 --> s=0]
[x>=0 --> y = x | x<=0 --> y = -x]
[P1?x --> y = x | P2?x --> y = x+1]
```

Sémantique :

Une des gardes passables est choisie

Si toutes les gardes sont échouées c'est une erreur

Si il n'y a pas de garde passable et qu'il y a des gardes en attente on attend.

19

CSP : répétitive

`*[<commande gardée>{<commande gardée>}*]`

Exemples :

```
x=10; *[x>0 --> x=x-1]
x=10; *[x>0; P?a --> x = x - a]
*[P1?y --> y = x | P2?x --> y = x+1]
```

Sémantique :

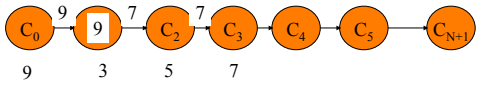
Tant qu'il existe une garde passable, une est choisie

Si toutes les gardes sont échouées la répétitive se termine

Si il n'y a pas de garde passable et qu'il y a des gardes en attente on attend.

20

CSP : recherche de nombres premiers




```
    [C0:gen || C1..N:crible || CN+1:bouchon]
gen : int x = 3; *[x < NMAX ; C1!x --> x = x+2]
crible : int prem, cour; Ci-1 ? prem ;
         *[Ci-1 ? cour -->
            [cour mod prem = 0  --> SKIP
            |cour mod prem != 0 --> Ci+1 ! cour]
          ]
bouchon : *[Ci-1 ? cour --> SKIP]
```

21

Exemple 2: M P I

- *MPI = Message Passing Interface*
- Spécification d'une bibliothèque de passage de messages
- pour
  - les machines parallèles,
  - les clusters de stations de travail (hétérogènes ou pas)
- 2 Versions
  - MPI 1 : très répendue mais limitée
  - MPI 2 : complète mais peu d'implémentations

22

Qu'est-ce qu'il y a dans MPI-1 ?

- Communications point-à-point .
  - plusieurs modes de communication/synchronisation
  - support pour l'hétérogénéité
- Routines de communication collectives
  - Ex: broadcast/reduce/scatter/gather
  - Permettent une mise en œuvre optimisée
  - Opérations pré-définies et utilisateur
  - Communication dans les sous-groupes de processus
  - Communications dans des topologies

23

Qu'est-ce qu'il y a dans MPI-1, suite ?

- Support pour les groupes de processus
- Mécanismes pour séparer les messages de diverses applications (pour le développement de bibliothèques sures)
- Interfaces C et Fortran 77
- Une interface de profiling

24

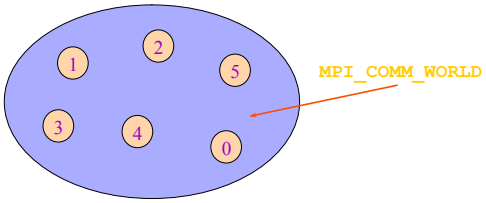
Processus et Groupes dans MPI-1

- MPI-1 suppose que les processus ont été démarrés (aucune routine pour les démarrer ou les arrêter) - style de programmation SPMD
- Chaque processus a son propre flot de contrôle et son propre espace d'adressage
- MPI supporte les groupes de processus dynamiques, mais l'appartenance est statique
  - Les groupes peuvent être créés et détruits
  - Les groupes peuvent se recouvrir
- Le placement des processus sur les processeurs ne fait pas partie de MPI

25

MPI_COMM_WORLD

- Chaque processus possède un rang unique dans la machine virtuelle
- Le communicateur MPI_COMM_WORLD contient tous les processus démarrés (statique)



26

Communicateurs, suite.

- Un ensemble de processus sensés se connaître.
- Chaque communication MPI a lieu par rapport à un communicateur.
- Définit l'étendue d'une communication (il contient les informations de groupe et de contexte).
- MPI_COMM_WORLD est un communicateur prédéfini.
- Les processus peuvent appartenir à plusieurs communicateurs (avec un rang différent pour chaque communicateur).
- Il définit un espace de communication sûr.

27

suite...

- MPI fournit des moyens pour séparer les messages.
  - Contextes de communication (dans les communicateurs).
    - peut être vu comme des étiquettes système.
  - Informations sur les groupes.
  - Etiquettes de messages.
- MPI_COMM_WORLD contient le premier contexte et le groupe de tous les processus démarrés.
- MPI Garanti l'ordre des messages.

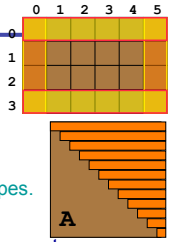
28

Bufferisation Classique de Messages

- Dans les bibliothèques de bas niveau
  - On envoie des zones contiguës de données.
  - On a besoin d'un pré-emballage des données.
- Cela simplifie le processus d'optimisation.
- mais
  - Cela empêche les communications dans les machines hétérogènes ou les réseaux de stations.
  - Le pré-emballage est parfois difficile.

29

Bufferisation MPI



- Vous pouvez avoir
  - Des tableaux de types élémentaires.
  - Des zones contiguës de données.
  - Des blocs de types avec saut.
  - Des tableaux indexés de blocs de types.
  - Des structures générales.
- Les types sont construits récursivement.
- Vous pouvez avoir des types de données dépendants de l'application.

30

Communications MPI simples

```
mpi_send( OUT start_of_buff,
          IN count,
          IN data_type,
          IN dest_rank,
          IN tag,
          IN communicator,
          OUT error_code)
```

```
mpi_recv( OUT start_of_buff,
          IN count,
          IN data_type,
          IN source_rank,
          IN tag,
          IN communicator,
          OUT return_status,
          OUT error_code)
```

`source_rank` et `tag` **peuvent être des jokers (**`MPI_ANY_RANK` et `MPI_ANY_TAG`**)**

31

Sous-ensemble MPI-1

- Il suffit de ces 6 routines pour écrire des programmes MPI (simples).

```
MPI_Init(...)
MPI_Comm_size(...)
MPI_Comm_rank(...)
MPI_Send(...)
MPI_Recv(...)
MPI_Finalize()
```

32

Exemple 1

```
#include <stdio.h>
#include <mpi.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 am %d of %d\n", rank, size);
  MPI_Finalize();
  return(0);
}
```

33

Exemple 2

```
#include <stdio.h>
#include <mpi.h>
int main (int argc, char *argv[])
{
  int i;     /* numero du processus courant */
  int p;     /* nombre total de processus */

  int recu; /* valeur recue par le processus courant */
  MPI_Status stat; /* pour l'initialisation */
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &i); // Rang du processus
  MPI_Comm_size(MPI_COMM_WORLD, &p); // Nombre de processus
if (p<=1) {
    printf("ce programme ne fonctionne qu\"avec au moins 2 processus\n");
    exit(-1);
  }
printf("je suis le pss %d qui demarre parmi %d processus \n",i,p);
```

34

```
if(i==0) {
 int n = 2;/* la valeur envoyée par le 1er processus */
 printf("je suis le pss 0 j'envoie au pss  %d la valeur  %d\n",i+1,n);

 MPI_Send(&n, 1, MPI_INT, i+1, 0, MPI_COMM_WORLD);

 printf("je suis le pss 0 j'attend du processus %d\n",p-1);
 MPI_Recv(&recu, 1,MPI_INT, p-1, MPI_ANY_TAG, MPI_COMM_WORLD, &stat);
 printf("je suis le pss 0 je recoit de %d la valeur %d\n",i+1,recu);
 }
```

35

```
else if (i<p-1) {
 printf("je suis le pss %d j'attend du processus %d\n",i,i-1);

 MPI_Recv(&recu, 1,MPI_INT, i-1, MPI_ANY_TAG, MPI_COMM_WORLD, &stat);
 recu = recu *2;
 MPI_Send(&recu, 1, MPI_INT, i+1, 0, MPI_COMM_WORLD);
  }
```

36

```
else
{
 printf("je suis le pss %d j'attend du processus %d\n",i,i-1);
 MPI_Recv(&recu, 1,MPI_INT, i-1, MPI_ANY_TAG, MPI_COMM_WORLD, &stat);
 recu = recu *2;
 MPI_Send(&recu, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
 }
```

37

```
MPI_Finalize();

return (0);

}
```

38

Modes de Communication

- Envoi **bloquant** jqa emission `MPI_Send/MPI_Bsend`
  L'envoi ne terminera pas tant que le message n'est pas envoyé ou copié dans un buffer système. Aucune supposition sur le côté receveur.
- Envoi **non bloquant** `MPI_Isend/MPI_Ibsend`
  On n'attend pas que l'envoi soit treminé
- Mode **Synchrone (=CSP)** `MPI_Ssend/MPI_Issend`
  L'envoi ne se termine pas tant que le message n'est pas arrivé.
- Mode **Ready** `MPI_Rsend/MPI_Irsend`
  L'utilisateur doit s'assurer qu'une réception a démarré avant l'envoi !!!

39

Communications Non-bloquantes

- Cela ne signifie pas asynchrones!
- Peuvent être très efficaces
  - si vous avez assez de calculs à recouvrir.
  - si votre machine possède un co-processeur de communication.
- Parfois il y a un surcoût système
  - bufferisation supplémentaire.
  - system polling ou interruptions.
- Il y a un compromis à trouver!

40

Compléments pour fonctions de communication non blocantes

- Attente blocante :
  - `mpi_wait(request, status, ierr)`
- Attente non blocante :
  - `mpi_test(request, flag, status, ierr)`
- Attend que toutes les comm. de la liste soient terminées
  - `mpi_wait_all(count, list_requests, status, ierr)`
- Attend qu'au moins une des comm. soit terminée
  - `mpi_wait_some(count, list_requests, count_done, list_index, list_status, ierr)`
- Attend qu'une communication soit terminée
  - `mpi_wait_any(count,list_requests,index, status,ierr)`
- Les mêmes fonctions existent pour tester si ...

41

Communications Collectives

- Transmettre des données entre tous les processus spécifiés par un communicateur.
  - Barrière de synchronisation
  - Diffusion
  - Scatter/gather
  - All-to-all
  - All-gather
  - Réductions (SUM, MAX, MIN, ..., fct utilisateur).
  - Préfixe/scan.

42

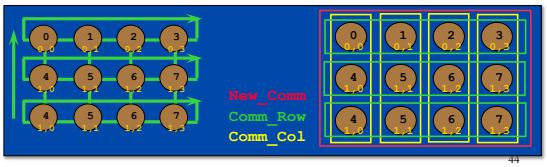
Communications collectives (suite)

- `MPI_Bcast`
  `MPI_Reduce_Scatter`
  `MPI_Gather`
  `MPI_Scatter`
  `MPI_Allgather`
  `MPI_Alltoall`
  `MPI_Allreduce`
  `MPI_Reduce`
  `MPI_Scan`
  `MPI_Gatherv`
  `MPI_Scatterv`
  `MPI_Allgatherv`
  `MPI_Alltoallv`
- Les versions `All` laissent le résultat sur tous les processus.
- Les versions `v` permettent de communiquer des morceaux de vecteurs.
- Les routines `MPI_*reduce*` et `MPI_Scan` autorisent l'emploi de fonctions utilisateur.

43

Topologies virtuelles

- Routines pour structurer les ensembles de processus.
- Pour répondre à cette simple question: *Où sont mes voisins dans cette topologie?*
- Peut aider les implémentations de MPI à arranger les processus sur les processeurs.
- Topologies cartésiennes ou graphes généraux.



44

MPI 1 : Conclusion

- MPI -1 est petit
  - Un petit nombre de concepts.
  - 6 fonctions seulement sont nécessaires !
  - MPI_INIT, MPI_Comm_size, MPI_Comm_rank, MPI_Send, MPI_Recv, MPI_Finalize.
- MPI-1 est assez complet (128 fonctions).
  - Pour avoir des performances et simplifier la programmation.
- MPI-1 a des manques graves
  - Pas de gestion du placement des processus
  - Pas de création dynamique de processus

45

MPI-2

- Compléments de MPI 1
  - Création dynamique de processus
  - Nouveaux modes de communications
  - Entrées/Sorties parallèles

46

Gestion Dynamique de Processus

- `MPI_Comm_spawn`
  démarre quelques processus et établit les communications (communication collective sur `comm`)

```
MPI_Comm_spawn(command, argv, maxp, info, root,
               comm, i_comm, error_codes);
```

- `MPI_Comm_spawn_multiple` démarre plusieurs exécutables.

```
MPI_Comm_spawn(count, array_of_commands,
               array_of_argv, array_of_maxp,
               array_of_info, root, comm, i_comm,
                error_codes);
```

- Il est conseillé de démarrer tous les processus en même temps.

47

Communications One-sided

- Ajout de communications put et get au paradigme de rendez-vous traditionnel.
  - Le transfert de données nécessite un appel de bibliothèque dans un seul processus.
  - meilleures performances sur certains systèmes (support matériel, machines à mémoire partagée).
  - Plus simple pour certaines applications (schémas de communication irréguliers ou dynamiques, compilateurs data-parallèles, ...).
- Quand même un modèle à mémoire distribuée.

48

MPI I/O

- Les E/S parallèles sont un point crucial pour le développement d'applications parallèles.
  Goulot d'étranglement pour certaines applications.
- Les E/S parallèles signifient un accès fichier concurrent à partir de processus multiples dans un programme parallèle.
- MPI I/O peut être considéré commes les E/S Unix plus de nombreuses autres fonctionnalités.
- Démarré à l'IBM T.J.Watson Research Center en 93.

49

Le TP

- Propagation de la chaleur sur une plaque rectangulaire
  - les bords sont en contact avec un milieu uniformément chaud.
  - La plaque est discrétisée sous forme d'un matrice de points.
  - Une matrice représente la température de la plaque en chacun de ses points.
- On va calculer de manière itérative la répartition de la chaleur sur cette plaque

50

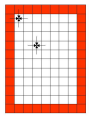
- Calcul à effectuer :

$$T_{t+1}(i,j) = (T_t(i,j+1)+T_t(i,j-1)+T_t(i+1,j)+T_t(i-1,j)+T_t(i,j))/5$$

- Le calcul se termine lorque l'écart moyen est inférieur à un certain seuil $\Delta = \Sigma_{i,j} |T_{t+1}(i,j) - T_t(i,j)|$
- Pour simplifier l'écriture de l'algorithme, les bords de la matrice représentent le milieu extérieur. Il n'y a ainsi pas de cas particulier à traiter.

51

L'algorithme

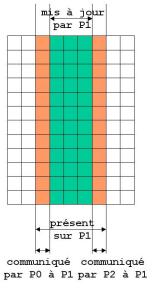


```
double T[N+2][M+2], T1[N+2][M+2];
Initialiser (T=0 sauf les bords = MAX) ;
Faire
   delta=0;
   Pourtout i=1..N,j=1..M :
      T1[i][j] =
      (T[i][j+1]+T[i][j-1]+T[i+1][j]+
       T[i-1][j]+T[i)[j])/5;
      delta = delta + |T_{t+1}(i,j)-T_t(i,j)| ;
      T = T1 ;
Jqa ( delta < seuil )
```

Attention : sur les bord (i=0 ou j=0 ou i=N+1 ou j=M +1) la valeur reste inchangée : il s'agit du milieu extérieurieur.

52

Exécution en distribué



- On « découpe la matrice en tranches verticales
  - Chaque processeur a une tranche
  - Il y a un recouvrement entre les tranches

53

Conclusion

- Messages :
  - adapté au calcul distribué
- Bibliothèques :
  - Simples a utiliser (cf MPI-1)
- Difficultés :
  - Liées à l'aspect distribué
    - Pas de variables globales
    - Pas de temps global

54