Super Calculateur HPC, pour High Performance Computing

Certaines applications ont besoin de faire un grand nombre de calcul, souvent pour des problèmes de simulation.

météorologie,

physique (nucléaire http://www.irisa.fr/orap/Publications/Forum9/Gonnord.pdf)

sismiques,

automobiles. avions, ...

Historique / Super Calculateur

Quelques Dates :

1642 : Pascal Pascaline (addition, soustractions)

1884 : Herman Hollerith crée une tabulatrice à cartes perforées (recensement)

1896 : Herman Hollerith, crée Tabulating Machine Corporation (TMC)

1924 : TMC devient International Business Machine (IBM)

(http://histoire.info.online.fr/super.html)

Historique / Super Calculateur

Quelques Dates :

1935 : IBM commercialise l'IBM 601 à 1500 ex.

(comptabilité, scientifiques – relais + cartes)

39-45 : La bombe, le colossus (décryptage)

1946 : Création de l'ENIAC (Electronic Numerical Integrator and Computer)

30 tonnes, 72 m2 140 kilowatts. Horloge : 100 KHz. 330 multiplications par seconde

Historique / Super Calculateur

Quelques Dates :

1951 : La Compagnie des Machines Bull réalise son premier ordinateur : le Gamma 2.

1951 : Mise au point du tambour de masse magnétique ERA 1101. Il s'agit de la première mémoire de masse. Capacité : 1 Mbits.

1951 : Invention du premier compilateur A0

Historique / Super Calculateur

Quelques Dates :

1955 : IBM lance l'IBM 704 premier co-processeur mathématique 5 kFLOPS

1955 : Premier réseau informatique à but commercial : SABRE (Semi Automated Business Related Environment) 1200 téléscripteurs

1958 : Lancement du premier ordinateur commercial entièrement transistorisé, le CDC 1604, développé par Seymour Cray.

Historique / Super Calculateur

Quelques Dates :

1964 : Lancement du super ordinateur CDC 6600 développé par Seymour Cray

1976 : Cray Research Inc. architecture vectorielle : le CRAY I.

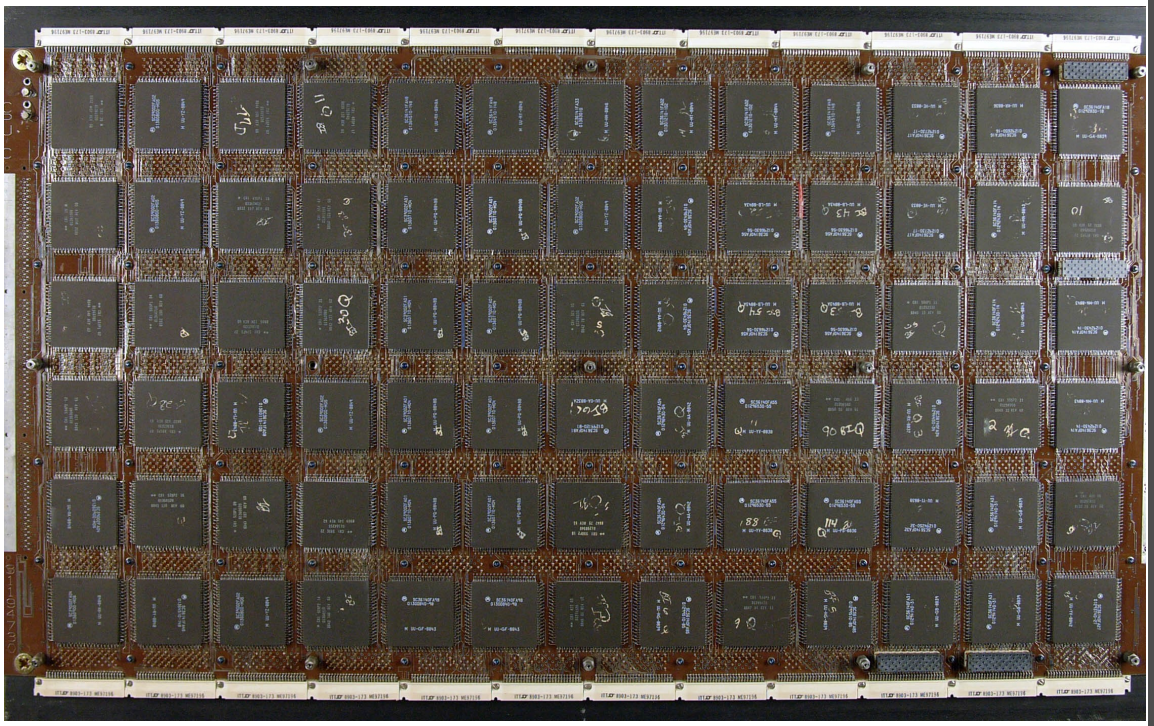
1982 : Annonce du Cray X-MP, le premier super-ordinateur Cray multiprocesseur.

FLOPS

FLoating point Operations Per Second

megaflop	10^6
gigaflop	10^9
teraflop	10^12
petaflop	10^15
exaflop	10^18
zettaflop	10^21
yottaflop	10^24

Le Cray XMP un super calculateur de type vectoriel 1984-1986 – 800 MFLOPS(wikipedia)



Le Cray 2 un des ordinateurs le plus puissant de 1985 à 1989 1,9 GFLOP (wikipedia)

Processeur Vectoriel / Vector processor

Processeur conçu pour effectuer des opérations mathématiques simultanéments sur des données différentes.

Souvent pour effectuer du calcul matriciel

Calcul Parallèle

- utilisation de machines ou de processus séparés,
- ces éléments communiquent par échange d'information.

Types de parallèlisme

Toutes ces méthodes utilisent différentes données pour chaque processus :

SIMD Data-parallel : Mêmes opérations sur des données différentes.

(SIMD, Single Instruction Multiple Data),

SPMD : Même programme avec des données différentes,

MIMD : Différents programmes avec des données différentes.

Loi d'Amdhal

Gene Myron Amdahl (16/11/1922,)
(architecte IBM - Amdahl Corporation)

$$\frac{1}{((1-P)+\frac{P}{N})}$$

P : proportion du code parallèle

N : nb processeurs

Loi d'Amdhal

Quelques exemples :

P=0,5

N=2 -> 1,33

N=4 -> 1,6

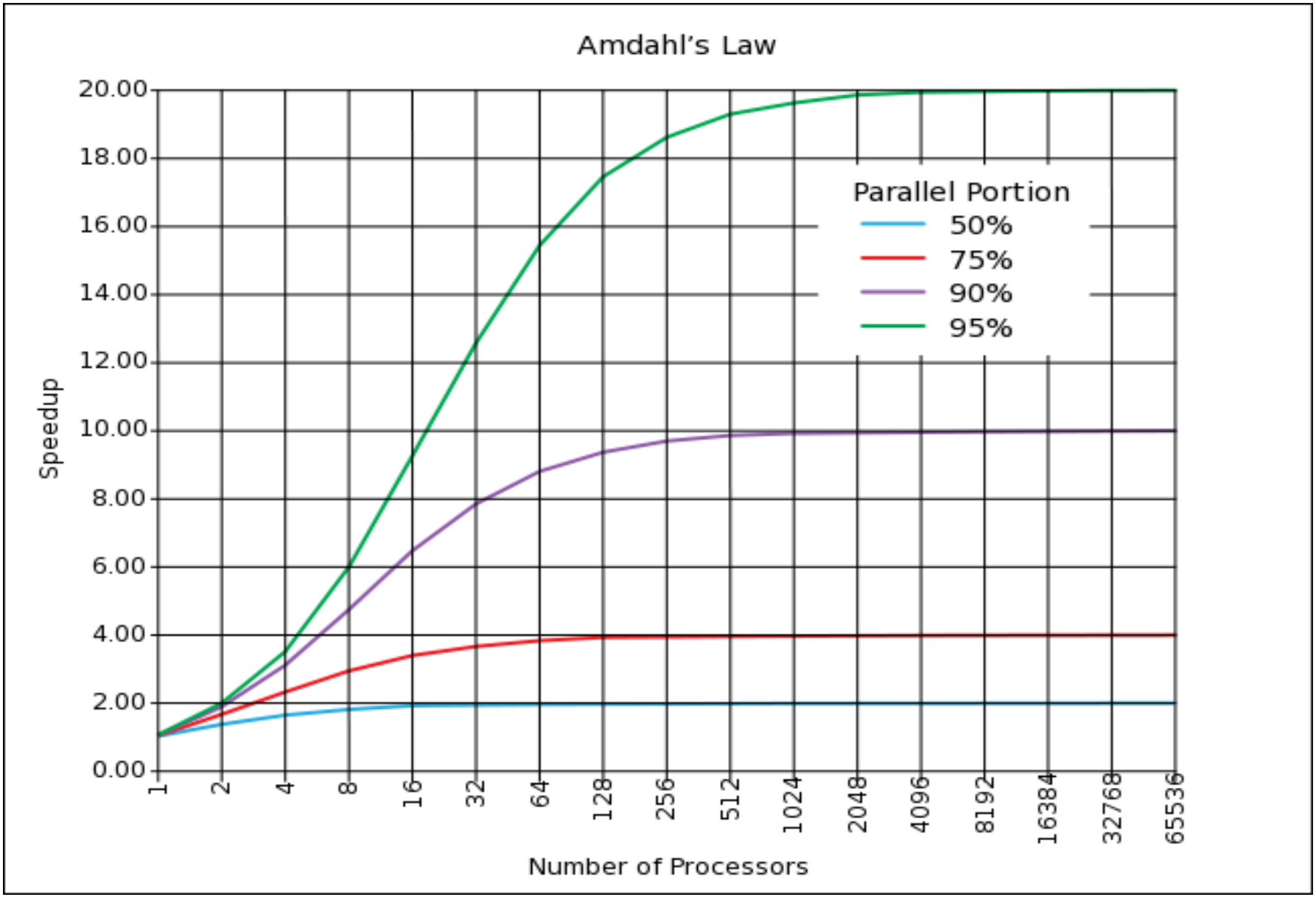
N=8 -> 1,78

N=16 -> 1,88

N=32 -> 1,94

N=64 -> 1,97

N=128 -> 1,98


Amdahl's Law
Speedup
Number of Processors
Parallel Portion
50%
75%
90%
95%
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536

Historique / Super Calculateur

1986 : La société Thinking Machines la Connection Machine CM-1 pouvant comporter jusqu'à 65536

1993 : Connection Machine CM-5 (65,5 GFLOPS)

mi-90 : chute du marché des super-calculateur

2002 : NEC Earth Simulator (35.86 TFLOPS)

(Le plus rapide 2002-2004 5120 processeurs)

Historique / Super Calculateur

2004 : IBM Blue Gene/L (70.72 TFLOPS)

(dual power-pc)

2007 : IBM Blue Gene/L (478.2 TFLOPS)

2008 : IBM Roadrunner (1.026 PFLOPS)

(12,960 IBM PowerXCell et 6,480 AMD Opteron dual-core)

TOP 500

http://www.top500.org/lists/2008/11

1 - Roadrunner 1,105 PFLOPS Etats-Unis DOE

2 - Jaguar - Cray XT5 1,059 PFLOPS Etats-Unis Oak Ridge

3 - Pleiades - SGI Altix 0,487 PFLOPS Etats-Unis NASA

4 - BlueGene/L 0,478 PFLOPS Etats-Unis DOE

5 - Blue Gene/P 0,450 PFLOPS Etats-Unis Argonne

6 - Ranger SunBlade 0,433 PFLOPS Etats-Unis Univ. Texas

7 - Franklin - Cray XT4 0,266 PFLOPS Etats-Unis DOE-Berkeley

8 - Jaguar - Cray XT4 0,205 PFLOPS Etats-Unis Oak Ridge

9 - Cray Red Storm 0,204 PFLOPS Etats-Unis Sandia

10 - Dawning 5000A 0,180 PFLOPS Chine Shangai

11 - Blue Gene/P 0,180 PFLOPS Allemagne

14 - Jade - SGI Altix – Xeon 0,128 PFLOPS France

TOP 500

Rmax - Valeur maximale LINPACK

Rpeak – Performance maximale théorique

Puissance en KW

LINPACK

http://www.netlib.org/linpack/

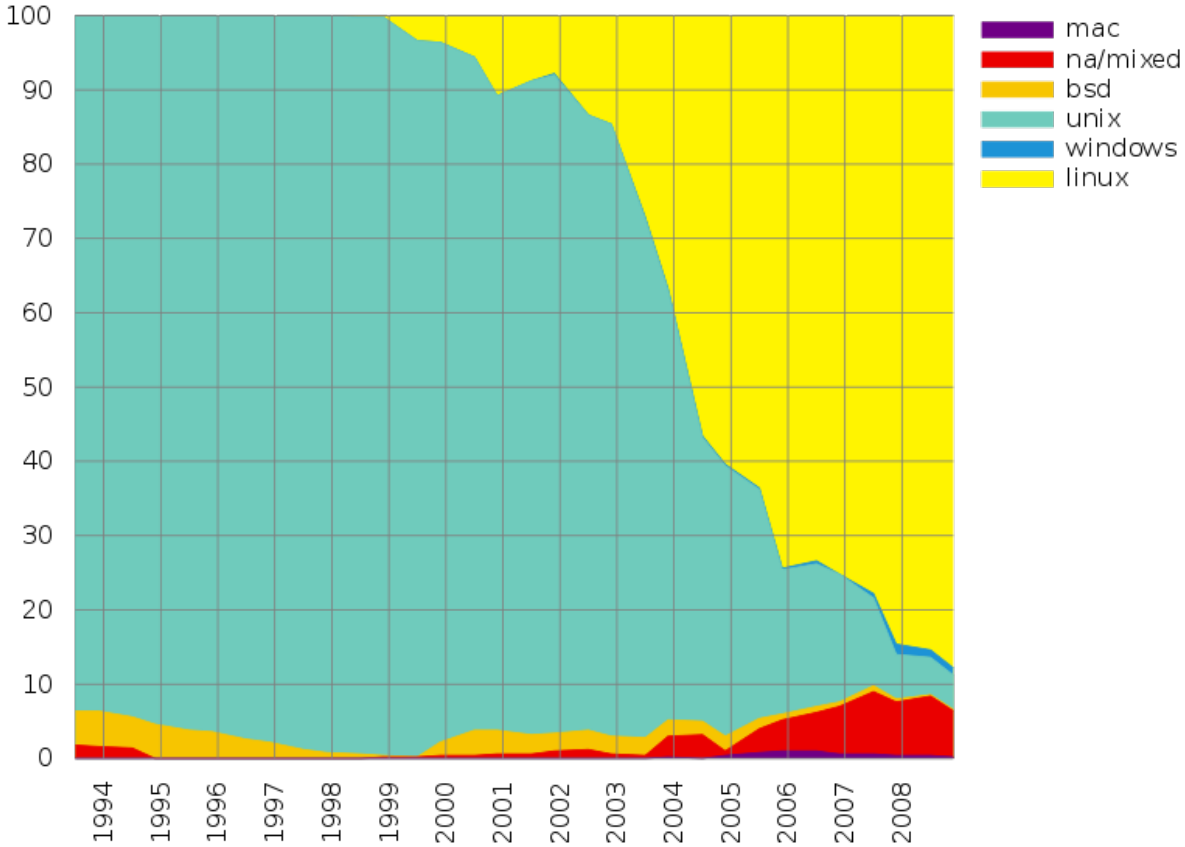
C'est une bibliothèque fortran de calcul linéaire

remplacé par LAPACK

http://www.netlib.org/lapack/

Système d'exploitation

Au début des systèmes par machine puis souvent UNIX, puis LINUX



CISC

Complex instruction set computer

Processeurs PDP11, 680x0, x86

Ce sont des processeurs avec de larges jeux d'instruction et des modes d'adressages de la mémoire complexes

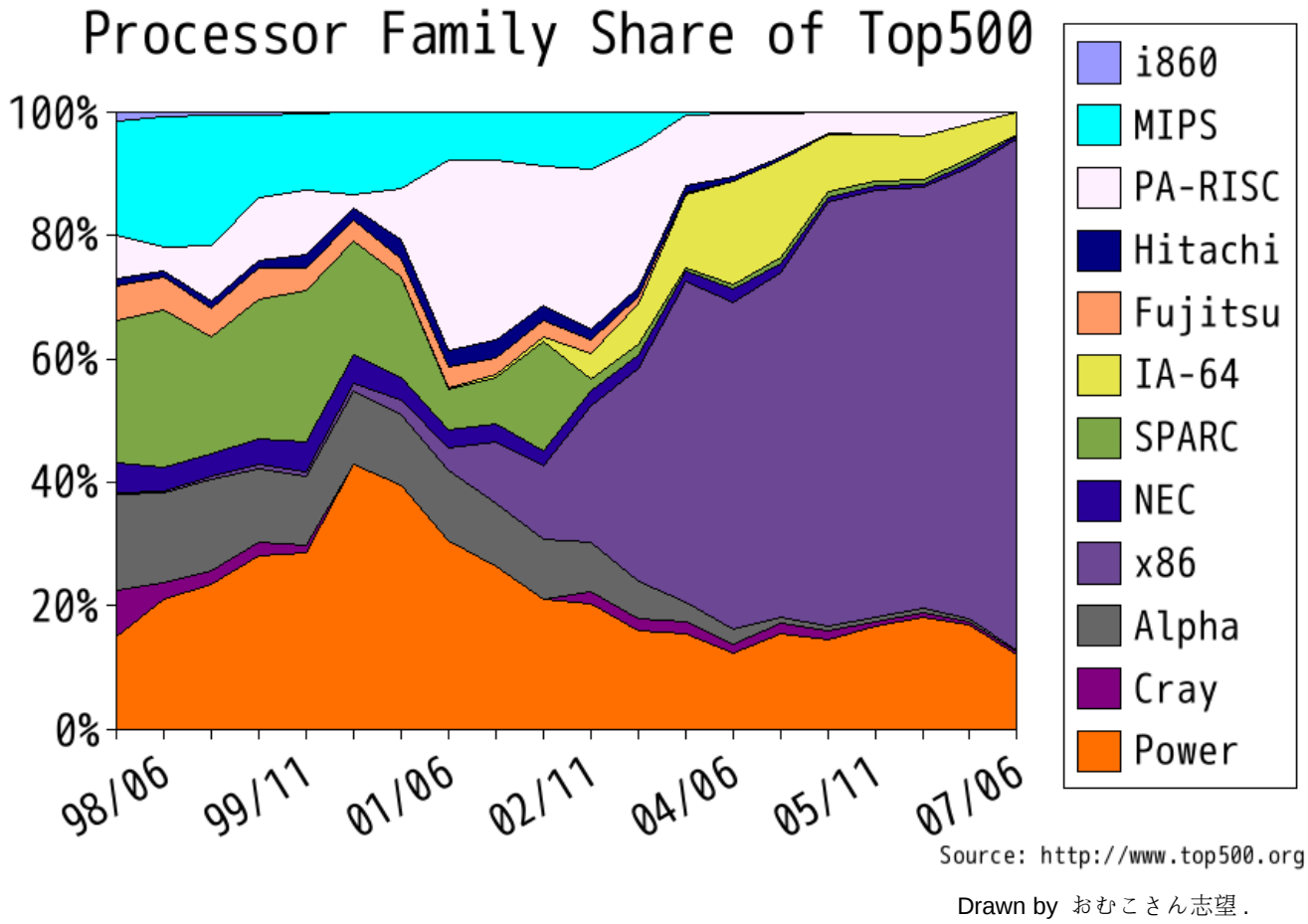
Souvent des mécanismes de microprogrammation

RISC

Reduced instruction set computer

- Un nombre réduit d'instruction avec le même nombre de cycle,
- Un mécanisme de pipeline pour commencer une instruction par cycle
- Moins d'instruction donc optimisation du compilateur plus simple
- Inconvénient code assembleur peu lisible et code exécutable plus important

Processeurs



Processeurs

- i860 – 1989 – 1995 Intel RISC (25-50 MHz)
- MIPS - Microprocessor without Interlocked Pipeline Stages 1985 – 2002 (8 MHz–1 GHz)
- PA-RISC –Precision Architecture HP 1989-2005 (66MHz – 1,1 GHz)
- IA-64 – Itanium, Itanium 2 Intel/HP 2001-2007 (733 MHz -1,66 GHz)
- SPARC Scalable Processor Architecture – TI, Atmel, Fujitsu 1987-2009 (14,2MHz – 2,5 GHz)

Processeurs X86

8086, 8088, 80186, 80286 (MMU) -16 bits

1978- ...

386,486, Pentium, Pentium MMX 32 bits

1985- ...

Pentium Pro, AMD K5, K6 32 bits (PAE)

1995- ...

Athlon64, Opteron, Pentium 4 64 bits

2003- ...

Intel Core 2, Phenom (multi-core)

2006- ...

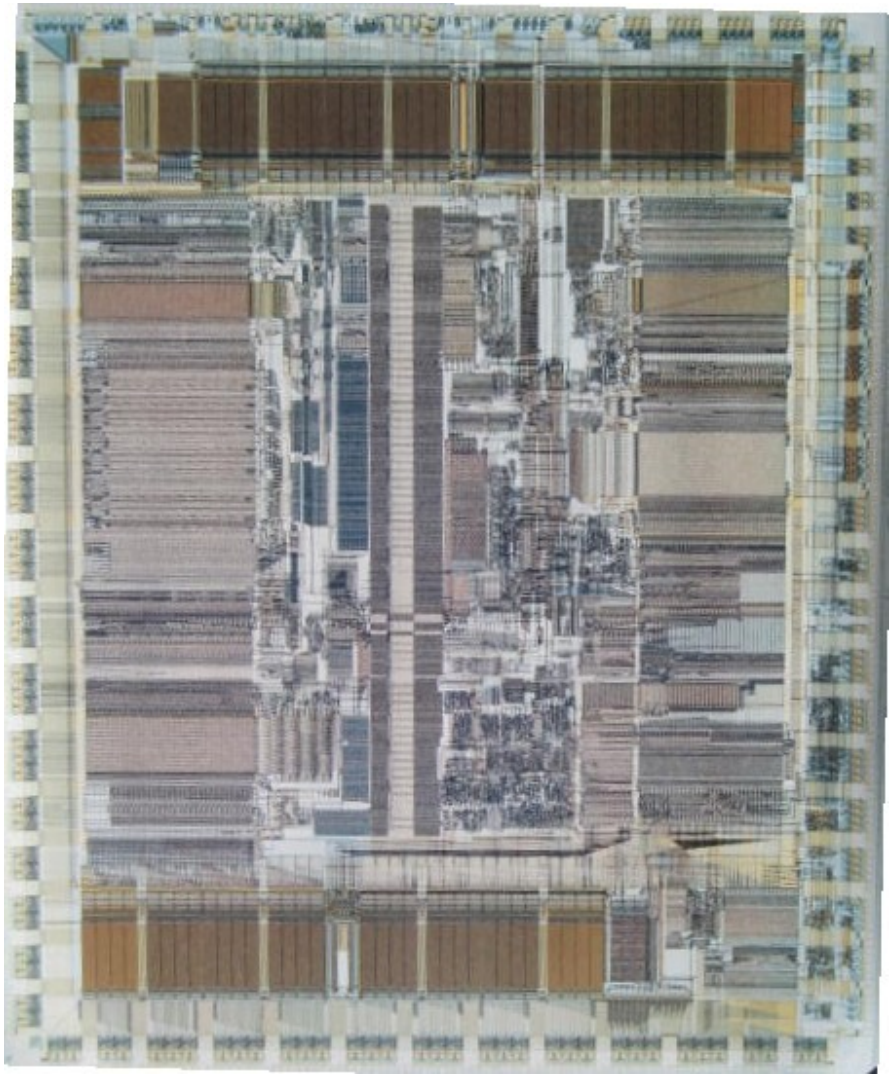
Alpha

Processeur DEC Alpha

RISC 64 bits

1992-2004

66 MHz – 1,3 GHz



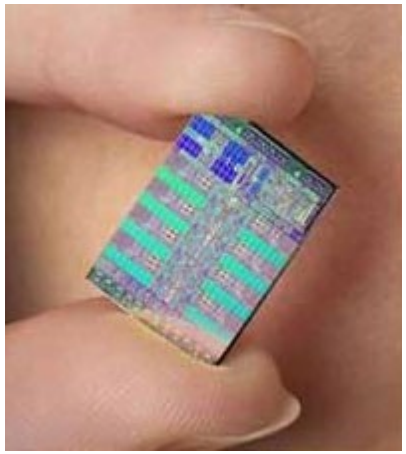
CELL

(Sony, Toshiba, IBM) vers 2006

Un processeur conçu pour la Playstation 3

Un power processing element et

8 Synergistic Processing Elements (25.6 GFLOPS en simple précision)



CELL

PowerXCell 8i

Une version adaptée pour les super calculateurs

qui fournit au total 102 GFLOPS en double précision

GPU
Graphics processing unit

Processeur dédié pour le graphisme,

Calcul en virgule flotante (3D)

GPGPU
General Purpose GPUs

Utilisation d'une carte graphique pour faire du calcul (GeForce 8800 GTX à 518.43 GFLOPS performance théorique)

Repère

Cortex humain 100 (10^{11}) milliards de neurones et 10^{14} synapses

La puissance pour simulation est d'environ 36,8 petaFLOPS et une mémoire de 3,2 petabytes

Calcul Parallèle/Répartis

Quelques éléments :

OpenMP

MPI

BOINC

Folding@home

...

Approche MIMD

Utilisation de MPI « The Message Passing Interface »

Permet d'utiliser le parallélisme sur un grand nombre d'architecture et en particulier les réseaux de stations ou de PC.

Communication

Les données doivent être echangées entre processus :

Coopérative :

les processus doivent être d'accord pour réaliser un transfert d'information,

Supervisé :

un processus réalise les échanges de données.

Historique

Depuis longtemps il existe des bibliothèques de calcul parallèle pour super-calculateur :

- Intel's NX,
- PVM (Parallel Virtual Machine),
- MPI (Message Passing Interface Standard),
- Fortran 90,
- ...

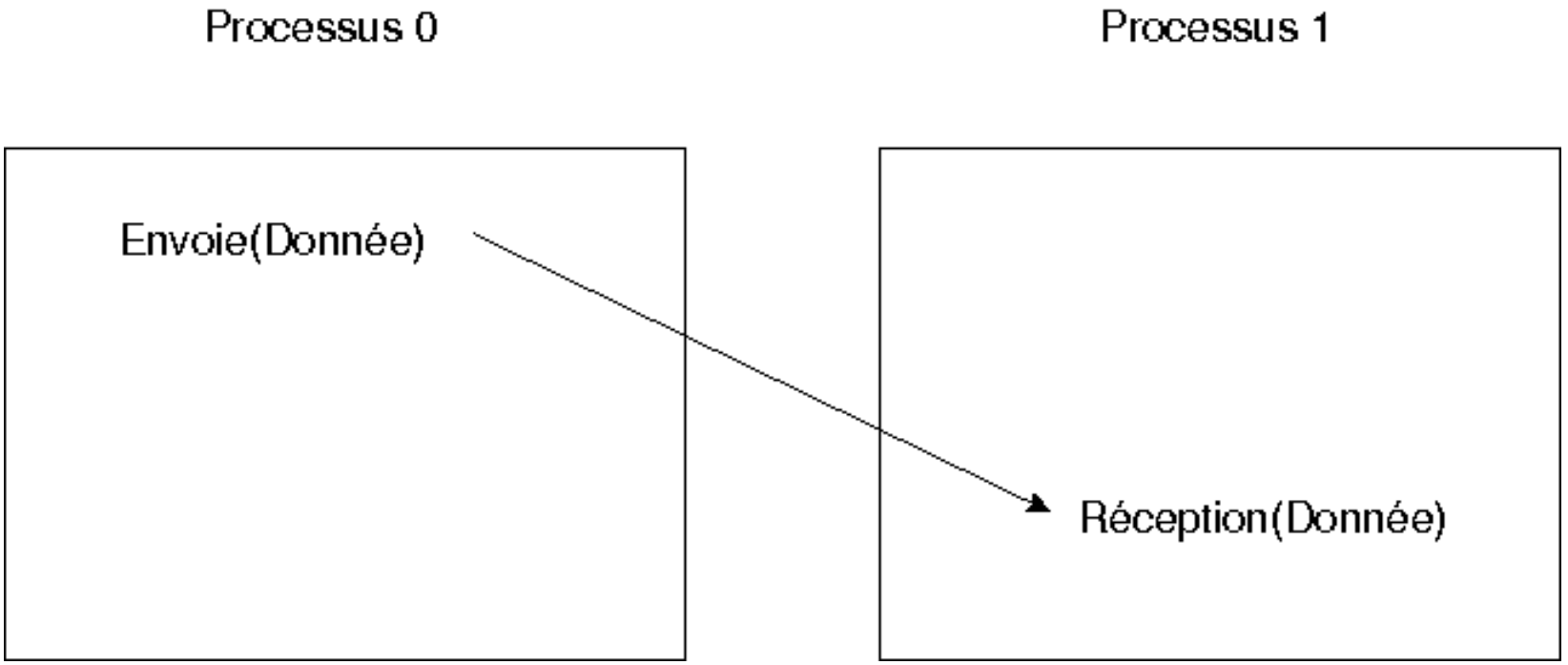
Communication Coopérative

L'envoie de message est une approche qui permet un échange d'information de façon coopérative.

Les données doivent être émise et reçue de manière explicite.

Un avantage est que la mémoire du récepteur est modifié qu'avec la participation du processus récepteur.

Communication Coopérative



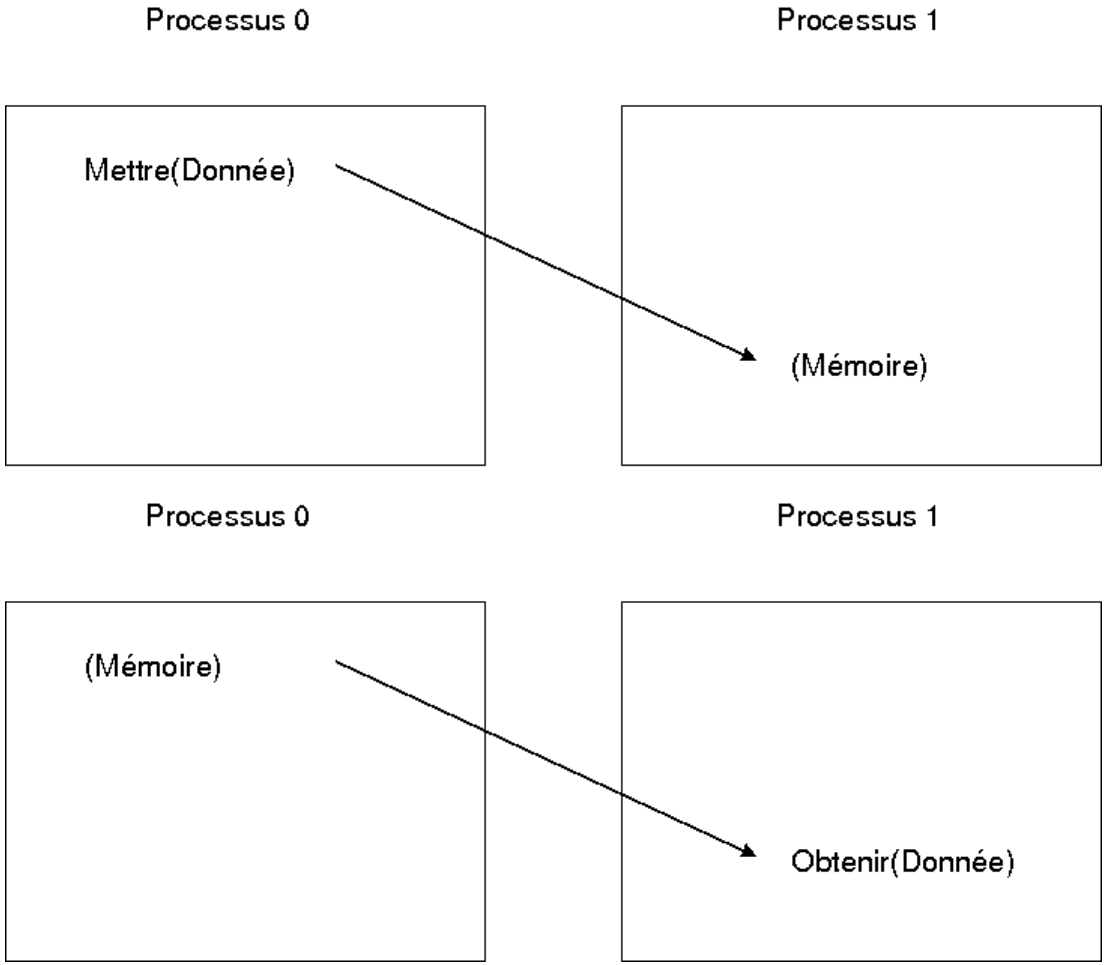
Communication unilatéral

Un processus écrit directement des données dans la mémoire d'un autre processus,

L'avantage est qu'il n'y a pas d'attente entre les processus,

L'inconvénient on peut avoir une perte de la cohérence des données.

Communication unilatéral



Aspect matériels (Hardware)

Les architectures :

Mémoire distribuée (ex. Paragon, IBM SPs, Workstation network),

Mémoire partagée, (ex. SGI Power Challenge, Cray T3D)

Qu'est ce MPI ?

La spécification d'une librairie par envoie de message,

- un modèle d'envoie de message,
- ce n'est pas les spécifications d'un compilateur,
- ce n'est pas un produit,

Conçu pour les ordinateurs parallèles, les clusters d'ordinateurs, et les réseaux hétérogènes,

Qu'est ce MPI ?

C'est un système complet,

Il est conçu pour permettre le développement de librairies,

Il permet l'utilisation de systèmes parallèles pour :

des utilisateurs,

des développeurs de librairies,

des développeurs d'outils.

Avantages

- Système modulaire,
- Bonne performance,
- Portable,
- Peut fonctionner avec des machines hétérogènes,
- Permet différentes topologies de communications,
- Il existe des outils de mesures des performances.

Qui a développé MPI ?

Vendeurs,

- IBM, Intel, TMC, Meico, Cray, Convex, Ncube.

Développeurs (de librairies),

- PVM, p4, Zipcode, TCGMSG, Chameleon, Express, Linda,

Modèle séquentiel

- Le programme est exécuté par un seul processeur
- Toutes les variables et constantes du programme sont allouées dans la mémoire centrale du processeur

Modèle par échange de messages

Plusieurs processus travaillant sur des données locales.

Chaque processus a ses propres variables et il n'a pas accès directement aux variables des autres processus

Le partage des données entre processus se fait par envoi et réception explicites de messages

Les processus peuvent s'exécuter sur des processeurs différents ou identiques.

Modèle par échange de messages

SPMD Single Program Multiple Data

- Le même programme s'exécute pour tous les processus
- Les données sont échangées par messages
- Les données sont en générales différentes pour chaque processus

Modèle par échange de messages

Avantages :

Peut être implémenter sur une grande variété de plates-formes :

Calculateur à mémoire distribuée, à mémoire partagée, réseau de stations mono ou multi-processeurs, station mono-processeur

Permet en général un plus grand contrôle de la localisation des données et par conséquent une meilleure performance des accès aux données.

Programmation parallèle

Permet en général d'obtenir de meilleures performances que la version séquentielle

Traiter de façon performante un volume de données plus important (répartition des données)

Pour obtenir de bonnes performances :

Équilibrer les charges sur chaque processus

Minimiser les communications

Minimiser les parties non parallèle

Performance des communications

Le mode de transfert :

- Avec recopie ou sans recopie des données transferrées
- Bloquant et non bloquant
- Synchrone et non synchrone

Performance des communications

Les performances du réseau :

- Latence : temps mis pour envoyer un message de taille nulle
- Débit ou bande passante
- le choix d’un mode de transfert adapté et effectuer des calculs pendant les communications

Caractéristiques de MPI

Générale,

- Les communications se font dans un contexte et un groupe pour la sécurité,
- Le système supporte le mécanisme des Thread.

Communication point-à-point,

- Communication de structure et de type de données, pour des architectures hétérogènes.
- La gestion de nombreux modes de communication (bloquant, non bloquant, ...).

Quand utiliser MPI ?

Pour :

- on souhaite un programme parallèle portable,
- on veut écrire une librairie de code parallèle,
- on souhaite un système parallèle spécifique (dynamique, ...).

Quand utiliser MPI ?

Contre :

- vous pouvez utiliser Fortran 90 (code Fortran),
- on n'a pas besoin de parallèlisme,
- on peut utiliser une librairie,
- dans certain cas on peut utiliser OpenMP

Routines de base de MPI

Gestion des communications

- Initialisation et fin des communications
- Création de groupe et de topologie virtuelle

Routines de communications entre deux processus

Routines de communications collectives

Création de types dérivés

Programme MPI

include « fichier entête MPI »

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

Déclarations des variables

Initialisation de MPI (par chaque processus)

…

Même programme exécuté par tous les processus

…

Fermeture de MPI (par chaque processus)

Programme MPI minimal

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{ /* Initialisation */
MPI_Init(&argc, &argv); /* code */
MPI_Finalize(); /* terminaison */
return(0);
}
```

Programme MPI minimal

```
MPI_Init(&argc, &argv);
```

Initialisation de MPI et passage des paramètres

```
MPI_Finalize();
```

Terminaison de MPI

Toutes les fonctions non-MPI sont locales, par exemple un printf sera exécuté sur chaque processus.

Compilation

Il est conseillé de créer un Makefile,

Il existe aussi la commande mpicc -o exec exec.c pour compiler un fichier.

Options :

-mpilog Génère un fichier de log des appels MPI

-mpitrace

-mpianim

Execution d'un programme MPI

mpirun -np 2 executable

-v

Option verbose, le système indique les opérations effectuées.

-help

Indique toutes les options possibles,

Communicateur MPI

Un communicateur définit un ensemble de processus succeptibles de communiquer entre eux

MPI_COMM_WORLD est le communicateur par défaut constitué de tous les processus de l'application:

Problème de la programmation parallèle

Combien sommes-nous a travailler ?

MPI_Comm_size

Indique combien il y a de processus,

Qui suis-je (pour répartir le travail) ?

MPI_Comm_rank

Indique le numéro du processus, entre 0 et le nombre de processus - 1.

Example

```
MPI_Comm_rank( MPI_COMM_WORLD,
  &rank);
MPI_Comm_size( MPI_COMM_WORLD,
  &size);
printf("Bonjour-Hello World je suis
  le %d parmi %d\n",rank, size);
```

Chaque processus va indiquer son numéro et le nombre total de processus.

Envoie et réception de messages

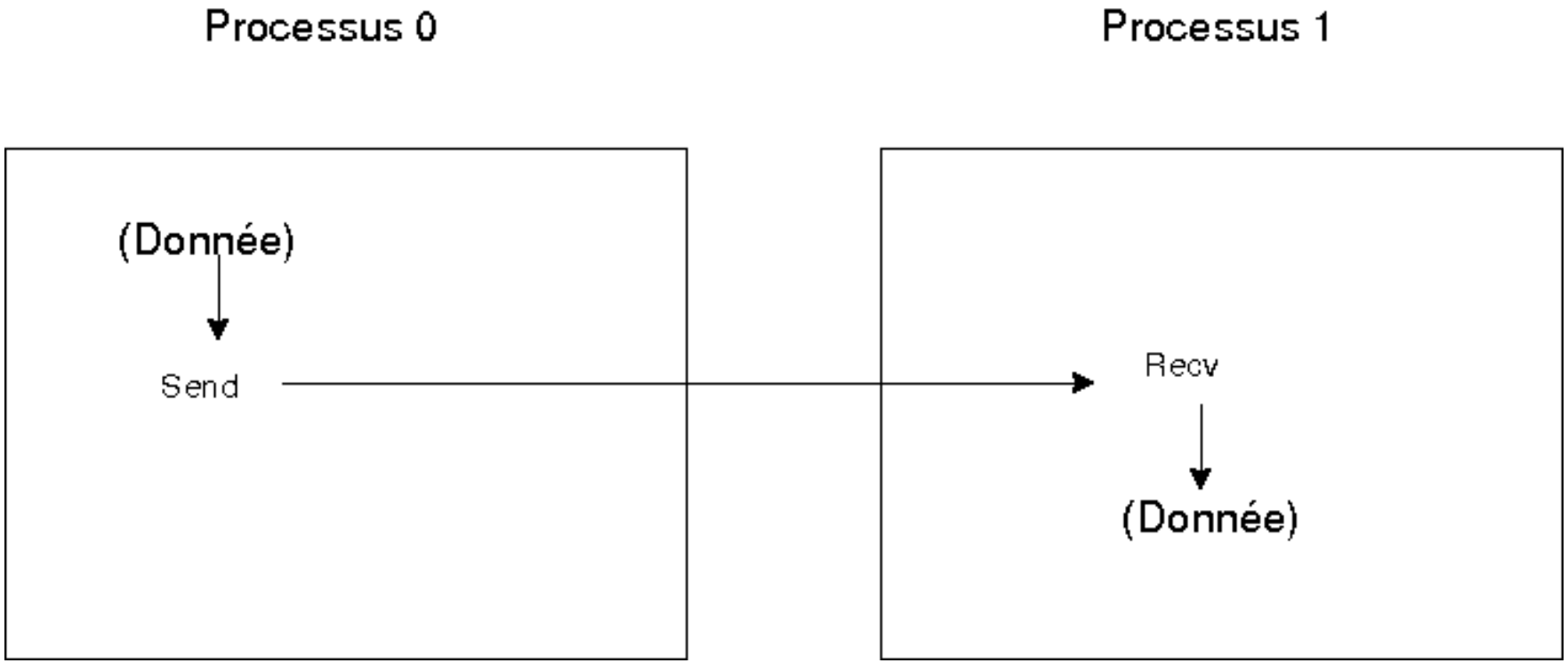
Problèmes :

a qui envoie-t-on les données ?

qu'est-ce qu'on envoie ?

comment les identifier à la réception ?

Envoie et réception de messages



Utilisation d'un Buffer

Pour MPI on utilise une adresse de départ, un type, un nombre d'éléments, les types sont :

- les types élélmentaires (C et Fortran),
- les tableaux a une dimension des types élémentaires,
- ...,

Communication Simple (Envoie)

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

- void * buf adresse de début de buffer,
- count nombre d'éléments,
- MPI_Datatype datatype type de chaque élément,
- int dest processus destinataire,
- int tag identifieur du message,
- MPI_Comm comm groupe de communication.

Communication Simple (Réception)

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

void * buf adresse de début de buffer,

count nombre d'éléments,

MPI_Datatype datatype type de chaque élément,

int source processus de provenance,

int tag identifieur du message,

MPI_Comm comm groupe de communication,

MPI_Status *status information d'état

Communication Simple (Brodcast)

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

Permet de répartir des données.

- void * buf adresse de début de buffer,
- count nombre d'éléments,
- MPI_Datatype datatype type de chaque élément,
- int root identificateur de l'expéditeur,
- MPI_Comm comm groupe de communication,

Communication Simple (Reduce)

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

- void* sendbuf adresse de début de buffer,
- void* recvbuf adresse de début de buffer,
- int count nombre d'éléments,
- MPI_Datatype datatype type des données,
- MPI_Op op opération à effectuer,
- int root processus root,
- MPI_Comm comm gr. de communication.

Communication Simple (Reduce)

Opérateurs :

- MPI_MAX maximum,
- MPI_MIN minimum,
- MPI_SUM sum,
- MPI_PROD product,
- MPI_LAND logical and,
- MPI_LOR logical or,
- ...

Types de données

Types :

- MPI_CHAR (char)
- MPI_INT (int)
- MPI_FLOAT (float)
- MPI_DOUBLE (double)
- ...

Informations sur les messages

```
MPI_Status status;
MPI_Recv(..., &status);
```

Si MPI_ANY_TAG ou MPI_ANY_SOURCE alors status.MPI_TAG et MPI_Get_count obtenir des informations sur l'état de la communication

Informations sur les messages

```
MPI_Get_count(MPI_Status *status,
  MPI_Datatype datatype, int
  *count);
```

MPI_Get_count permet d'obtenir le nombre d'éléments reçus.

MPI_Status *status le status de la communication,

MPI_Datatype datatype le type des éléments,

int *count le nombre d'éléments reçus.

Informations sur les messages

```
MPI_Status status;
MPI_Recv(..., &status);
```

Si MPI_ANY_TAG ou MPI_ANY_SOURCE alors status.MPI_TAG et MPI_Get_count obtenir des informations sur l'état de la communication

Informations sur les messages

```
MPI_Get_count(MPI_Status *status,
  MPI_Datatype datatype, int
  *count);
```

MPI_Get_count permet d'obtenir le nombre d'éléments reçus.

MPI_Status *status le status de la communication,

MPI_Datatype datatype le type des éléments,

int *count le nombre d'éléments reçus.

MPI_2

- Gestion dynamique des processus
- Entrées/Sorties parallèles
- Interfacage c++, fortran 90
- Communication mémoire à mémoire
- Possibilité de définir des interfaces externes (processus léger)

MPI_2 I/0

- Les applications qui font des calculs volumineux font en général également de nombreuses entrées-sorties (données à traiter et résultats)
- Le traitement de ces entrées-sorties représente une part importante dans le temps globale l'application
- L'optimisation des E/S d'une application parallèle se fait :
  - Par leur parallèlisation
  - Et /ou leur recouvrement par des calculs

MPI_2 I/0

Propriétés d'accés aux fichiers

- Positionnement explicite
  - Nb octets depuis le début du fichier
- Positionnement implicite pointeur
  - Individuel pour un processus
  - Commun pour tous les processus
- Synchronisation
  - Bloquant
  - Non bloquant

MPI_2 I/0 – E/S parallèle

C'est à dire :

- Entrée/Sortie réalisée par plusieurs tâches ou processus
  - A partir d'un même espace mémoire
  - A partir d'espaces mémoire séparés
- Avantages
  - Éviter de créer des goulets d’étranglement
  - Techniques au niveau de la programmation permettant de gérer les lectures/écritures asynchrones
  - Performances: opérations spécifiques prises en charge par le système

MPI_2 I/0 – E/S parallèle

C'est à dire :

- Fermeture et ouverture de fichier sont des opérations collectives associées à un communicateur
- Chaque opération de transfert est associée à un type de donnée
  - Type de base MPI
  - Type dérivé MPI: transfert d'un zone discontinue de la mémoire vers un stockage continu sur fichier ou inversement

MPI_2 I/0 – E/S parallèle

- Descripteur(file handle): objet caché créé à l'ouverture et détruit à la fermeture du fichier
- Pointeurs tenu à jour automatiquement par MPI (associé à chaque file handle)
  - Individuel : propre à chaque processus ayant ouvert le fichier
  - Partagé : commun à chaque processus ayant ouvert le fichier

MPI_2 I/0 – Fonctions

- MPI_File_open(): associe un descripteur à un fichier
- MPI_File_seek(): déplace le pointeur de fichier vers une position donnée
- MPI_File_read(): lit à partir de la position courante une quantité de données(pointeur individuel)

MPI_2 I/0 – Fonctions

- MPI_File_write(): écrit une quantité de données à partir de la position courante (pointeur individuel)
- MPI_File_close(): élimine le descripteur
- MPI_File_sync(): force l'écriture sur disque des buffers associés à un descripteur

MPI_2 I/0 – Fonctions

Valeur courante du pointeur :

- Individuel:

MPI_File_get_position()

- Partagé:

MPI_File_get_position_shared()

Positionner le pointeur

- Individuel: MPI_File_seek()
- Partagé: MPI_File_seek_shared()

MPI_2 I/0 – Fonctions

Trois manières de positionner le pointeur :

- Une valeur absolue: MPI_SEEK_SET
- Une valeur relative : MPI_SEEK_CUR
- À la fin d'un fichier: MPI_SEEK_END

```
int MPI_File_seek(MPI_File mpi_fh, MPI_Offset
    offset, int whence);
```

MPI_2
Communication non bloquante

- Deux appels séparés par communication:
  - Initialisation de la communication
  - Complétion de la communication
- Un identificateur de requête de communication

MPI_2
Communication non bloquante

Permet :

- d'éviter les deadlocks
- De superposer les communications et les calculs

(L'algorithme doit gérer la synchronisation des processus)

- Les étapes de communication sous-jacentes sont les mêmes, ce qui diffère ce sont les interfaces avec la bibliothèque

MPI_2
Envoi non bloquant

Appel de MPI_Isend

Initialise un envoi

Retour immédiat au programme

Même argument que MPI_Send

+ un argument identifiant la requête

MPI_2
Envoi non bloquant

Appel de MPI_Irecv

Initialise une réception

Retour immédiat au programme

Pas d'argument ''status'' (pas de complétion)

Un argument identifiant la requête

MPI_2 Complétion

Test de complétion

sans bloquer le processus sur la complétion:

int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)

request [in] MPI request (handle)

flag [out] true if operation completed (logical)

status [out] status object (Status). peut-être MPI_STATUS_IGNORE.

MPI_2
Complétion

Attendre la complétion: MPI_WAIT

int MPI_Wait(MPI_Request *request,MPI_Status *status);

request

[in] request (handle)

status

[out] status object (Status). peut-être MPI_STATUS_IGNORE.

MPI_2 Avantages/Inconvénients

- Avantages:
  - Évite les deadlocks
  - Couvrir les communications par des calculs
- Inconvénients
  - Augmente la complexité des algorithmes
  - Difficile à déboguer

MPI_2
8 Fonctions Send

	bloquant	non bloquant
• Standard	MPI_SEND	MPI_ISEND
• Synchrone	MPI_SSEND	MPI_ISSEND
• Ready	MPI_RSEND	MPI_IRSEND
• Bufferisé	MPI_BSEND	MPI_IBSEND

MPI_2
Optimisation

- Algorithmique: il faut minimiser la part de communication
- Communications: lorsque la part des communications est importante par rapport au calcul
  - Recouvrir les communications par les calculs
  - Éviter la recopie des messages dans un espace mémoire temporaire

MPI_2
Communications persistantes

Pour une boucle on peut mettre en place un schéma persistant de communication

Par exemple :

Envoi standard: MPI_SEND_INIT()

MPI_2
Communications collectives

Synchronisation globale:

MPI_BARRIER()

Transfert des données:

Diffusion globale des données: MPI_Bcast()

Diffusion sélective des données: MPI_Scatter()

Collecte des données réparties: MPI_Gather()

MPI_2
Communications collectives

- Collecte par tous les processus des données réparties: MPI_ALLGather()
- Diffusion sélective, par tous les processus, des données réparties: MPI_Alltoall()
- Communication et opérations sur les données
- Réduction de type prédéfinie ou personnel: MPI_Reduce()
- Réduction avec diffusion du résultat: MPI_Allreduce()

MPI_2
Barrière de synchronisation

Impose un point synchronisation à tous les processus du communicateur

Routine MPI_Barrier

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

Bloque tous les processus jusqu'a l'appel de tous les processus

MPI_2 Echange

Routine MPI_Alltoall()

```
int MPI_Alltoall(
  void *sendbuf,
  int sendcount,
  MPI_Datatype sendtype,
  void *recvbuf,
  int recvcount,
  MPI_Datatype recvtype,
  MPI_Comm comm);
```

MPI_2 Echange

Routine MPI_ALLTOALL()

```
P0  A0  A1  A2  A3- A0  B0  C0  D0
P1  B0  B1  B2  B3- A1  B1  C1  D1
P2  C0  C1  C2  C3- A2  B2  C2  D2
P3  D0  D1  D2  D3- A3  B3  C3  D3
...
```

MPI_2 Implémentation

MPICH: MPI I/O partiel, Type dérivé partiel

LAM : Gestion dynamique, MPI I/O partiel, Type dérivé partiel

Fujitsu : Gestion dynamique, MPI I/O, Type dérivé

NEC : Gestion dynamique, MPI I/O, Type dérivé

IBM : MPI I/O, Type dérivé

• • •

OpenMP Architecture Review Board

Permanent Members of the ARB:

- AMD (David Leibs)
- Cray (James Beyer)
- Fujitsu (Matthijs van Waveren)
- HP (Uriel Schafer)
- IBM (Kelvin Li)
- Intel (Sanjiv Shah)
- NEC (Kazuhiro Kusano)
- The Portland Group, Inc. (Michael Wolfe)
- SGI (Lori Gilbert)
- Sun Microsystems (Nawal Copty)
- Microsoft (-)

OpenMP

Est une API adaptée aux architectures multi-coeur.

Elle est conçu pour une utilisation en multi-thread et avec de la mémoire partagée.

OpenMP

Tous les threads peuvent utiliser la même mémoire partagée,

Les données peuvent être privées ou partagées

Les données partagées sont utilisables par tout les threads

Les données privées sont accessibles que par le thread qui les possède

OpenMP

Les transferts de données sont transparents pour le programmeur

La synchronisation est généralement implicite

Les données possèdent des étiquettes

OpenMP

Les données sont de deux types :

Partagées :

- Il y a qu'une instance des données,
- Tous les threads peuvent lire ou écrire simultanément (par défaut)
- Toutes les modifications sont visibles pour tout les threads,

(mais pas nécessairement immédiatement)

OpenMP

Master thread

Region parallèle- worker thread

Synchronisation

Master thread

Region parallèle- worker thread

Synchronisation

....

OpenMP

Utilisation d'une boucle parallèle

```
int main(int argc, char *argv[])
 {
   const int N = 100000;
   int i, a[N];
   #pragma omp parallel for
   for (i=0;i<N;i++)
      a[i]= 2*i;
   return 0;
 }
```

OpenMP

Thread 0 i=0-9999 `a[i]= 2*i`

Thread 1 i=10000-19999 `a[i]= 2*i`

Thread 2 i=20000-29999 `a[i]= 2*i`

Thread 3 i=30000-39999 `a[i]= 2*i`

Thread 4 i=40000-49999 `a[i]= 2*i`

Thread 5 i=50000-59999 `a[i]= 2*i`

Thread 6 i=60000-69999 `a[i]= 2*i`

Thread 7 i=70000-79999 `a[i]= 2*i`

....

Strucuture OpenMP

Directives et clauses de compilation (pragma) :

elles servent à définir le partage du travail, la synchronisation et le statut privé ou partagé des données ;

elles sont considérées par le compilateur comme des lignes de commentaires à moins de spécifier une option adéquate de compilation pour qu'elles soient interprétées.

Strucuture OpenMP

Fonctions et sous-programmes : ils font partie d'une bibliothèque chargée à l'édition de liens du programme.

Variables d'environnement : une fois positionnées, leurs valeurs sont prises en compte à l'exécution.

OpenMP/MPI

MPI est un modèle multiprocessus dont le mode de communication est explicite

OpenMP est un modèle multitâche ont le mode de communication est implicite (compilateur)

Forme OpenMP

Définition des fonctions

```
#include <omp.h>
```

Directive au compilateur

```
#pragma omp parallel ...
```

Forme OpenMP

Dans une region parallèle, par défaut, le statut des variables est partagé

Au sein d'une même région parallèle toutes les tâches concurrentes exécutent le même code

Il y a une barrière implicite à la fin de région parallèle

Forme OpenMP

```
#pragma omp parallel if (n > thre)\
  shared(n,x,y) private(i,j)
```

If (expression)

indique si une exécution séquentielle ou parallèle

private (i,j)

référence à des variables locales privées

shared (n,x,y)

variable utilisable par tous les threads

Forme OpenMP

Pour certaine directive on peut supprimer la synchronisation en fin de zone parallèle.

```
#pragma omp for nowait
```

Dans ce cas il n'y a pas d'attente de synchronisation

Forme OpenMP

```
#pragma omp sections (clause)
{
#pragma omp section
block 1
#pragma omp section
block 2
}
```

block 1 // block 2

Forme OpenMP

```
#pragma omp single (clause)
{
block 1
}
```

Un thread execute le block 1

Forme OpenMP

```
#pragma omp master (clause)
{
block 1
}
```

Le thread master execute le block 1 dans ce cas il n'y a pas de barrier implicite

Forme OpenMP

```
#pragma omp critical (nom)
{
block 1
}
```

Permet de définir des zones critiques tous les threads executent le code mais un seul à la fois

Forme OpenMP

```
#pragma omp atomic
    a[indx[i]] += b[i];
```

Permet de faire des opérations sur des variables de manière atomique

(version particulière de section critique)

GOMP

```
gcc -fopenmp test.c -o test
```

```
OMP_NUM_THREADS =4
```

Exemple pour 4 thread avec gcc > 4.3

LUSTRE

http://wiki.lustre.org/

Lustre est un système de fichier de type objet,

C'est un système distribué,

Il a été conçu par SUN

(1999 ...)

Lustre 1.0 2003

Lustre 1.2 2004

Lustre 1.6 2007

Lustre 1.8 2008

LUSTRE

Lustre utilise des systèmes de fichier de type :

ext3 ou zfs

Pour le noyau système, on charge un module, le système est comme un système de fichier global

ZFS

ZFS est un système de fichier conçu par SUN (2005 ...)

(« Zettabyte File System »)

Il supporte le RAID 0, RAID 1, ...

Système de fichier 128 bits

ZFS Quelques chiffres

2^48 entrée dans un répertoire,

2^64 octets pour la taille d'un fichier

2^64 octets pour la taille d'un fichier

2^78 taille d'un zpool

....

possibilité de faire des snapshots,

mécanisme de transaction

Calculateur ULR

ymir est une SGI Altix ICE 8200 XE

Cluster de 16 noeuds reliés par un réseau infiniband.

Chaque noeud de calcul comporte deux processeurs Intel Xeon 5472 (Quad core, 3Ghz) et de 16 Giga de mémoire ; soit au total 128 cores et 256 Giga de mémoire.

L'ensemble du système fonctionne avec le système d'exploitation SUSE (SLES 10).

Calculateur ULR

Intel MPI 3.1 Librairie MPI Intel MPI3.1

Strucuture OpenMP

Directives et clauses de compilation (pragma) :

elles servent à définir le partage du travail, la synchronisation et le statut privé ou partagé des données ;

elles sont considérées par le compilateur comme des lignes de commentaires à moins de spécifier une option adéquate de compilation pour qu'elles soient interprétées.

Fin