Introduction à la programmation parallèle

Maîtrise d'informatique

Philippe MARQUET

Philippe.Marquet@lifl.fr

Laboratoire d'informatique fondamentale de Lille
Université des sciences et technologies de Lille

- Création janvier 1998, dernière révision janvier 2002
- Cette présentation reprend en partie l'introduction du cours *Outils pour le calcul scientifique à haute performance* que je donne avec Pierre BOULET aux étudiants de l'école doctorale SPI. Ce dernier est lui-même en partie bâti sur une ancienne version du présent cours. Voir `http://www.lifl.fr/west/courses/cshp/`
-
- La dernière version de ce cours est accessible à partir de `http://www.lifl.fr/~marquet/ens/pp/`
- `$Id: intro.tex,v 1.2 2002/02/11 13:07:17 marquet Exp $`

Table des matières

Références

- *Design and Building of Parallel Programs*
  Ian FOSTER
  Addison-Wesley, 1995, `http://www.mcs.anl.gov/dbpp/`
- *Algorithmes et architectures parallèles*
  Michel COSNARD et Denis TRYSTRAM
  InterÉditions, 1993
- *Initiation au parallélisme*
  Marc GENGLER, Stéphane UBEDA, et Frédéric DESPREZ
  Manuel Informatique, Masson, Paris, 1996
- *Ordinateurs et calcul parallèles*
  Observatoire français des techniques avancées, série ARAGO, Paris, 1997
- *Scalable Parallel Computing, Technology, Architecture, Programming*
  Kai HWANG et Zhiwei XU
  McGraw-Hill, 1998, `http://andy.usc.edu/book/book98.html`

Calcul haute performance et parallélisme

Motivation pour la haute performance

- « Grand Challenge »:
  - problème fondamental
  - grands impacts sur l'industrie, la science ou la société
  - solution par l'utilisation du calcul haute performance
  - utilisation/définition des prochaines générations de machines/langages pour le calcul haute performance
- Besoins importants
  - en puissance de calcul
  - en mémoire
- Ordres de grandeur : (G : giga), T : téra, P : péta
  - 1 TeraFLOPs $= 2^{40} \approx$ 1.000.000.000.000 (un billion) opérations flottantes par seconde
  - 1 PetaByte : vidéo de 2300 ans, 1 milliard de livres...

Exemples de « Grand Challenges »

- Simulations de dynamique des fluides
  - conception d'avions hypersoniques, aérodynamisme automobile, sous-marins furtifs
  - prévision météorologiques à cours et long terme, climatiques
  - prospection pétrolière...
- Calculs de structures électroniques et conception de nouveaux matériaux
  - catalyse chimique
  - agents immunitaires
  - superconducteurs
- Calculs pour comprendre la nature fondamentale de la matière
  - chromodynamique quantique
  - théorie de la matière condensée...
- Calculs symboliques dont :
  - reconnaissance de la parole
  - vision par ordinateur
  - compréhension du langage naturel
  - raisonnement automatique
  - outils pour concevoir, construire et simuler des systèmes complexes

Calcul haute performance ?

- Calcul haute performance =
  - algorithme efficace
  - calcul de solutions approchées
  - etc.

  \+ Parallélisme
- Parallélisme ?=
  - programmation séquentielle
    - un seul flot d'exécution
    - une instruction exécutée à la fois
    - un processeur
  - programmation parallèle
    - plusieurs flots d'exécution
    - plusieurs instructions exécutées simultanément
    - plusieurs processeurs

Parallélisme, pourquoi ?

- Puissance de calcul
  - vitesse des processeurs augmente
  - limitations, principalement technologiques
  - applications encore plus demandeuses en puissance de calcul
  - solution : exécuter plusieurs opérations en même temps
    - au sein d'un processeur
    - par duplication des éléments de calcul
- Coût
  - excellent rapport coût/performance nécessaire
  - accroissement de la puissance d'un élément de calcul
    $\Rightarrow$ explosion des coûts : coût marginal prohibitif
  - grande puissance de calcul : faire coopérer de nombreux éléments de calculs de « faible » puissance et de moindre coût

Modèle pour le parallélisme

- Programmation séquentielle
  - modèle unique (von Neumann)
- Programmation parallèle
  - pas de modèle unique
  - différents aspects à considérer
    - architecture
    - logiciel
    - algorithmes
- Le parallélisme est orthogonal à l'informatique

Différents aspects du parallélisme

- **Architecture**
  comment organiser et implanter de multiples processeurs
  - Unité de contrôle unique / multiples
    - moins de flexibilité / trop de liberté
  - Mémoire partagée / distribuée
    - bande passante d'une mémoire partagée
    - réseau d'accès à la mémoire
  - Systèmes distribués
    - topologie du réseau d'interconnexion
    - surcoût des communications
- **Logiciel**
  - Approche de la programmation parallèle
  - Correction d'un programme parallèle
  - Implémentation
  - Outils spécifiques
- **Algorithmes**

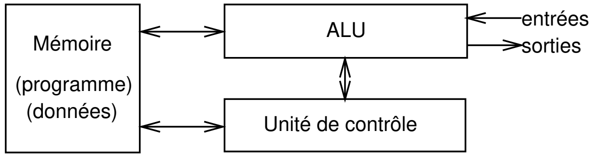
Implications du parallélisme

- Nouveaux concepts
  - concurrence
  - communication
  - synchronisation
- Facteurs de performance
  - allocation de processeurs/ordonnancement
  - distribution des données
  - granularité
  - ratio communications/calcul
  - équilibre (dynamique) de la charge
  - extensibilité
- Problèmes spécifiques
  - non-déterminisme
  - correction/justesse
  - interblocage
  - terminaison
- Dimension algorithmique
  - pensée/raisonnement parallèle
  - meilleur algorithme séquentiel *vs* meilleur algorithme parallèle

Taxinomie des architectures de machines

Modèle von Neumann

- John VON NEUMANN (1946)
  - grandes lignes pour construire une machine électronique
  - appliquées jusqu'à nos jours
- Modèle procédural
  - une séquence d'opérations peut décrire tous les problèmes
- Différents blocs fonctionnels :
  - mémoire : données et instructions
  - unité de contrôle : charge les instructions depuis la mémoire
  - ALU : exécute les instructions
  - I/O : transferts de données



- Séquencement des instructions inhibe le parallélisme

Développements architecturaux

- Règle générale

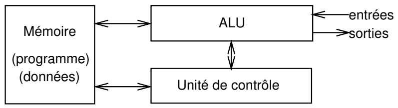
  extraction du parallélisme
- Plusieurs aspects
  - fonctions du processeur
  - hiérarchie du système mémoire
  - interface processeur/mémoire
  - systèmes multiprocesseurs

Classification de Flynn

- Machine parallèle :
  - plusieurs processeurs
  - classification selon l'agencement des processeurs
  - nombreuses méthodes de classification
- Michael J. FLYNN (1972)
  - flux d'instructions : séquence d'instructions exécutées par la machine
  - flux de données : séquence des données appelées par le flux d'instructions
  - insuffisant pour classer finement les architectures parallèles

$$\left.\begin{array}{llll} \textbf{S} & \text{Single} & \textbf{I} & \text{Instruction} \\ \textbf{M} & \text{Multiple} & \textbf{D} & \text{Data} \end{array}\right\} \Rightarrow \left\{\begin{array}{ll} \textbf{SISD} & \textbf{MISD} \\ \textbf{SIMD} & \textbf{MIMD} \end{array}\right.$$

- **SISD** La plupart des ordinateurs actuels

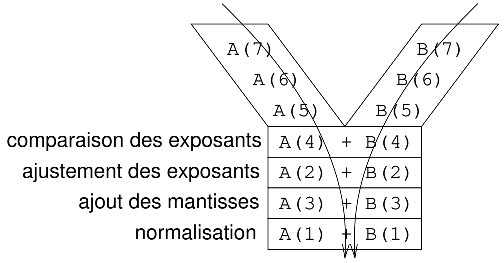


Classification de Flynn — SIMD

- **SIMD** *Array processors*
- Chaque instruction s'applique à des données différentes
- Déroulement synchrone
- Parallélisme de données, *data parallelism*
- Deux sous-classes :
  - machines SIMD vectorielles — processeurs de tableaux
  - machines SIMD parallèles — tableaux de processeurs

Classification de Flynn — SIMD vectoriel

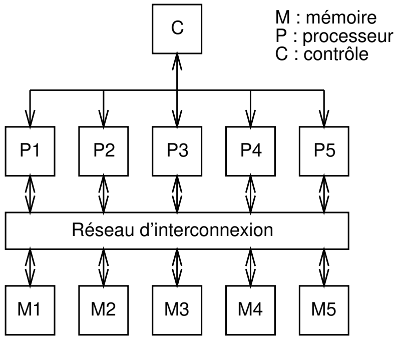
- Instructions sur des vecteurs de données
- Exécution pipeline
- Cray-1 (1976), Fujitsu VP, NEC SX-5, Tera/Cray SV1
- Multiprocesseurs vectoriels



- Supercalculateurs « traditionnels » : grandeur et déclin
  - dans les centres de calcul depuis 30 ans
  - programmation vectorielle « maîtrisée »
  - coûteux
    - technologie complexe
    - système de refroidissement
  - suprématie fréquence horloge terminée

Classification de Flynn — SIMD parallèle

- Tableau de processeurs
- Exécution synchrone de la même instruction
- Connection Machine CM-1 (1985), MasPar MP-1 et MP-2
  - MasPar 16.384 PE à Lille (1992–1998)



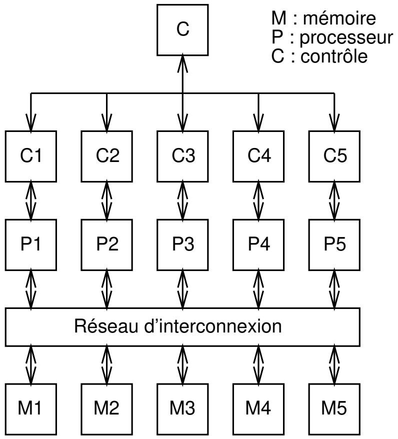
- En voie de disparition
  - processeurs élémentaires spécifiques
  - diffusion synchrone des instructions à la fréquence actuelle des processeurs
  - architecture simple : processeurs enfouis/embarqués

Classification de Flynn — MISD

- **MISD** n PE recevant des instructions différentes sur un même flux de données
- Pas de machine existante !
- Parfois confondu avec SIMD vectoriel

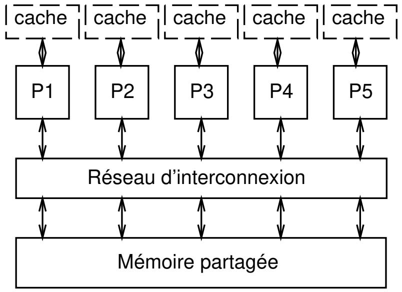
Classification de Flynn — MIMD

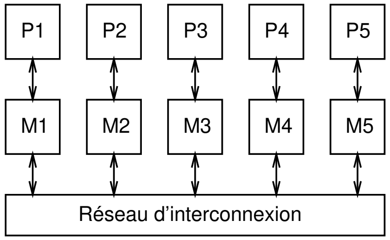
- **MIMD**
  n PE et n flux d'instructions
  - flux d'instructions indépendants
  - flux de données indépendants
- Multiples configurations
  - mémoire (données) propre
  - mémoire partagée
- Tendance actuelle
  - processeurs banalisés
  - architecture de la machine = réseau
- Difficulté de programmation



- **SPMD** Variante de MIMD. Même programme sur des données différentes
  SIMD $<$ SPMD $<$ MIMD

Mémoire partagée / mémoire distribuée

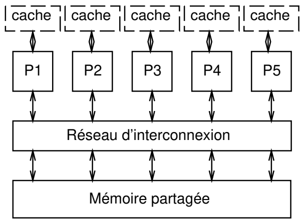




- Machine MIMD à mémoire partagée
- PC bi-processeur, Cray SV1, SGI Origin 3000
  Un nœud de l'IBM SP-3

- Machine MIMD à mémoire distribuée
- Stations de travail + Ethernet, Tera/Cray T3E
  IBM SP-3

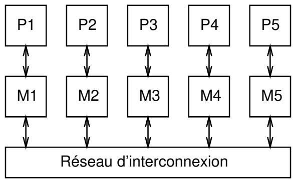
MIMD à mémoire partagée



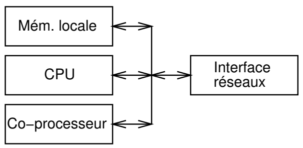
- Multiprocesseurs classiques (multiprocesseurs vectoriels)

- Deux niveaux mémoire
  - cache
  - mémoire partagée
  - localité des données
  - (faux partages...)
- Synchronisation
  - contrôle des accès à la mémoire partagée
  - réduire ces synchronisations
- Extensibilité limitée
  - réseau d'interconnexion
  - clusters de multiprocesseurs

MIMD à mémoire distribuée



- MIMD à passage de messages
  - un ensemble de « nœuds »
  - un réseau
- Nœud
  - unité de calcul
  - mémoire locale
  - gestion de la communication entre les nœuds



- Mémoire et messages
  - mémoire locale accessible par les processus locaux
  - communication par le réseau
  - message : (données, référence de processus)
  - attention portée sur la diminution des communications
  - passe par une bonne décomposition des données

MIMD extensible à mémoire partagée

- MIMD à mémoire partagée : grand nombre de processeurs ?
- **SMP** *Symetric Multiprocessor*
  - processeurs identiques
  - accès indifférencié à la mémoire
  - aussi nommé CC-UMA : *Cache Coherent Uniform Memory Access*
  - cohérence des caches : lors d'une mise à jour mémoire, tous les processeurs sont avertis

  extensibilité ?
- **NUMA** *Non-Uniform Memory Access*
  - exemple : SGI Origin 3000
  - aussi nommé CC-NUMA : *Cache Coherent NUMA*
  - souvent construit comme reliant plusieurs SMP
  - chacun des SMP peut accéder la mémoire des autres SMP
  - temps d'accès à la mémoire n'est pas égal pour tous

  extensible

MIMD CLUMPS

- **CLUMPS** *Clusers of SMP*
  - construits à partir de SMP
  - mémoire partagée au sein de chacun des SMP
    - accès (via le cache du processeur) à la mémoire du SMP
  - mémoire distribuée entre les SMP
    - communication par messages entre les SMP
- Exemple : l'IBM SP-3 de l'USTL
  - nœud SMP 16 processeurs NH2
  - 4 nœuds reliés par un réseau dédié Colony

Systèmes parallèles et systèmes distribués

- Systèmes parallèles
  - ensemble d'éléments de calcul (PE)
  - qui peuvent communiquer et coopérer
  - pour résoudre rapidement de grands problèmes
- Systèmes distribués
  - ensemble de processeurs autonomes
  - qui ne se partagent pas de mémoire primaire
  - mais qui coopèrent par envoi de messages au travers un réseau de communication

Systèmes parallèles / systèmes distribués

- Pas parallèles, pas distribués
  - PCs ou stations de travail (qui ne peuvent pas communiquer)
- Parallèles, pas distribués
  - Machines vectorielles
  - Machines multiprocesseurs à mémoire partagée (SMP, *Symmetric MultiProcessor*)
- Distribués, pas parallèles
  - Réseau de stations large distance (communications trop lentes)
  - Base de données distribuées
- Parallèles et distribués
  - Réseau de stations de travail connectées à un réseau local/spécialisé
    NOW/COW : *Network/Cluster of Workstations*
    CLUMPS : *Cluster of SMP Workstations*

Modèles de programmation parallèle

Modèle de fonctionnement et modèle de programmation

- Modèle de fonctionnement des machines parallèles/distribuées
  - lié à l'architecture de la machine
  - comment sont exécutées les instructions élémentaires
- Modèle de programmation parallèle
  - lié à l'expression de l'algorithme
  - quelles sont les « entités » parallèles du programme
  - comment sont exprimées leurs interactions
- Passer du modèle de programmation au modèle d'exécution :
  - compilateur, support d'exécution
  - de immédiat à ... travail ardu de recherche en informatique
- Programmeur ne se préoccupe pas du modèle de fonctionnement

Multiplicité des modèles de programmation

- Deux grandes approches du parallélisme :
  - **Parallélisme de tâches** (*task or control parallelism*)
    - décomposition d'une tâche en sous-tâches
    - variables partagées entre les tâches, ou communications par messages entre les tâches
    - mise en œuvre :
      tâche $\leftrightarrow$ processeur
  - **Parallélisme de données** (*data parallelism*)
    - structures homogènes « tableaux »
    - affectation de « tableaux »
    - mise en œuvre :
      donnée « élémentaire » $\leftrightarrow$ processeur
- Combinaison des modèles !

Modèles de programmation à parallélisme de tâches

- Un ensemble de tâches : processus ou thread
  - chaque tâche travaille mémoire locale/privée
    + synchronisation entre les tâches
  - communication par messages
    - opération coopérative : un émetteur un récepteur

    ou mémoire partagée
    - mécanisme de protection des accès à cette mémoire
- Bibliothèque de fonctions
  - parfois directives/extension de langage
- Processus communicants
  - PVM *parallel virtual machine*
  - MPI *message passing interface*
- Processus (légers, threads)
  - interface POSIX `pthreads`
  - standard OpenMP

Modèles de programmation à parallélisme de données

- Manipulation de structures homogènes
  - ≡ tableaux
  - haut niveau d'expression du parallélisme
  - découpage des structures
  - distribution des structures sur les processeurs
  - travail sur la structure locale
  - restructuration des structures : communications
- Langages à parallélisme de données
  - Fortran 95 (norme Fortran)
    - manipulation de tableaux
  - HPF *High Performance Fortran*
    - extension de Fortran 95 pour machines à mémoire distribuée
    - distribution des tableaux sur des processeurs
  - DPCE *Data-Parallel C Extension*

Modèles de fonctionnement vs modèles de programmation

- Modèle de fonctionnement est lié à l'architecture de la machine
  Trois grandes classes :
  - SISD — Simple Instruction, Simple Donnée
  - MIMD — Multiples Instructions, Multiples Données
  - SIMD — Simple Instruction, Multiples Données
- Modèle de programmation est lié à la traduction de l'algorithme
  Deux principaux modèles :
  - parallélisme de tâches (↪ SISD, MIMD)
  - parallélisme de données (↪ SISD, MIMD, SIMD)

Performance

Performance ?

- Comment caractériser les performances d'un système ?
  - d'un système pour une (classe d')application donnée
  - benchmark
- Rapport entre performance et coût
- Quelle métrique pour la performance ?
- Théorèmes et lois

Benchmark

- Programmme de test des performances
  - mesure + prédiction des performances
  - découverte des points faibles et forts
- Macro benchmark
  - applications ou applications synthétiques
  - mesure les performances globales
- Micro benchmark
  - noyau de procédures
  - mesure des aspects spécifiques (CPU, mémoire, I/O...)
- Nombreux benchmarks disponibles
  - macro : PARKBENCH, SPEC ($\rightarrow$ SPECint...), Splash
  - micro : STREAM, LINPACK
- LINPACK
  - noyau d'algèbre linéaire
  - classement TOP-500, `http://www.top500.org`

Performance ? À quel coût ?

- Temps entre le début et la fin de l'application
  - utilisateur : minimiser ce temps
  - minimise le coût d'une exécution
    - heures CPU payantes
- Rapporter le coût d'exécution au coût de développement
  - utilisation de paralléliseur automatique
  - code peu stable ($\neq$ code de production)
- Débit d'une machine
  - nombre de jobs exécutés en une unité de temps
  - multi-processing
    - partitionner une machine multiprocesseurs
    - partition statique/dynamique
  - réduire le coût : améliorer le débit

Métriques

- Temps d'exécution
  - en secondes : *wallclock time* ou *elapsed time*
  - en temps CPU
- Nombre d'instructions
  - nombre d'instructions exécutées par la machine
  - pas seulement dépendant du programme
    - compilateur, optimisations
    - architecture du processeur : RISC, CISC
  - vitesse en MIPS, millions d'instructions/seconde
- Nombre d'opérations (flottantes)
  - nombre d'opérations nécessaires à la production du résultat
  - `add`, `sub`, `mul`= une opération
    `div`, `sqrt` = 9 opérations
    `sin`, `exp`= 17 opérations
  - Flop, Mflop, Gflop
  - vitesse en Mflop/s, Gflop/s (Mflops, Gflops !)

Facteur d'accélération

- Un algorithme
  - exécution sur p processeurs en un temps t_p
  - exécution sur 1 processeur en un temps t_1
- Accélération (*speedup*)

$$S_p = \frac{t_1}{t_p}$$

- t_1 temps du meilleur algorithme séquentiel
  et non temps de l'algorithme parallèle exécuté sur 1 processeur
- Étude de l'accélération

$$1 \leq S_p \leq p$$

Code séquentiel : $S_p = 1$
Code « purement » parallèle : $S_p = p$

Loi d'Amdahl

- G. M. Amdahl (1967)
- Trouver une borne à l'accélération
  - pour un problème donné
  - pour une taille de problème donnée
- Identifier la partie parallélisable du code
  - charge de travail fixe W (par exemple en Mflop)
  - distribution de cette charge entre P processeurs
  - décomposition de la charge

$$\begin{aligned} W = {} & \alpha W && \text{partie séquentielle} \\ & + (1-\alpha)W && \text{partie parallélisable} \end{aligned}$$

- Accélération

$$S_p = \frac{t_1}{t_p} = \frac{W}{\alpha W + (1-\alpha)(W/p)} = \frac{p}{1+(p-1)\alpha} \rightarrow \frac{1}{\alpha} \quad \text{quand} \quad p \rightarrow \infty$$

Illustration de la loi d'Amdahl

- Accélération pour p processeurs

$$S_p = \frac{p}{1+(p-1)\alpha}$$

α pourcentage de code séquentiel

- Illustration

$p \backslash \alpha$	50%	10%	1%
10	1.82	5.26	9.17
100	1.98	9.17	50.25
1000	1.99	9.91	90.99
10000	1.99	9.91	99.02

Implications de la loi d'Amdahl

$$\boxed{S_p \rightarrow \frac{1}{\alpha} \quad \text{quand} \quad p \rightarrow \infty}$$

- Accélération limitée par la partie séquentielle
  - limite indépendante du nombre de processeurs

$p \backslash \alpha$	50%	10%	1%
∞	2	10	100

- Bonne accélération : diminuer la partie séquentielle
  - optimiser cette partie séquentielle
  - ne pas focaliser sur la partie parallèle

Loi de Gustafson

- Loi Amdahl = accélération à travail constant
- John Gustafson (1988)
  - parallélisme : problème de plus grande taille
  - meilleure précision du résultat...
- Charge de travail augmente avec la taille de la machine p

$$\begin{aligned} W' = {} & \alpha W && \text{partie séquentielle} \\ & + (1-\alpha)pW && \text{partie parallèle} \end{aligned}$$

  - W charge de travail du problème originel
  - sur un processeur : temps d'exécution de W' = proportionnel à W'
  - sur p processeurs : par définition, temps d'exécution proportionnel à W
- Accélération à temps constant

$$S'_p = \frac{t_1}{t_p} = \frac{\alpha W + (1-\alpha)pW}{W} = \alpha + (1-\alpha)p$$

Implication de la loi de Gustafson

$$\boxed{S'_p = \alpha + (1-\alpha)p}$$

- Accélération à temps constant linéaire en p
- Problème extensible
  - partie séquentielle n'est plus un problème
  - partie séquentielle αW doit rester constante
  - problèmes extensibles juqu'à un certain degré

Machine parallèle de l'USTL

Architecture générale

- IBM RS6000/SP 3
  - 4 nœuds de 16 processeurs Power3-NH2
  - 64 Go de mémoire vive
  - nouveau réseau d'interconnexion Colony
  - 582,4 Go de disque dont 436,8 Go de stockage
- TOP500
  - 70,6 Gflops sur le benchmark Linpack
  - rang
    - novembre 2000 : 354^e (15^e français)
    - juin 2001 : 485^e (19^e français)
    - septembre 2001 : plus dans la liste

Architecture d'un nœud SMP NightHawk-2

- 1 réseau interne crossbar avec 4 ports processeur, 4 ports mémoire et 1 port E/S
- 16 processeurs Power3-II
- 16 Go de mémoire partagée (protocole de cohérence de caches MESI)
- 2 disques de 18,2 Go
- Interconnexion : réseau Colony
  - architecture distribuée, indirecte de topologie de type oméga
  - latence avec MPI : 19 μs
  - débit d'un lien en MPI : 800 Mo/s

Processeur Power3-II

- Technologie :
  - 64 bits, 375 MHz, 23 millions de transistors
  - connectique cuivre, lithographie à 0,22 μm
- Multiples unités d'exécution :
  - 2 unités de calcul flottant
  - 3 unités de calcul entier
  - 2 unités de chargement/déchargement
- Hiérarchie mémoire :
  - cache L1 : 32 Ko instructions / 64 Ko données, latence 1 cycle
  - cache L2 : 8 Mo, latence de l'ordre de 7 cycles, 32 octets/cycle, débit de 12 Go/s
  - supporte plusieurs défauts de caches simultanés et préchargement
  - débit mémoire théorique : 3 Go/s

Processeur Power3-II : performances

- Exécution de $\leq$ 8 instructions dans le désordre par cycle
- Terminaison de $\leq$ 4 instructions par cycle
- Nombres flottants :
  - débit du pipeline : 1 cycle/opération
  - temps d'une opération (add, mult ou mult+add) : 3 cycles
- Benchmarks :
  - SPECint95 : 23,5 (Pentium III - 866 MHz : 41,7)
  - SPECfp95 : 51,3 (Pentium III - 866 MHz : 33,6)
- Points forts :
  - capable de maintenir une puissance proche de la puissance de crête sur les calculs flottants
  - débit mémoire particulièrement important

Environnement logiciel

- Système d'exploitation : AIX 4.3.3
- Compilateurs :
  - *C for AIX V5.0* (parallélisation auto, directives OpenMP)
  - *Fortran* V7.1 (fortran 77, 90, 95, parallélisation auto, directives OpenMP)
  - Visual Age C++ V5.0 (pas de support SMP)
  - *HPF* V1.4 (standard HPF 1.1)
- Bibliothèques :
  - de calcul : ESSL, *P-ESSL*, ScaLAPACK, MASS
  - de communication : *MPI*, PVM, LAPI
  - de threads : `pthreads`, *OpenMP*
- Outils :
  - debogage : dbx, pdbx (MPI)
  - analyse de performance : gprof, xprofiler et VT

Références

- La page du CRI :
  `http://ustl.univ-lille1.fr/calcul-intensif/`
- La page IBM RS6000/SP :
  `http://www.ibm.com/servers/eserver/`
  `pseries/hardware/largescale/`
- Les documentations IBM SP :
  `http://www.rs6000.ibm.com/resource/`
  `aix_resource/sp_books/`
- Le groupe d'utilisateurs de calcul scientifique sur SP :
  `http://www.spscicomp.org/`

Programmation parallèle par passage de messages

Processus communicants

- Machine parallèle :
  - un ensemble de processeurs « indépendants »
  - communicants via un réseau
- Application parallèle :
  - un ensemble de processus « indépendants »
  - s'échangeant des messages
- Implantation « naïve » :
  - un processus par processeur
  - transit des messages via le réseau

Primitives de programmation par passage de messages

- Processus :
  - programme standard
  - échange de messages
- Environnement de programmation par passage de messages basique :
  - un compilateur (C par exemple)
  - quelques fonctions de bibliothèque :

    `nproc()` : nombre de processus alloués à l'application
    `myproc()` : mon identification au sein de ces processus
    `send(dest, mess)` : envoi d'un message
    `recv(mess)` : réception d'un message
- Environnement de programmation par passage de messages actuel :
  - bibliothèque complète (plus de 100 fonctions !)
  - outils de debogage
  - analyseur de performances
  - etc.

Programmation par passage de messages

- Appréhender le fonctionnement d'une application
  - déroulement simultané des différents processus
  - interactions / dépendances entre les processus
- On se limite à des schémas simples de fonctionnement
  - tous les processus exécutent le même code
  - un processus particulier : le maître
- Schémas de communication avancés
  - diffusion : un vers tous
  - réduction : tous vers un
  - etc.
- Efficacité de ces schémas de communication
  - implantation optimisée
  - utilisation effective dans une application

Déploiement de l'application

- Complexité de la phase initiale de déploiement de l'application
  - identification des processeurs
  - nombre de processus
  - répartition des processus sur les processeurs
  - faire que les processus se connaissent entre-eux
- Évolution du déploiement au cours de l'exécution de l'application

Bibliothèques de communications existantes

- PVM : *Parallel Virtual Machine*
  - `http://www.epm.ornl.gov/pvm/`
  - 1989
  - produit universitaire (ORNL, University of Tennessee Knoxville)
  - portable (réseaux de stations (NOW/COW, Unix), machines parallèles)
- MPI : *Message Passing Interface*
  - `http://www.erc.msstate.edu/mpi/`
  - conception en 1993-94 $\rightarrow$ standard
  - efficacité
  - différentes implémentations : MPICH, CHIMP, LAM, constructeurs...
- Évolution
  - PVM $\rightarrow$ MPI
  - MPI $\rightarrow$ PVM
  - MPI prend le pas devant PVM

Parallélisme de données

Structures de données

- Quelles sont les structures de données utilisées en calcul scientifique ?
  - essentiellement les *tableaux*
  - collections
- Programmation de haut niveau :
  - programmation de type *SPMD* (Single Program Multiple Data)
  - le programmeur place ses données sur les processeurs
  - le compilateur gère les communications

Le calcul est dirigé par les données

- Règle de base : le processeur qui effectue le calcul est celui qui possède la donnée (*owner computes rule*)
- Répartition des calculs dirigée par répartition des données
  - importance primordiale du placement des données
  - placements proposés par le langage ?
- Communications implicites
  - langage de haut niveau
  - compilateur génère et optimise les communications
  - estimation du volume de communication pas toujours facile

Placement des données

- Alignement
  - placement relatif de plusieurs structures de données
  - deux tableaux qu'on manipule ensemble doivent être répartis de la même façon
- Distribution
  - visibilité de l'architecture de la machine au niveau du langage ?
  - indépendance du placement par rapport au nombre de processeurs et à la topologie de la machine
  - en général placement sur une grille virtuelle

Distributions typiques de tableaux

- Distribution par dimension
  - quelle(s) dimensions répartir/répliquer ?
- Modes de répartition
  - par blocs
    - simplicité des accès aux tableaux et des communications
    - taille du bloc = taille du tableau / nombre de processeurs
  - cyclique par blocs
    - complexité des accès aux tableaux et des communications
    - meilleur équilibrage de charge pour de nombreux algorithmes
    - taille du bloc choisie par le programmeur
- Une répartition pour tout le programme ou redistribution en cours d'exécution ?

Expression du parallélisme

- Accès parallèle aux structures
  - sans passer par les éléments
  - expressions de tableaux
- Structures de contrôle parallèles
  - boucles principalement
- Fonctions parallélisées
  - intrinsèques au langage / bibliothèques
  - indépendance par rapport au placement ?

Langages à parallélisme de données

- Fortran 95 : norme Fortran
  - langage haut niveau « moderne »
  - calcul scientifique
  - machines parallèles et stations de travail
- HPF *High Performance Fortran*
  - distribution des tableaux
  - itérateur parallèle `forall`

Processus à mémoire partagée

Mémoire partagée

- Plusieurs tâches concurrentes
  - gestion de cette concurrence (processus, threads, ?)
- Communication par lecture/écriture dans une mémoire commune
  - mémoire physiquement partagée (un nœud de l'IBM SP3)
  - mémoire virtuellement partagée (illusion entretenue par le système)
- Accès concurrents aux données
  - gestion des conflits
  - accès en section critique
  - possibilité d'interblocages

Processus légers

- Unité d'ordonnancement
  - choix entre coopératif et préemptif
  - dirigé par l'implémentation et la durée de vie typique
- Threads utilisateur
  - gérés par une bibliothèque
  - changement de contexte très rapide
  - choix éventuel de l'ordonnancement
  - problème de gestion des entrées/sorties
- Threads noyau
  - gérés par le système d'exploitation
  - changement de contexte moins rapide
  - bon recouvrement des entrées/sorties
- On n'a pas forcément le choix !

Sections critiques

- Verrous
  - accès concurrent aux mêmes données
  - gestion de sections critiques
  - synchronisation ?
- Faux partages
  - exemple 1 : verrouillage d'un tableau et accès à des éléments distincts
  - exemple 2 : 2 données indépendantes dans la même ligne de cache / page
- Utilisation de variables locales si possible

Répartition du travail entre les tâches

- Manuelle
  - parallélisme de tâches
  - MIMD
  - synchronisation explicite
- Automatique (boucles)
  - partition des itérations entre les processeurs
  - en blocs de taille fixe ou variable, de manière cyclique
  - statique ou dynamique

Quid de la répartition des données ?

- Si temps d'accès à la mémoire uniformes
  - inutile
  - gestion purement matérielle
- Si accès non uniformes
  - devient utile, voire indispensable
  - quelques extensions proposent des placements à la HPF
  - sinon, mécanismes de cohérence de caches et/ou de migration de pages

Outils de gestion de processus légers

- API bibliothèque de gestion de threads : `pthread`
  - standard Posix
  - complet et/mais bas niveau
- norme OpenMP
  - standard industriel
  - directives + bibliothèque
  - en évolution

Paradigmes de programmation parallèle

Paradigme de programmation parallèle

- Structure d'algorithmes
- Pour exécution sur une machine parallèle
- Quelques paradigmes reconnus
- Un programme = combinaison de plusieurs des paradigmes

« Parallélisme de phases »

- Programme = suite d'étapes
- Étape = 2 phases
  - calcul
  - interaction
- Calcul
  - de multiples processus
  - réalisent chacun
  - un calcul indépendant
- Interaction entre les processus
  - synchronisation
  - communication bloquante, etc.
- Inconvénients :
  - pas de recouvrement en l'interaction et le calcul
  - nécessité et difficulté de maintenir l'équilibre de charge entre les processus

Itérations parallèles

- Cas particulier du parallélisme de phases
- Itérations parallèles synchrones

```
parfor p=0 to P-1
    for i=0 to N
       x[p] = f (p, i, x)
       rendez-vous ()
```

  Les étapes sont des itérations d'une boucle
- Itérations parallèles asynchrones

```
parfor p=0 to P-1
    for i=0 to N
       x[p] = f (p, i, x)
```

  Itérations indépendantes
  Utilisation de `x[p]` produit au temps `i-1` par un processus autre que `p` ?

Diviser pour régner

- Découverte dynamique d'un arbre de calculs
- Un processus parent divise le travail entre ses fils
- Les processus fils combinent leur résultat vers leur père
- Division et regroupement récursifs
- Gestion des fils :
  - création dynamique et terminaison, ou
  - utilisation d'un pool de processus

Pipeline

- De multiple processus forment un pipeline virtuel
- Un flot de données parcourt le pipeline
- Recouvrement entre les calculs des différents processus
- Recouvrement possible des communications entre les processus par les calculs
- Équilibre du pipeline

Maître/esclaves

- Ferme de processus ou maître/esclave
- Un processus maître = coordinateur
  - exécute le code séquentiel
  - initie des processus esclaves
  - transmet du travail à ces processus esclaves
  - attend les résultats des esclaves
  - itération de l'assignation de travail
- Des processus esclaves
  - attend du travail du maître
  - exécute le travail
  - retourne le résultat au maître
- Paradigme simple
- Maître = goulot d'étranglement
  Paradigme non-extensible

Pool de travail

- Pool de travail = structure de données globale/partagée
- Multiples processus
- Processus « libre »
  - prend un travail dans le pool
  - produit zéro, un, ou plusieurs nouveaux travaux dans le pool
- Fin du programme quand le pool est vide
- Implantation en mémoire partagée plus naturelle et plus facile
- Équilibre de la charge de travail
- Problème du dimensionnement des travaux
  - trop petit : surcoût d'accès au pool
  - trop grand : non équilibrage de la charge
  - variable : diminution en fonction de la somme des travaux restants

Conception d'applications

À la découverte du parallélisme

- Où est le parallélisme dans l'application ?
  - étude des *dépendances* entre les données
- Découpage en tâches élémentaires
  - parfois évident
  - peut nécessiter des transformations du programme
  - peut aller jusqu'à des transformations de l'algorithme
- Partitionnement des données
  - recherche des données « proches »
  - but : minimiser les communications

Choix du modèle de programmation

- Une affaire de compromis
  - efficacité
  - facilité de programmation
  - contrôle fin du comportement du programme
  - maintenance du code
  - expertise de l'utilisateur
  - compilateurs de la machine cible
- Mixage de plusieurs modèles
  - prototypage en parallélisme de données puis optimisation avec passage de messages
  - parallélisme de données avec utilisation de bibliothèques de passage de messages
  - ensemble de tâches data-parallèles

Choix de la granularité

- Maximiser le degré de parallélisme et minimiser le volume de communication sont contradictoires
- Il ne sert pas forcément à grand chose d'avoir un parallélisme trop fin
  - une bonne approche est un découpage fin suivi d'un regroupement
- Influence de la hiérarchie mémoire
  - parallélisme au niveau des instructions
  - effets de cache
  - alignement des tableaux en mémoire
  - calcul « out of core » pour les très gros volumes de données

Équilibrage de charge

- Statique
  - en général : trouver la distribution de données la plus adaptée
  - optimiser les communications
- Dynamique
  - dans le cas où le calcul dépend des données
  - dans le cas d'une machine hétérogène ou non-dédiée