Bibliothèques de communications

Maîtrise d'informatique

Philippe MARQUET

Philippe.Marquet@lifl.fr

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

- Création février 1998, légère révison avril 2002
-
- La dernière version de ce cours est accessible à partir de `http://www.lifl.fr/~marquet/ens/pp/`
- `$Id: mess.tex,v 1.6 2002/04/26 15:40:48 marquet Exp $`

Table des matières

Références

- *Les bibliothèques de communications*
  Frédéric DESPREZ et Pierre FRAIGNAUD
  in « Ordinateurs et calcul parallèles », Observatoire français des techniques avancées, série ARAGO, Paris, avril 1997
- *Initiation au parallélisme*
  Marc GENGLER, Stéphane UBEDA, et Frédéric DESPREZ
  Manuel Informatique, Masson, Paris, 1996

Paradigmes de programmation parallèle

Paradigmes de programmation parallèle

- Deux paradigmes principaux de programmation parallèle
  - *Parallélisme de contrôle* :
    Rechercher des instructions, voire des processus pouvant s'exécuter en parallèle
  - *Parallélisme de données* :
    Rechercher une répartition des données pour maximiser les opérations (souvent identiques) pouvant être exécutées en parallèle sur ces données

Paradigmes de programmation parallèle *(cont'd)*

- Trois manières d'aborder la parallélisation d'une application en vue d'une exécution sur une machine MIMD
  - *mémoire virtuellement partagée*
    - programmation « assez » aisée
    - performances « assez » faibles
  - *parallélisation automatique*
    - situation « idéale » !
    - compilateur-paralléliseur : travail ardu de recherche
  - *passage de messages*
    compromis entre
    - difficulté de programmation
    - qualité des performances

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.

Fonctionnalités d'une bibliothèque de communications

Fonctionnalités d'une bibliothèque de communications

- Choix lors de la conception d'une bibliothèque de communications
- Évolution des fonctionnalités :
  - maturation des bibliothèques
  - retour des utilisateurs
- Quelques fonctionnalités :
  - Portabilité, Gestion des processus, Gestion des groupes de processus, Contextes de communication, Communications point-à-point, Communications collectives, Topologies de communication, Types de données « évolués », Liaisons avec les langages, Génération de trace, Comportement face aux processus légers, Traitement de l'hétérogénéité, Tolérance aux pannes, etc.

Portabilité

- Deux tendances :
  - bibliothèque destinée à une machine spécifique et optimisée très finement pour celle-ci
  - bibliothèque portable sur un grand nombre de plates-formes
- La bibliothèque idéale :
  - portable sur toutes les plates-formes
  - optimisée pour chacune d'entre elles

Gestion des processus communicants

- Possibilité de créer dynamiquement des processus
- Création dynamique de processus ⇒ influence sur les performances des communications
- Fonctions indispensables à la gestion des processus :
  - permettre à chaque processus d'identifier son numéro
  - connaître le nombre total de processus de l'application

Gestion de groupes de processus

- Réalisation de communications collectives
- Appartenance à un groupe : statique ou dynamique
- Dynamicité ⇒ coût de maintien d'une vision globale du système

Contextes de communication

- Marquer les messages pour les différencier d'autres messages : Étiquettes (ou « tag »)
- Nécessaire pour l'écriture de bibliothèques parallèles
  - algèbre linéaire, imagerie...
  - protéger les messages « internes » échangés par la bibliothèques des messages de l'application
- Une étiquette gérée par l'utilisateur et insuffisant ; il faut
  - soit que la bibliothèque connaisse les étiquettes utilisées par l'application
  - soit que l'application connaisse les étiquettes utilisées par la bibliothèque
- Il faut
  - des étiquettes attribuées par l'environnement
  - une garantie d'unicité de ces étiquettes
  - possibilité de propager ces étiquettes aux processus du même contexte

Communications point-à-point

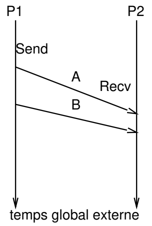
- Communications de base
- Émission d’un message par un processus
  Réception du message par un autre processus
- Paramètres habituels pour l’émission :
  - taille du message (nombre d’éléments d’un type donné ou nombre d’octets)
  - adresse d’un tampon contenant le message
  - étiquette du message
  - adresse du processus destinataire
- Pour la réception :
  - adresse d’un tampon où écrire le message
  - (étiquette du message)
  - (origine du message)

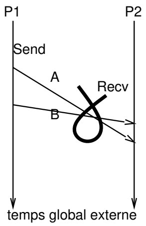
Liens fiables, FIFO, et temps global

- Un lien de communication est un lien *logique* reliant deux sites
- Sur les machines parallèles, on considère que les liens
  - sont fiables
    - le lien de communication ne perd pas de messages, ne duplique pas les messages, n’altère pas les messages
  - garantissent l’ordre *premier entré, premier servi* (FIFO)
    - les ordres d’envoi et de réception sur un lien sont les mêmes

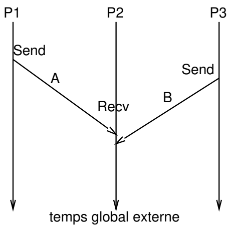
Liens fiables, FIFO, et temps global (cont’d)

- Liens fiables et FIFO $\Rightarrow$ deux messages A et B envoyés dans l’ordre par un premier processus vers un second arrivent dans le même ordre



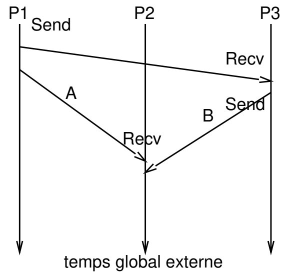


- Rien ne garantit l’ordre d’arrivée de deux messages A et B envoyés par des processus différents



Liens fiables, FIFO, et temps global (cont’d)

- Le problème est l’absence d’un *temps global* commun à tous les processus
- On raisonne avec un *temps local* sur chacun des processeurs
- La notion d’*ordre causal* :
  La cause précède l’effet
  - l’émission d’un message précède sa réception
  - `i1 ; i2` : l’instruction `i1` précède `i2`
- Liens fiables et FIFO
  $\not\Rightarrow$ respect de l’ordre causal



Communications synchrones et asynchrones

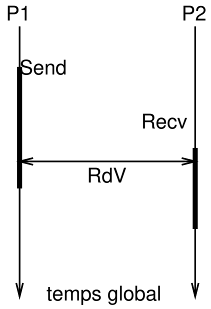
- Réalisation d'une communication point-à-point :
  - un expéditeur réalise une opération d'envoi du message
  - un destinataire réalise l'opération de réception
- Si ces deux opérations ont faites « en même temps » :
  - *mode de communication synchrone* (ou rendez-vous)
  - analogie avec une communication téléphonique : la communication met en présence les deux interlocuteurs
- Si ces deux opérations se font à des instants quelconques :
  - *mode de communication asynchrone*
  - analogie avec le courrier postal : l'expéditeur poste une lettre que le récepteur recevra plus tard

Communications synchrones et asynchrones *(cont'd)*

- *Communication par rendez-vous* : réalisation quand les deux sites sont prêts à communiquer

⇒ Le premier site prêt doit attendre l'autre

- Conséquences :
  - temps d'attente peuvent être longs
  - l'expéditeur est assuré que son message est arrivé au destinataire
- Commentaire sur le terme *synchrone*:
  - les différents sites ne disposent pas d'un temps global
  - comment parler d'opération synchrone dans ce cadre ?
  - les opérations ne sont pas instantanées et il existe un instant pendant lequel les deux opérations sont présentes

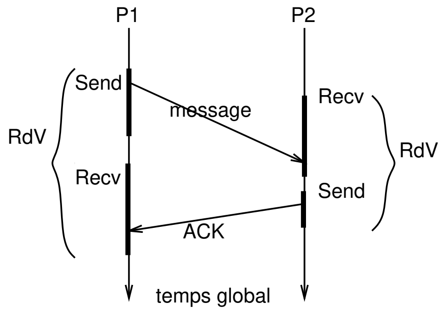


Communications synchrones et asynchrones *(cont'd)*

- *Communication asynchrone*
  - l'émetteur envoie son message dès qu'il est prêt
  - la réception se fait plus tard
- Conséquences :
  - l'émetteur n'attend jamais
  - le récepteur attend si le message n'est pas encore arrivé lors de l'appel de la primitive de réception
  - l'expéditeur ne sait pas, à un moment donné, si le message a déjà été pris en compte par le récepteur
- Communication asynchrone : disponible dans toutes les bibliothèques
- Communication synchrone
  - rarement disponible
  - réalisation à l'aide de plusieurs communications asynchrones

Communications synchrones et asynchrones *(cont'd)*

- *Rendez-vous à l'aide de communication asynchrones*
- Le destinataire renvoie à l'expéditeur un message d'acquittement




```
procedure
  rdv_send (destinataire, mess)
  debut
    send (destinataire, mess)
    recv (destinataire, ACK)
  fin
procedure
  rdv_recv (expediteur, mess)
  debut
    recv (expediteur, mess)
    send (expediteur, ACK)
  fin
```

⇒ Les réceptions doivent indiquer l'expéditeur et le type du message

Communications bloquantes et non-bloquantes

- Les communications asynchrones peuvent être bloquantes ou non-bloquantes
- Mode bloquant
  - primitive `bsend` d'envoi se termine quand le message a quitté l'expéditeur
    (Le message n'est pas nécessairement arrivé au destinataire)
  - primitive `brecv` de réception se termine quand le message est arrivé et a été recopié dans le tampon de réception
- Mode non-bloquant :
  - tente de diminuer le temps d'attente des primitives
  - idée : une primitive déclenche la communication mais n'attend pas la terminaison de l'opération
  - appel à une primitive non-bloquante (`nsend`, `nrecv`) = appeler la couche logiciel sous-jacente et continuer aussitôt

Communications bloquantes et non-bloquantes *(cont'd)*

- Intérêt des communications non-bloquantes :
  - réalisation simultanée de calculs et communications
    $\Rightarrow$ masquer les durées des communications
- Attention : le tampon de communication ne peut pas être réutilisé sans risque
- Terminaison d'une communication non-bloquante ?
  - primitive d'attente (`msgwait`) : bloque l'exécution jusqu'à terminaison de la communication
  - primitive de test (`msgdone`) : informe de la terminaison de la communication
- Plusieurs communications en attente :
  - `nsend` et `nrecv` retournent un identificateur de la communication
  - `msgwait` et `msgdone` utilisent cet identificateur

$$\texttt{bsend (...)} \equiv \begin{cases} id \ = \texttt{nsend (...)} \\ \texttt{msgwait } (id) \end{cases}$$

Mode de communication par interruptions

- Mode défini pour les communications asynchrones
- L'arrivée d'un message génère une interruption au niveau de l'application
- `irvc (tampon, taille, expditeur, type, procdure)`
- À l'arrivée d'un message correspondant
  - le message est placé dans le tampon
  - la procédure est appelée
  - le programme reprend ensuite sont exécution
- Mode de communication rarement disponible
- Solution élégante pour des schémas de communications non-statiques : équilibrage de la charge
- Alternative : vérification à intervalles réguliers de la présence d'éventuels messages

Communications collectives

- Opération mettant en jeu tous les processus de l'application
- Exemple : diffuser des données à tous les autres processeurs
- Possibilité de les coder à l'aide de communication point-à-point
  - source d'erreurs
  - efficacité moindre
- Possibilité de combiner communications et calculs simples (somme, maximum...)
- Sous-ensemble de processus : les groupes de processus
- Exécution d'une communication collective :
  - l'ensemble des processus réalise une série d'opérations correspondant à la communication
  $\Rightarrow$ appel de la même primitive de communication globale par tous les processus

Communications classiquement utilisées en algorithmique parallèle

Transfert Communication point-à-point

Synchronisation (*synchronization*)
- Pas d'échange d'informations
- Tous les processus sont assurés que tous ont ralliés le *point de synchronisation*

Diffusion (*broadcast* ou *one-to-all*)
- Envoi d'un même message
- Depuis un processeur distingué
- À tous les autres (y compris lui-même ?)

Distribution ou diffusion personnalisé (*scattering* ou *personalized one-to-all*)
- Un processus distingué
- Envoie un message distinct
- À chacun des autres processus

utilisées en algorithmique parallèle (cont'd)

Rassemblement (*scattering*)
- Inverse de la distribution
- Un processus distingué
- reçoit un message distinct
- de chacun des autres processus

Commérage (*all-to-all*, *total-exchange*, ou *gossiping*)

« There a n ladies, and each of them knows a scandal which is not known to any of the others.

They communicate by telephone, and whenever two ladies make a call, they pass on to each other as much scandals as they know at the time.

How many calls are needed before all ladies know all the scandals? »
- Chacun des n processus possède une information
- À la fin, tous les processus connaissent les n informations

utilisées en algorithmique parallèle (cont'd)

Transposition (*multi-scattering* ou *all-to-all personalized*)
- Chaque processus effectue simultanément une distribution

Réduction (*reduction*)
- Regroupe sur un processus donné
- Une donnée d
- À partir de n données d_i détenues par chacun des n processus
- Opérateur de réduction $\otimes$: opération associative (et commutative)
- $d = \otimes_i d_i$

Réduction diffusion
- Variante de la réduction
- Chaque processus connaît la valeur finale d

Topologies de communication

- Algorithmes facilement programmable sur des machines de géométrie donnée (par exemple les grilles...)
- Primitive de la bibliothèque pour construire une topologie virtuelle au dessus des processus
- Exemples :
  - communications point-à-point sur une grille torique
  - communications globales sur des lignes ou colonnes virtuelles de processus

Types de données « évolués »

- Communication utilisent un tampon
  Couple (adresse, taille en octets)
- Primitives de plus haut niveau pour transmettre des valeurs de type évolués
  - structurés : vecteurs, matrices...
  - non-structurés : utilisation de tableaux d'indices...
  - définis par l'utilisateur : ...

Traitement de l'hétérogénéité

- Utilisation de machines hétérogènes
  NOW (Network of Workstations), COW (Cluster of Workstations)
  Représentation différentes des données (little-/big-endian...)
- La bibliothèque de communication peut assurer les conversions nécessaires
- Il existe des formats de conversion et bibliothèques standard (XDR)

Traitement de l'hétérogénéité *(cont'd)*

- Conversions nécessaires ?
  - existe-t-il différents types de processeurs ?
    1. gestion statique des processus/processeurs
       $\rightarrow$ on peut répondre *Oui* ou *Non*
    2. gestion dynamique des processus/processeurs
       $\Rightarrow$ *?*
  - si *Oui* ou *?* $\Rightarrow$ il faut réaliser des conversions
- Quelles conversions ?
  Alternatives :
  1. choix d'un format standard
  2. conversion à l'émission si nécessaire
  3. format indépendant

Traitement de l'hétérogénéité *(cont'd)*

1. Choix d'un format standard
   - celui de la majorité par exemple
   - les données qui transitent sur le réseau sont dans ce format
   - un processus « minoritaire » qui utilisent une autre représentation réalise localement la conversion
   - Si deux processus « minoritaires » communiquent $\Rightarrow$ 2 conversions
2. Conversion à l'émission si nécessaire
   - l'émetteur convertit au format du destinataire
   - chaque émetteur doit connaître le format de chaque récepteur (et la procédure de conversion)
3. Format indépendant
   - pour éviter la gestion de la connaissance croisée de tous les formats par tous les émetteurs
   - choix d'un format indépendant
   - un émetteur convertit systématiquement vers ce format
   - un récepteur convertit systématiquement depuis ce format

Liaisons avec les langages

- Interface possible à la bibliothèque depuis les langages standard
  - C
  - Fortran (calcul scientifique)
- Problèmes pour certains langages, exemples :
  - C++ : envoi d'objets quelconques
  - Fortran 90 : section de tableaux

Génération de traces

- Problème de la compréhension de l'exécution de programmes parallèles
- Génération de traces d'exécutions : liée à la bibliothèque de communications
- Utilitaire de visualisation de traces d'exécutions
- Coût non négligeable à l'exécution
- Perturbation du déroulement de l'exécution
- Pas de standards pour le format des traces d'exécutions

Comportement face aux processus légers

- Utilisation des processus légers sur les machines parallèles :
  - Passage d'une activité à une autre quand le processus est bloqué (par exemple sur une communication)
- Problèmes :
  - réentrance de la bibliothèque
  - identification des processus légers

Tolérance aux pannes

- Aspects importants pour les « grosses applications »
- Au mieux dans la bibliothèque :
  - information de l'émetteur que le récepteur ne répond pas
  - détection de la non-réponse d'un processus
- Sur-coût non négligeable

Autres fonctionnalités

- Pour certaines applications d'autres fonctionnalités sont utiles
- Exemples :
  - Communications unilatérales
    - lire ou écrire dans la mémoire d'un processus sans l'interrompre
    - proche du modèle de programmation à mémoire partagée
    - utile pour l'identification des acteurs présents au début de l'application
  - Démarrer un processus sur un autre processeur par un envoi de message
    - messages actifs
    - simplifie la gestion de l'asynchronisme dans les programmes

Bibliothèques de communications

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 → standard
  - efficacité
  - différentes implémentations : MPICH, CHIMP, LAM, constructeurs...
- Évolution
  - PVM → MPI
  - MPI → PVM
  - MPI prend le pas devant PVM