Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Parallélisme & Applications

Corrections TD n°4

Programmation MPI

Modèle Master/Worker

Premier Source : la fonction `main`

- ligne 10 : `MPI_Init` : cette fonction permet de « *connecter* » le processus à la machine parallèle et de définir les paramètres de l'application parallèle, en particulier le nombre de nœuds utilisés ;
  Ce doit être la première instruction à exécuter
- ligne 11 : `MPI_Comm_rank` : cette fonction retourne le numéro du « processeur » courant au sein de l'application parallèle, ce qui permet d'en tenir compte dans l'algorithme ;
- ligne 12 : `MPI_Comm_size` : cette fonction retourne le nombre de processeurs/processus utilisés dans la machine parallèle ;
- ligne 33 : `MPI_Finalize` : termine l'application parallèle.

Second source : la fonction `mastercode`

- ligne 12 : `MPI_Comm_size` : obtenir le nombre de nœuds de l'application parallèle ;
- ligne 14 : on détermine combien de Workers vont s'exécuter dont on devra récupérer un résultat ;
- ligne 19-21 : on tient compte du fait qu'il peut y avoir plus de nœuds dans la machine parallèle, le « *cluster* », que nécessaire dans l'application ;
- ligne 22 à 33 : on transmets le travail à chaque nœud « worker » :
  - ligne 24 : on détermine la borne min de la tranche de tableau que le worker va traiter ;
  - ligne 25 : on détermine la borne max ;
  - ligne 26-27 : on tient compte du fait que la dernière tranche pourrait dépasser la taille du tableau ;
  - ligne 28 : `MPI_Send` : on transmets au worker deux entiers : les bornes min et max de la tranche qu'il doit traiter et on utilise une étiquette de 1 (cette étiquette permet de distinguer au besoin différentes natures de messages) ;
    Ici, le destinataire est identifié de manière précise et unique par la variable `who`
  - ligne 35 à 65 : réception des résultats de la part des workers
    - ligne 38 `MPI_Recv` : on récupère un « `long long` » comme résultat de **n'importe quel** worker en indiquant `MPI_ANY_SOURCE` et en indiquant une étiquette de 1.
    - ligne 46 : on récupère le numéro du nœud qui a envoyé le résultat ;
    - ligne 47 : on fait la somme du résultat intermédiaire avec le résultat courant ;
    - ligne 48 : on incrémente le nombre de résultats obtenus ;
    - ligne 54 à 64 : s'il reste du travail à effectuer on le transmets au nœud qui vient de nous envoyer son résultat ;
  - ligne 71 à 91 : on avertit chaque worker qu'il doit arrêter de travailler : on transmets un entier et un message avec une étiquette de 2 (différente de celle utilisée pour la transmission de travail)
    Ici, le worker est spécifiquement désigné par son numéro.
    - lignes 72 : on transmet le message d'arrêt qui est reconnu par l'étiquette 2 ;
    - lignes 84 : on reçoit du worker le nombre de traitements qu'il a effectué ;
- ligne 92 à 98 : on affiche le résultat de la somme complète et celle reçue de chaque worker.

Troisième source : la fonction `workercode`

- ligne 9 : on récupère son numéro au sein de l'application parallèle ;
- ligne 14 : on reçoit les deux entiers en provenance du *master* en indiquant que l'on accepte nimporte quelle étiquette avec `MPI_ANY_TAG` ;
- ligne 26 : si l'étiquette du message reçu est 2 alors on a terminé son travail ;
- lignes 28 à 54 :
  - lignes 32 à 33 : on calcule la somme locale pour la tranche de données entre la borne min et max reçues ;
  - ligne 34 : on incrémente le nombre de traitement réalisé, c-à-d le nombre de tranches traités ;
  - ligne 35 : on transmet le résultat obtenu au master en utilisant l'étiquette 1 pour le message ;
  - ligne 40 : on reçoit un message depuis le master : soit une nouvelle tranche de tableau à traiter, soit un message d'arrêt ;
- ligne 61 : on transmet au worker le nombre de traitements réalisés en utilisant l'étiquette 7.

La **structure de communication** correspond à celle présentée au début de la fiche : c'est le master qui distribue le travail à chaque worker et qui reçoit ensuite le résultat de ce travail pour le finaliser et l'afficher.