Université Bordeaux 1 Master Informatique **INF 477**
Examen du mardi 15 avril 2008 Durée : 1h30 – Sans document

Programmation des Architectures Parallèles

Prédiction de branchement

Rappeler les différentes stratégies utilisées dans les microprocesseurs modernes pour tenter de prédire si un branchement sera pris ou non. Comparer leur efficacité sur l'exemple suivant :

```
for(i=0 ; i < 1000 ; i++)
        for(j=0 ; j < 10 ; j++)
            if (j & 1) // i est impair
              k++ ;
```

Décrire une expérience permettant d'apprécier la qualité de l'unité de prédiction d'un processeur.

Programmation OpenMP

Il s'agit de paralléliser le plus efficacement possible la boucle suivante (en modifiant au besoin le code):

```
for(i=0 ; i < 1000 ; i++)
   s += f(i) ;
```

1. En supposant que le temps de calcul de f(i) ne dépends pas de la valeur de i ;
2. En supposant que le temps de calcul de f(i+1) est toujours (très) supérieur à celui de f(i).

Programmation MPI

On désire paralléliser *efficacement* le traitement d'une séquence d'images en utilisant une grappe de pc et en adoptant un schéma de programmation de type maître/esclave. Le maître distribue la séquence image par image aux esclaves inactifs ; le travail (répétitif) d'un esclave consiste à réceptionner une image, la traiter et la retourner au maître ; le maître remplace la version originale de chaque image par celle traitée.

Écrire le pseudo code du programme en supposant qu'une image est codée par 3 tableaux (RVB) de 1024*768 octets et que l'on dispose de N esclaves. On précisera bien les appels MPI.

Programmation des architectures distribuées

Expliquer pourquoi la communauté des programmeurs parallèles préfère actuellement l'approche de programmation de type MPI pure aux approches hybrides telles MPI + OpenMP ou MPI + pthread en matière de programmation des grappes de multiprocesseurs.

1 - Environnement
Initialiser MPI et quitter MPI :
int MPI_Init(int *argc,char*** argv)
int MPI_Finalize(void)
Quitter brutalement MPI :
int MPI_Abort(MPI_Comm comm)
Savoir si un processus à fait un MPI_Init :
int MPI_Initialized(int *flag)
Récupérer la chaine de caractères associée au code d'erreur err:
int MPI_Error_string(int errcode,char *chaine,int *taille_chaine)

2 - Communications point à point bloquantes
Envoyer un message à un processus :
int MPI_[R,S,B]send(void *buf,int count,MPI_Datatype datatype,int dest,int tag,MPI_Comm comm)
Recevoir un message d'un processus :
int MPI_Recv(void *buf,int count,MPI_Datatype datatype,int source,int tag,MPI_Comm comm,MPI_Status *status)
Envoyer et recevoir un message :
int MPI_Sendrecv(void *sendbuf,int sendcount,MPI_Datatype sendtype,int dest,int sendtag,void *recvbuf,int recvcount,MPI_Datatype recvtype,int source,int recvtag,MPI_Comm comm,MPI_Status *status)
Compter le nombre d'éléments reçus :
int MPI_Get_count(MPI_status *status,MPI_Datatype datatype,int *count)
Tester l'arrivée d'un message :
int MPI_Probe(int source,int tag,MPI_Comm comm,MPI_Status *status)

3 - Communications point à point non bloquantes
Commencer à envoyer un message :
int MPI_I[r,s,b]send(void *buf,int count,MPI_Datatype datatype,int dest,int tag,MPI_Comm comm,MPI_Request * request)
Commencer à recevoir un message :
int MPI_Irecv(void *buf,int count,MPI_Datatype datatype,int source,int tag,MPI_Comm comm,MPI_Request *request)
Compléter une opération non bloquante :
int MPI_Wait(MPI_Request *request,MPI_Status *status)
Tester une opération non bloquante :
int MPI_Test(MPI_Request *request,int *flag,MPI_Status *status)
Libérer une requête avant de la réutiliser :
int MPI_Request_free(MPI_Request *request)
MPI_Test checks the status of a specified non-blocking send or receive operation. The "flag" parameter is returned logical true (1) if the operation has completed, and logical false (0) if not. For multiple non-blocking operations, the programmer can specify any, all or some completions.
int MPI_Testany(int count,MPI_Request *array_of_requests,int *index, int *flag,MPI_Status *status)
int MPI_Testall(int count,MPI_Request *array_of_requests, int *flag,MPI_Status *array_of_statuses)
int MPI_Testsome(int incount,MPI_Request *array_of_requests, int *outcount,int *array_of_indices, MPI_Status *array_of_statuses)
int MPI_Iprobe(int source,int tag,MPI_Comm comm,int *flag, MPI_Status *status)
int MPI_Cancel(MPI_Request *request);
MPI_Waitall(), MPI_Cancel MPI_Iprobe(), MPI_Test_cancelled()

4 - Communications persistantes
Décrire un schéma persistant :
int MPI_[R,S,B]send_init(void *sendbuf,int count,MPI_Datatype datatype,int dest,int tag,MPI_Comm comm,MPI_Request *request)
int MPI_Recv_init(void *recbuf,int count,MPI_Datatype datatype,int dest,int tag,MPI_Comm comm,MPI_Request *request)
Démarrer une communication persistante :
int MPI_Start(MPI_Request *request)
Autres fonctions : MPI_Startall(), MPI_Request_free()

5 - Communications collectives
Diffusion générale d'un message :
int MPI_Bcast(void *buf,int count,MPI_Datatype datatype,int root,MPI_Comm comm)
Collecte de données :
int MPI_Gather(void *sendbuf,int sendcount,MPI_Datatype sendtype,void *recvbuf,int recvcount,MPI_Datatype recvtype ,int root,MPI_Comm comm)
Diffusion sélective d'un message :
int MPI_Scatter(void *sendbuf,int sendcount,MPI_Datatype sendtype,void *recvbuf,int recvcount,MPI_Datatype recvtype ,int root,MPI_Comm comm)
Collecte de données et rediffusion :
int MPI_Alltoall(void *sendbuf,int sendcount,MPI_Datatype sendtype,void *recvbuf,int recvcount,MPI_Datatype recvtype,MPI_Comm comm)
Calcul d'une réduction :
int MPI_Reduce(void *sendbuf,void *recvbuf,int count,MPI_Datatype datatype,MPI_Op operation,int root,MPI_Comm comm)
Calcul d'une réduction et rediffusion du résultat :
int MPI_Allreduce(void *sendbuf,void *recvbuf,int count,MPI_Datatype datatype,MPI_Op operation,MPI_Comm comm)
Synchronisation de processus :
int MPI_Barrier(MPI_Comm comm)

6 - Types dérivés
Construire un type de données contigües :
int MPI_Type_contiguous(int nbre,MPI_Datatype ancien_type,MPI_Datatype *nouveau_type)
Type de données distantes d'un pas constant :
int MPI_Type_[h]vector(int nbre,int taille_bloc,MPI_Aint pas,MPI_Datatype ancien_type,MPI_Datatype *nouveau_type)
Type de données distantes d'un pas variable :
int MPI_Type_[h]indexed(int nbre,int *taille_bloc,MPI_Aint *pas,MPI_Datatype ancien_type,MPI_Datatype * nouveau_type)
Contruire un type structuré :
int MPI_Type_struct(int nbre,int *taille_bloc,MPI_Aint *pas,MPI_Datatype *anciens_types,MPI_Datatype *nouveau_type)
Valider un type :
int MPI_Type_commit(MPI_Datatype *datatype)
Routine portable pour retourner l'adresse d'une variable :
int MPI_Address(void *variable, MPI_Aint *adresse)
Autres fonctions :
MPI_Type_free, MPI_Type_extent, MPI_Type_size(), MPI_Type_[u,l]b()

7 - Constantes
Jokers :
MPI_ANY_TAG, MPI_ANY_SOURCE
Datatypes élémentaires :
MPI_CHAR, MPI_SHORT, MPI_INT, MPI_LONG, MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE, MPI_UNSIGNED, MPI_UNSIGNED_CHAR, MPI_UNSIGNED_SHORT ,MPI_UNSIGNED_LONG, MPI_LOGICAL, MPI_BYTE, MPI_PACKED
Constantes reservées :
MPI_PROC_NULL, MPI_UNDEFINED
Communicateurs reservés :
MPI_COMM_WORLD, MPI_COMM_SELF
Opérateurs de MPI_Reduce et MPI_Allreduce :
MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_BAND, MPI_BOR, MPI_BXOR, MPI_LAND, MPI_LOR, MPI_LXOR

8 – MPI_Status

status.source status.tag status.error status.length status.size status.bytes