Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Master 1ère année

Parallélisme & Applications

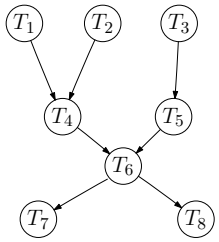
Examen — mai 2013

Durée : 2h — Documents autorisés

Programmation OpenMP – 6 points

1 – 2pts Donnez une version OpenMP du programme :

- ▷ décomposé en 8 « *tasks* », $\{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8\}$
- ▷ l'exécution de chacune de ces « *tasks* » s'organisant suivant le graphe de dépendances temporelles ci-contre :



Pour indiquer dans votre programme que vous exécutez la « task » T_i, vous écrirez le commentaire :

```
1  /* Task i */
```

2 – 2pts Donnez une version du programme suivant, en utilisant uniquement la directive « `#pragma omp parallel sections` » :

```
1  int a[230];

3  #pragma omp parallel for schedule(static,50)
4   for (i=0; i<230; i++)
5    a[i] = super_fonction_tres_complexe(i);
```

3 – 2pts Soit une machine multiprocesseur disposant de 100 cœurs et un tableau d'entier, `t[N]`, avec `N=100`. Écrire un programme OpenMP de complexité $O(1)$ permettant de trouver si la valeur 20 est présente dans le tableau.

Questions de cours – 4 points

4 – 4pts Soit la séquence d'intructions suivante :

```
1  R3 = R1 * R1;
2  R4 = R0 * R2;
3  R4 = R4 * 4;
4  R0 = R3 - R4;
5  R2 = SQRT R0;
6  R3 = -R1 - R2;
7  R4 = R2 - R1;
```

a. Qu'est-ce que le « critère de Bernstein » permet de vérifier ?

b. Quelle sorte de parallélisme peut-on exploiter ?

c. Donnez un graphe permettant de connaître les possibilités d'exploitation du parallélisme.

d. Est-il possible de tirer parti de l'effet « *pipeline* » ? et Comment ?

CUDA – (10 points)

5 – 2pts Soit les lignes suivantes :

```
1  #define TAILLE 1000
2  ...
3  dim3 dimBlock(25,25);
4  dim3 dimGrid(TAILLE/25,TAILLE/25);
5  float *table = (float *) malloc(TAILLE*TAILLE*sizeof(float));
6  ...
7  mon_kernel<<<dimGrid, dimBlock>>>(ref_table)
```

a. Combien de threads CUDA vont être créées ?

b. Comment va se faire la correspondance entre les données de la variable `table` et les différentes threads ?

c. Donnez la formule de calcul à inclure dans le « *kernel* » qui permet d'associer une thread à une case de la table.

Rappel sur les accès tableau en C

```
1  int a[N][N];
2  int i = 2;
3  int j = 3;
4  a[i][j] = 5;
5  printf("%d\n",a[j+i*N]);
```

Les notations :
- ☐ `a[i][j]`
- ☐ `a[j+i*N]`

sont équivalentes.

6 – 8pts Soit une table 2D de type `char` contenant de manière aléatoire les caractères `'A'`, `'B'`, `'C'` et `'D'` :

A	B	B	A	D	C	A	B	C	D	...
B	B	D	A	A	C	C	A	B	C	...
C	D	B	C	B	B	D	C	A	C	...
C	A	C	D	A	C	C	A	D	B	...
B	C	D	D	A	B	D	C	B	C	...
A	B	C	A	C	D	A	B	A	D	...
A	D	C	B	C	A	B	A	B	D	...
C	B	A	B	D	A	C	D	C	C	...
B	C	C	A	B	D	D	C	D	A	...
D	A	B	C	C	A	C	B	B	C	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

La table 2D, appelée « `table_occurences` », est :
- ▷ déclarée suivant une dimension de `1000x1000`
- ▷ remplie par la fonction de prototype : `void init_table(char *t);`

On veut calculer le **nombre d'occurrences** de chacun de ces caractères dans la table, suivant différentes méthodes.

a. On utilisera pour lancer le kernel « `calcul_nb_car_methode1` » l'instruction suivante : *(2pts)*

```
1  dim3 grille(1000, 1000);
2  calcul_nb_car_methode1<<<grille,1>>>(ref_table,...);
```

Combien de threads vont être créées ?
Comment une thread va être associée à une case de la table 2D ?
Comment le calcul va se dérouler ?
Quelles sont les structures de données nécessaires pour effectuer le calcul (sur le CPU et le GPU) ?
Est-ce intéressant en terme d'exploitation du parallélisme ?
Vous discuterez et justifierez en fonction des accès concurrents, de la mémoire utilisée, de l'efficacité, du travail que peut réaliser le kernel par rapport à ses paramètres de lancement.

b. On utilisera pour lancer le kernel « `calcul_nb_car_methode2` » l'instruction suivante : *(2pts)*

```
1  dim3 grille(10, 10);
2  dim3 bloc(100, 100);
3  calcul_nb_car_methode2<<<grille,bloc>>>(ref_table,...);
```

Comment une thread va être associé à une case de la table 2D ?
Quelles sont les structures de données nécessaires pour effectuer le calcul (sur le CPU et le GPU) ?
Quelles sont les optimisations que peut apporter la méthode 2 par rapport à la méthode 1 ?
Est-ce intéressant en terme d'exploitation du parallélisme ?
Vous discuterez et justifierez en fonction des accès concurrents, de la mémoire utilisée, de l'efficacité, du travail que peut réaliser le kernel par rapport à ses paramètres de lancement.

c. Vous **écrirez le code** : *(4pts)*
- ◇ d'un kernel « `calcul_nb_car_methode3` » implémentant une **méthode efficace** pour réaliser le calcul en choisissant et, en indiquant, ses paramètres de lancement ;
- ◇ de l'allocation de mémoire sur la carte GPU ;
- ◇ de la déclaration/allocation des données nécessaires sur le CPU ;
- ◇ de transfert des données du CPU $\Longrightarrow$ GPU et GPU $\Longrightarrow$ CPU ;
- ◇ de finalisation du traitement sur le CPU si nécessaire.

Vous justifierez votre choix d'implémentation.