Systèmes et supports d'exécution pour le calcul parallèle et distribué

Université de Bordeaux 1 — Master 2 SDRP, 2008 – 2009

Partiel

Alexandre DENIS

28 novembre 2008

Durée : 2 h. **Documents interdits.**
Les exercices sont indépendants les uns des autres. Le barème est donné à titre indicatif.
Les parties de code demandées doivent être rédigées en langage C.

1 Questions de cours (7 points)

1. Définissez brièvement :
   (a) système distribué
   (b) processeur superscalaire
   (c) processeur multithread
   (d) machine SMP, machine NUMA
2. Il existe actuellement de nombreux processeurs qui utilisent du parallélisme en interne.
   (a) Pourquoi a-t-on introduit du parallélisme à l'intérieur des processeurs ? Justifiez.
   (b) Quelles formes peut prendre le parallélisme à l'intérieur des microprocesseurs ?
   (c) Comment utilise-t-on, au niveau logiciel, ces différentes formes de parallélisme offertes par les microprocesseurs ?
3. Qu'appelle-t-on une construction d'intergiciel par abstraction ?
4. Qu'appelle-t-on la *portabilité* ?

2 Questions de réflexion **(6 points)**

1. Quels sont les aspects internes des bibliothèques de *threads* qui sont intrinsèquement non-portables (ou difficilement portables) ? Par rapport à quels types de ressources la question de la portabilité se pose-t-elle pour les *threads* ? Vous pourrez distinguer le cas des *threads* utilisateurs et des *threads* noyau.
2. La bibliothèque de *threads* Marcel dispose d'une API qui lui est propre mais également d'une API `pthread`.
   (a) Que représente cette API `pthread` pour *Marcel* ?
   (b) Quelle en est l'utilité ?
3. La bibliothèque `GNUPth` (GNU Portable Thread) de Ralf S. Engelschall est une bibliothèque de threads écrite en C et 100 % portable. Elle a la double particularité de fonctionner uniquement en espace utilisateur et d'être non préemptive. À quoi peut-elle servir ? Expliquez en quelques lignes l'intérêt, l'utilité de threads utilisateurs non préemptifs.
4. La notion de thread et la notion de tâche se distinguent par le fait que la création et l'instanciation d'un thread sont une seule et même étape alors que la création et l'intanciation d'une tâche sont deux étapes distinctes. Commentez en quelques lignes les bénéfices qu'offrent les tâches en dissociant ces deux opérations

3 Problème : compilateur OpenMP (7 points)

Nous étudions dans ce problème la parallélisation d'un code à l'aide d'OpenMP et la transormation de code faite par le compilateur.

Soit le code suivant calculant la transformée en ondelettes (DWT — *Discrete Wavelet Tranform*) d'un vecteur `input` de taille `n`. Le résultat est dans `output`. (*Note :* le contenu de `input` est détruit par cette fonction !)

```
void DWT(int*input, int*output, int n)
{
  int l;
  for(l = n / 2; l > 1 ; l = l / 2)
  {
    int i;
    for(i = 0; i < l; i++)
    {
      output[i]   = input[i*2] + input[i*2+1];
      output[i+l] = input[i*2] - input[i*2+1];
    }
    memcpy(input, output, l*2*sizeof(int));
  }
}
```

3.1 Parallélisation

1. Quelle(s) boucle(s) `for` peut être parallélisée(s) en OpenMP. Pourquoi ?
2. Donnez l'idée générale d'une parallélisation de ce code en OpenMP. Quelles variables doivent être partagées entre les *threads* et lesquelles doivent être locales ?
3. Sur `P` processeurs, quelle est l'accélération attendue d'une version OpenMP ? Justifiez.

3.2 Transformation OpenMP → C

Le pragma OpenMP `for` peut recevoir un attribut optionnel `schedule` pour guider la répartition de la charge, pouvant prendre entre autres les valeurs suivantes :

`static` — pour une boucle allant de 0 à N-1, chaque *thread* reçoit le travail correspondant à `N/P` itérations de la boucle.

`dynamic` — chaque itération est considérée comme une tâche, les *threads* piochent du travail dans la liste de tâches jusqu'à ce que tout le travail soit effectué.

1. Proposez une version `pthread` du code `DWT` correspondant à ce qu'un compilateur OpenMP génère pour un ordonnancement `static`.
2. Proposez une version `pthread` du code `DWT` correspondant à ce qu'un compilateur OpenMP génère pour un ordonnancement `dynamic`.
3. Quelle est l'utilité du mode `dynamic` d'un point de vue général ?
4. L'ordonnancement `dynamic` est-il nécessaire dans le cas du code `DWT` ? Justifiez.