Exemples d'applications de la programmation parallèle

Module 10

INF8601 Systèmes informatiques parallèles

Michel Dagenais

École Polytechnique de Montréal
Département de génie informatique et génie logiciel

Sommaire

Exemples d'applications

- Décomposition à fine granularité avec organisation MapReduce
- Préfixe parallèle
- Calcul matriciel
- Tri parallèle

Exemples d'applications de la programmation parallèle

Map Reduce

- Le calcul parallèle suivi d'une réduction est un patron commun, bien supporté par OpenMP, MPI...
- Cadriciels génériques pour le calcul parallèle basés sur ce paradigme: Google Map Reduce, Hadoop, Apache Spark...
- Idéalement, chaque serveur de calcul possède déjà les données à traiter grâce à un système de fichiers distribué comme HFS (Hadoop File System).
- Apache Spark évite de tout écrire sur disque, par exemple pour le traitement en ligne, et peut dans certains cas être beaucoup plus rapide que Hadoop.

Les phases de Map Reduce

- L'entrée est décomposée en morceaux selon des critères implicites (e.g. ordre d'arrivée) ou explicites (e.g. localisation spatiale, intervalles de numéro d'identification, URL).
- La même fonction (programme) est appliquée (map) à toutes les entrées et produit une sortie clé-valeur.
- Les sorties peut être envoyées vers les noeuds pour la réduction selon les clés (shuffle).
- La réduction (reduce) combine les sorties pour produire le résultat final.

Exemple: compter les mots avec Java/Hadoop

```
// Tutorial from https://hadoop.apache.org/

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

Exemple: compter les mots avec Java/Hadoop

```
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedExceptio
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

Tolerance aux pannes

- Chaque morceau de fichier est présent en plusieurs (e.g. 3) copies.
- Plusieurs entrées peuvent être envoyées à chaque noeud dans la grappe de calcul parallèle.
- Si un noeud tombe en panne, les entrées peuvent être réparties sur d'autres noeuds.
- Le délai encouru en cas de panne demeure faible.

Exemples d'applications de la programmation parallèle

Le problème

- Une opération associative mais non commutative est appliquée à une suite d'éléments, et produit en sortie une autre suite, dont chaque élément dépend de tous les éléments précédents ou égal en entrée (e.g. $a_i = \sum_{j=1}^{i} a_j$).
- Problème très simple à programmer dans une boucle mais difficile à paralléliser.
- Quelques stratégies de parallélisation existent comme procéder en deux passes ou faire un arbre de calcul.
- MPI offre la fonction MPI_Scan avec un choix d'opérations à appliquer comme addition, maximum, minimum...
- TBB offre parallel_scan.

Préfixe parallèle en 2 passes

- Les données sont divisées en plusieurs morceaux à traiter en parallèle (un morceau par thread ou par noeud).
- Lors d'une première passe, l'opération est effectuée localement, sans tenir compte du préfixe venant des morceaux précédents.
- Ensuite, une réduction est effectuée pour calculer le préfixe pour chaque morceau.
- Lors de la seconde passe, l'opération est effectuée à nouveau sur chaque morceau mais cette fois-ci en tenant compte du préfixe venant du morceau précédent.

Exemple de calcul en 2 passes avec OpenMP

```
void seqprfsum(int *u, int m)
{ int i = 1, s=u[0]; for(; i < m; i++) u[i] += s; s = u[ i ]; }

void parprfsum (int *x, int n, int *z)
{ #pragma omp parallel
  { int i, j, me = omp_get_thread_num ( ),
        nth = omp_get_num_threads( ), chunksize = n / nth,
        start = me * chunksize;
    seqprfsum(&x[start], chunksize);

    #pragma omp barrier
    #pragma omp single
    { for(i = 0; i < nth - 1; i++) z[i] = x[(i+1) * chunksize - 1];
      seqprfsum(z ,nth -1);
    }
    if(me > 0 ) for(j = start; j < start + chunksize; j ++) x[j] += z[me - 1];
  }
}
```

Exemple de décodage RLE

```
void uncomprle(int *x, int nx, int *tmp, int *y, int *ny)
{ int i, nx2 = nx / 2;
  int z[MAXTHREADS];
  tmp[0] = 0;
  for(i = 0; i < nx2; i++) tmp[i + 1] = x[2 * i];
  parprfsum(tmp + 1, nx2 + 1, z);

  #pragma omp parallel
  { int j, k;
    #pragma omp for
    for(j = 0; j < nx2; j++) {
      int start = tmp[j];
      int val = x[2 * j +1];
      int nrun = x[2 * j];
      for(k = 0; k < nrun; k++) y[start + k] = val;
    }
  }
  *ny = tmp[nx2];
}
```

Calcul en arbre

- Dans la barrière en *Papillon*, chaque envoie à un autre à chacune de $\log_2(n)$ étapes (i+1 % n, i+2 % n, i+4 % n...).
- Somme en *papillon*, chaque noeud ajoutant un élément de la séquence (plutôt que 1 pour une barrière).
- Si on veut la somme courante ($\sum_{j=1}^{i} a_i$), plutôt que la somme totale, il suffit de sauter des étapes. On peut vérifier par substitution que $x_6 = (x_0 + (x_1 + x_2)) + ((x_3 + x_4) + (x_5 + x_6))$

$x_1 \leftarrow x_0 + x_1$
$x_2 \leftarrow x_1 + x_2$
$x_3 \leftarrow x_2 + x_3$
$x_4 \leftarrow x_3 + x_4$
$x_5 \leftarrow x_4 + x_5$
$x_6 \leftarrow x_5 + x_6$
$x_7 \leftarrow x_6 + x_7$

$x_2 \leftarrow x_0 + x_2$
$x_3 \leftarrow x_1 + x_3$
$x_4 \leftarrow x_2 + x_4$
$x_5 \leftarrow x_3 + x_5$
$x_6 \leftarrow x_4 + x_6$
$x_7 \leftarrow x_5 + x_7$

$x_4 \leftarrow x_0 + x_4$
$x_5 \leftarrow x_1 + x_5$
$x_6 \leftarrow x_2 + x_6$
$x_7 \leftarrow x_3 + x_7$

Exemples d'applications de la programmation parallèle

Le calcul matriciel

- Plusieurs applications avec beaucoup de données sont exprimées sous forme de calcul matriciel.
- Simulation de phénomènes physiques en 2D ou 3D.
- Traitement d'image et de vidéo.
- Calcul sur des graphes, par exemple représentant des réseaux routiers ou électriques.
- Traitement de statistiques.
- Apprentissage machine.

Opérations courantes

- Multiplication de matrices.
- Inversion de matrices.
- Calcul de déterminant et de valeurs propres.
- Puissance de matrices (M^n).
- Les matrices éparses sont un cas particulier pour lequel plusieurs raccourcis sont disponibles pour en réduire la taille et l'effort de calcul.

La multiplication de matrices

- Chaque élément du produit de deux matrices carrées de taille n (n^2 éléments) demande n multiplications et additions ($\mathcal{O}(n^3)$).
- Le calcul d'un élément c_{ij} de la matrice $C = AB$ demande la rangée i de A et la colonne j de B.
- Si on a p noeuds disponibles, formant une matrice carrée de $\sqrt{p}$, il faut décomposer la matrice de taille n en sous-matrices de taille $m = n/\sqrt{p}$ (ou en groupes de n/p rangées).
- Chaque noeud s'occupe alors d'une sous-matrice de taille m et accède les m rangées de A et colonnes de B correspondantes.
- On peut envoyer les matrices complètes A et B à chaque noeud mais il est mieux de faire circuler des morceaux de A dans les rangées et de B dans les colonnes de la grappe de calcul.

Exemple multiplication de matrice par rangées

```
int rank, size, row, column, count, i, j, k;
char ch;
float *A, *B, *C, a, b, c, n;
...
MPI_Init(NULL, NULL);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

if(rank==0) { /* Lire A et B carrées de taille size = row*row */ }

MPI_Bcast(&row, 1, MPI_INT, 0, MPI_COMM_WORLD);
int periods[] = {1, 1};
int dims[] = {row, row};
int coords[2]; /* 2 dimensions */
int right = 0, left = 0, down = 0, up = 0; /* voisins */
MPI_Comm cart_comm;
MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 1, &cart_comm );
```

Exemple multiplication de matrice par rangées

```
MPI_Scatter(A, 1, MPI_FLOAT, &a, 1, MPI_FLOAT, 0, cart_comm);
MPI_Scatter(B, 1, MPI_FLOAT, &b, 1, MPI_FLOAT, 0, cart_comm);
MPI_Comm_rank(cart_comm, &rank);
MPI_Cart_coords(cart_comm, rank, 2,coords);
MPI_Cart_shift(cart_comm, 1, coords[0], &left, &right);
MPI_Cart_shift(cart_comm, 0, coords[1], &up, &down);
MPI_Sendrecv_replace(&a,1,MPI_FLOAT,left,11,right,11,cart_comm,MPI_STATUS_IGNOR
MPI_Sendrecv_replace(&b,1,MPI_FLOAT,up,11,down,11,cart_comm,MPI_STATUS_IGNORE);
c = c + a * b;
for(i = 1; i < row; i++) {
  MPI_Cart_shift(cart_comm, 1, 1, &left, &right);
  MPI_Cart_shift(cart_comm, 0, 1, &up, &down);
  MPI_Sendrecv_replace(&a,1,MPI_FLOAT,left,11,right,11,cart_comm,MPI_STATUS_IGN
  MPI_Sendrecv_replace(&b,1,MPI_FLOAT,up,11,down,11,cart_comm,MPI_STATUS_IGNORE
  c = c + a * b;
}
MPI_Gather(&c, 1, MPI_FLOAT, C, 1, MPI_FLOAT, 0, cart_comm);
if(rank == 0) { /* Imprimer le résultat C */ }
MPI_Finalize();
```

Autres opérations matricielles

- Mettre une matrice à la puissance n peut se faire en combinant avantageusement des multiplications, par exemple $A^8 = ((A^2)^2)^2$.
- Inverser une matrice ou solutionner un système d'équations linéaires $Ax = b$, $x = A^{-1}b$
  - Formule de Laplace: $A^{-1} = \frac{1}{\det A}{}^{\mathrm{t}}\mathrm{com}\, A$
  - Elimination de Gauss-Jordan ou Décomposition LU
  - Algorithme itératif de Jacobi
- Les valeurs propres sont difficiles à calculer analytiquement. Elle peuvent se calculer de manière itérative par multiplication, comme avec la méthode de Jacobi $b_{k+1} = \frac{Ab_k}{\|Ab_k\|}$.

Inversion par la méthode itérative de Jacobi

- $b = Ax$ (b vecteur de valeurs, A matrice de coefficients, x vecteur de variables)
- $x = A^{-1}b$.
- D une matrice avec la diagonale de A et O une matrice en enlevant la diagonale de A
- $x_{k+1} = D^{-1}(b - Ox_k)$.

Exemples d'applications de la programmation parallèle

Les algorithmes classiques de tri

- Le tri rapide (quicksort) et le tri fusion (merge sort) sont faciles à paralléliser. La phase initiale du premier (séparer les éléments par le pivot) et la phase finale du second (fusionner à la racine) constituent un goulot d'étranglement.
- Le tri par comparaison, comme le tri en bulle, normalement moins efficace en série, n'est pas si mal en parallèle car plus équilibré.
- Le tri par paquets demeure très efficace en parallèle, si on choisit de bons seuils pour décomposer en paquets. Une phase d'échantillonage permet d'estimer la distribution et choisir de bons seuils.

Exemples d'applications de la programmation parallèle

Conclusion

- Il est intéressant de connaître un certain nombre de stratégies classiques de parallélisation pour des problèmes courants.
- Cependant, chaque plate-forme (OpenMP, OpenCL, MPI) a des particularités différentes qui affectent l'efficacité des algorithmes.
- Les particularités des jeux de données, pour les très grands ensembles, peuvent aussi affecter le choix de l'algorithme (e.g. matrices éparses).
- Il existe une littérature scientifique étendue sur les meilleurs algorithmes pour un très large éventail de problèmes.