Demandez le Programme !

Optimisation logicielle pour les nouvelles architectures parallèles et vectorielles.

Quelques bonnes pratiques algorithmiques et d'utilisation des outils logiciels à travers un exemple.

Un exemple pas si trivial

ENTRÉES : VC(nv,3)
VN(nv,3)
TV(nt,3)

SORTIES : TN(nt,3)
TS(nt)
VA(nv)
S

```
LOOP i=1 à NT
  v1 = TV(i,1)
  v2 = TV(i,2)
  v3 = TV(i,3)
  u = VC(v2) - VC(v1)
  v = VC(v3) - VC(v1)
  w = u x v
  a = ||w||
  n = w / a
  TN(i) = n
  TS(i) = a
END LOOP
```

```
LOOP i=1 à NT
  s = s + TS(i)
END LOOP
```

```
LOOP i=1 à NT
  v1 = TV(i,1)
  v2 = TV(i,2)
  v3 = TV(i,3)
  VA(v1) = VA(v1) + VN(v1) . TN(i)
  VA(v2) = VA(v2) + VN(v2) . TN(i)
  VA(v3) = VA(v3) + VN(v3) . TN(i)
END LOOP
```

Les trois principaux types de boucles

- Travail local : tous les tableaux sont accédés directement via l'indice de boucle principal U(i)

- Réduction : idem, mais on écrit le résultat dans une seule variable scalaire s=s+U(i), (somme, résidu, norme)

- Travail global : on lit de manière indirecte, U(V(i)), ou pire encore, on écrit de cette manière !

Les trois principaux types d'accès à la mémoire

- Lecture / écriture directe : rapide grâce à la largeur de bus de 1024 bits

- Lecture indirecte : peut-être rapide grâce à la mémoire cache

- Écriture indirecte : toujours lente et sujette à conflits !

Étude du code séquentiel

- Boucle de travail local : simple à paralléliser et rapide si on gère bien l'accès mémoire continue et le cache

- Boucle de réduction : assez simple et efficace aussi

- Boucle de travail global : complexe et lente

- Réorganiser les données et les boucles pour un maximum d'accès directs

- Renuméroter les données pour maximiser l'effet des caches (optimisation d'un maillage tétra : x2 sur un laptop et près de x4 sur un serveur à 2 puces !

Synchronisation

- Pas de duplication de données
- À chaque accès mémoire, on déclenche un verrouillage de la donnée et une lecture ou écriture distante
- Ne modifie pas la structure des données et du code
- Dépends uniquement de la latence du système
- Efficace lorsqu'il y a peu de processus (une seule puce multicoeurs) sinon la complexité crois avec le carré du nombre de processus !

Scatter-Gather

- On duplique toutes les données : dans le cas d'un maillage, chaque sommet stocke une valeur locale pour chaque triangle incident

- Très grosse perte de mémoire et de temps d'échange

- Les phases de scatter et gather étant distinctes des étapes de calcul, l'effet de latence est faible et c'est surtout le débit qui compte (grosse machine à mémoire partagée, GPU)

Renumérotation

- Une méthode hybride entre les précédentes mais plus proche de la synchronisation

- Petit surcout mémoire pour stocker la "couleur" de chaque entité du maillage

- Renumérotation initiale assez rapide, il est aussi possible de renuméroter en cours de calcul si les données sont dynamiques

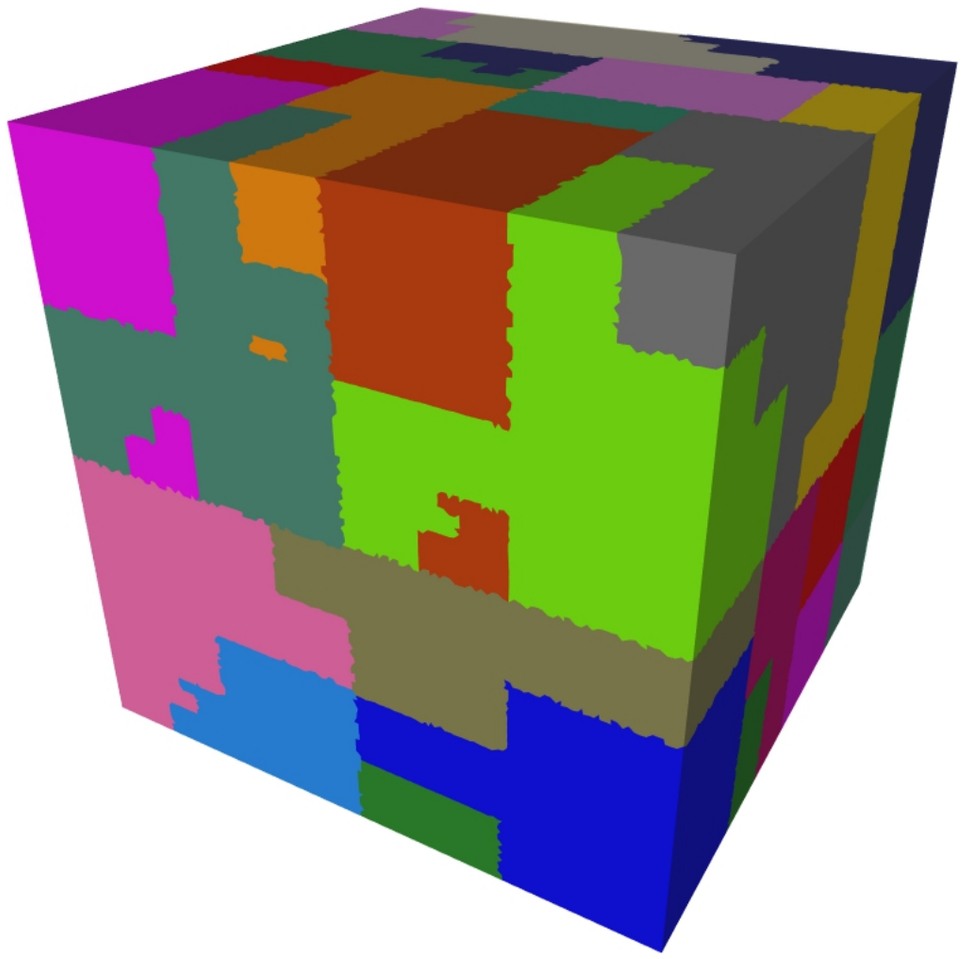
- Encore beaucoup de synchronisation, mais nettement diminuée, car un synchronise seulement des blocs d'éléments de la même couleur

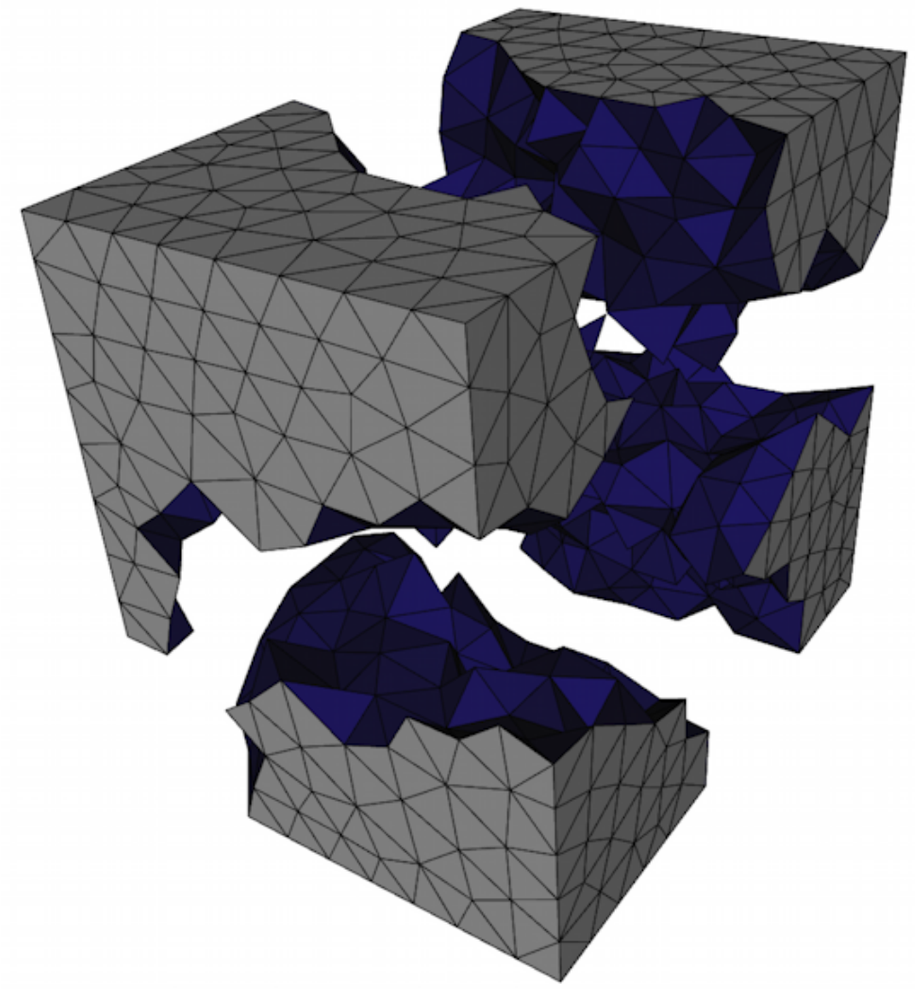
- Efficace avec des systèmes à latence assez bonne comme des clusters infiniband rapides ou des serveurs à 64 coeurs

Partitionnement

- Duplication des données aux interfaces entre les blocs de partitionnement
- Temps de partitionnement assez long
- Efficace si les données sont statiques
- Peu de synchronisations, car le nombre de blocs est minimal
- Échange de données tout de même assez important entre les blocs, efficace sur cluster Ethernet 10gb

Renumérotation / Partitionnement





Parallélisation d'une boucle à lecture ou écriture directe

```
LOOP globale // i=1 à NT
LOOP locale i=début(p) à fin(p)
  v1 = TV(i,1)
  v2 = TV(i,2)
  v3 = TV(i,3)
  u = VC(v2) - VC(v1)
  v = VC(v3) - VC(v1)
  w = u x v
  a = ||w||
  n = w / a
  TN(i) = n
  TS(i) = a
END LOOP
```

Parallélisation d'une boucle de réduction

TEMPORAIRES :
SL(np)

```
LOOP globale // i=1 à NT
LOOP locale i=début(p) à fin(p)
  SL(p) = SL(p) + a
END LOOP
```

```
LOOP i=1 à np
  S = S + SL(i)
END LOOP
```

Parallélisation d'une boucle à écriture indirecte (synchronisation)

```
LOOP globale // i=1 à NT
LOOP locale i=début(p) à fin(p)
  v1 = TV(i,1)
  v2 = TV(i,2)
  v3 = TV(i,3)
  Réserve(v1)
  Réserve(v2)
  Réserve(v3)
  VA(v1) = VA(v1) + VN(v1) . TN(i)
  VA(v2) = VA(v2) + VN(v2) . TN(i)
  VA(v3) = VA(v3) + VN(v3) . TN(i)
  libère(v1)
  libère(v2)
  libère(v3)
END LOOP
```

Parallélisation d'une boucle à écriture indirecte (scatter-gather)

TEMPORAIRES :
VD(nv)
VT(nv,d)
VL(nv,3)
LA(nt,3)
SL(np)

```
LOOP globale // i=1 à NT
LOOP locale i=début(p) à fin(p)
  v1 = TV(i,1)
  v2 = TV(i,2)
  v3 = TV(i,3)
  LA(i,1) = VN(v1) . TN(i)
  LA(i,2) = VN(v2) . TN(i)
  LA(i,3) = VN(v3) . TN(i)
END LOOP
```

```
LOOP globale // i=1 à NV
LOOP locale i=début(p) à fin(p)
  LOOP j=1 à VD(i)
    VA(i) = VA(i) + LA( VT(i,j), VL(i,j) )
  END LOOP
END LOOP
```

Adaptation de la méthode à l'architecture

- Système à faible latence : synchronisation (une puce quad cores)

- Système à latence moyenne : synchronisation avec renumérotation (serveur à deux puces multicoeurs, cluster avec 36 noeuds connectés directement à un switch Infiniband)

- Système à latence élevée et débit moyen : partitionnement (cluster ethernet 10gbits, machine multi-GPU)

- Système à latence et débit élevé : scatter-gather (machine vectorielle, cluster ethernet 100gbits)

Latence vs Débit