

CentraleSupélec

Mineure CalHau1

MPI : communications asynchrones et recouvrement calculs/communications

Stéphane Vialle

Stephane.Vialle@centralesupelec.fr
http://www.metz.supelec.fr/~vialle

CentraleSupélec

MPI-1 : communications asynchrones et recouvrement calculs/comms.

1. Intérêt du recouvrement calculs/communications
2. Communications point à point non bloquantes
3. Communications gérées par des threads explicites
4. Performances expérimentales

CentraleSupélec

MPI-1 : communications asynchrones et recouvrement ...

1 – Intérêt du recouvrement calculs/communications

CentraleSupélec

Intérêt du recouvrement

Quand implanter un recouvrement ?

Mieux vaut (tenter de) recouvrir des temps similaires :

Meilleur cas : $T_{calcul} = T_{comm} \rightarrow \approx 50\%$ de temps économisé

CentraleSupélec

Intérêt du recouvrement

Quand implanter un recouvrement ?

Mieux vaut (tenter de) recouvrir des temps similaires :

Ex 1 :

→ Ne pas implanter de recouvrement dans ce cas :

- code + complexe
- gain faible

Ex 2 :

→ Arrêter de paralléliser ce cas là !

- Majorité du temps passé en communications !!

CentraleSupélec

MPI-1 : communications asynchrones et recouvrement...

2 – Communications point à point non bloquantes de MPI

- Principes
- Ibsend-Irecv
- Issend-Irecv
- Irsend-Irecv
- Identificateurs persistents
- Limites expérimentales

Communications point à point non bloquantes

Principes des comm. Non bloquantes

Principes des communications non-bloquantes

```
MPI_Ixxxx (... , MPI_Request *request)
Ibsend/Irecv - Issend/Irecv - Irsend/Irecv
```

- **Isend ET Irecv** non bloquants
- Un paramètre de plus que leurs homologues bloquants :
→ Id sur la communication demandée (**MPI_Request**)
- Besoin de savoir si les communications sont terminées :
→ **MPI_Wait(request, ...)** ou **MPI_Test(request, ...)**

Comparaison aux communications bloquantes :

- Nécessite une synchronisation explicite – Plus complexe
- Permet de réaliser du recouvrement calcul-communication

Communications point à point non bloquantes

Ibsend – Irecv

Mode *bufferisé* et non-bloquant : **MPI_Ibsend – MPI_Irecv**

Rappel : **Bsend-Recv**

Non-bloquant : **Ibsend-Irecv**

Communications point à point non bloquantes

Issend – Irecv

Mode *synchrone* et non-bloquant : **MPI_Issend – MPI_Irecv**

Rappel : **Ssend-Recv**

Non-bloquant : **Issend-Irecv**

Communications point à point non bloquantes

Irsend – Irecv

Mode *ready* et non-bloquant : **MPI_Irsend – MPI_Irecv**

Rappel : **Rsend-Recv**

Non-bloquant : **Irsend-Irecv**

Communications point à point non bloquantes

Identificateurs persistents

Autre mode d'utilisation des comm. non bloquantes :

```
Loop:
MPI_Issend(..., &s_request); MPI_Irecv(..., &r_request);
..... // calculs
MPI_Wait(&s_request,...); MPI_Wait(&r_request,...);
```

↓

```
MPI_Ssend_init(..., &s_request); MPI_Recv_init(..., &r_request);
Loop:
MPI_Start(&s_request); MPI_Start(&r_request);
..... // calculs
MPI_Wait(&s_request,...); MPI_Wait(&r_request,...);
MPI_Request_free(&s_request);
MPI_Request_free(&r_request);
```

→ Evite de réinitialiser des communications non-bloquantes

Communications point à point non bloquantes

Limites expérimentales

Re-développement en **Ibsend/Irecv**, avec recouvrement calcul/comm :

- Temps de développement ↗
- Complexité ↗
- **Mais performances inchangées!!**

Re-développement en **Bsend/Recv** dans un thread OpenMP :

- Temps de développement ↗
- Complexité ↗
- **Performances accrues 😊**

Le recouvrement calculs/communications fonctionne, mais

- pas toujours avec les communications non bloquantes de MPI
- avec des threads explicites encapsulant des communications MPI bloquantes

MPI-1 : communications asynchrones et recouvrement ...

3 – Communications non bloquantes gérées par des threads explicites

- MatrixProduct en MPI + OpenMP, par code répliqué
- MatrixProduct en MPI + OpenMP, par `#pragma omp sections`

Produit de matrices avec recouvrement calculs/comms.

Même schéma initial de circulation des données (en P étapes)

Même opération : $C = A \times B$

Faire circuler A en parallèle des calculs de l'étape :

Produit de matrices avec recouvrement calculs/comms.

Même schéma initial de circulation des données (en P étapes)

Même opération : $C = A \times B$

Faire circuler A en parallèle des calculs de l'étape :

- Utiliser des *threads* OpenMP dans chaque processus MPI
 - un thread réalisera un `MPI_Sendrecv_replace` (ou un `MPI_Sendrecv`, ou un couple `Bsend/recv...`)
 - les autres threads se répartiront les calcul de l'étape
- Les calculs doivent lire une matrice *A_s* stable :
 - donc lire une copie (dans un buffer *A_s_buf*)

Rmq : Faudra-t-il mettre des `#pragma omp barrier` ?

Comms. gérées par threads explicites

MatrixProduct en MPI + OpenMP

Programmation **synchrone** : exemple 1 avec le **kernel 0**

Calcul et circulation de $As[...]$

```
#pragma omp parallel private(step)
{
  for (step = 0; step < NbStep; step++) {
    // rmk: omp_get_thread_num() -> thread Id.
    //      omp_get_num_threads() -> current number of threads

    ligDebTh = .....; nbLigTh = ..... // Compute a sub-slice
    for (i = ligDebTh; i < ligDebTh + nbLigTh; i++) // of C
      for (j... )
        for (k... )
          C[i][j] += As[i][k]*Bs[k][j]; // optim Bs[j][k]

    #pragma omp barrier // Sync. all threads
    #pragma omp single {
      MPI_sendrecv_replace(As,...); // MPI exchange of As
    }
  }
}
```

Comms. gérées par threads explicites

MatrixProduct en MPI + OpenMP

Programmation **synchrone** : exemple 1 avec le **kernel 1**

Calcul et circulation de $As[...]$

```
#pragma omp parallel private(step)
{
  for (step = 0; step < NbStep; step++) {
    // rmk: omp_get_thread_num() -> thread Id.
    //      omp_get_num_threads() -> current number of threads
    ligDebTh = .....; nbLigTh = ..... // Compute a
    cblas_dgemm(...,nbLigTh,..., // sub-slice of C
                &As[ligDebTh][0],...,
                &Bs[0][0],...,
                &Cs[ligDebTh][0],...);

    #pragma omp barrier // Sync. all threads
    #pragma omp single {
      MPI_sendrecv_replace(As,...); // MPI exchange of As
    }
  }
}
```

Comms. gérées par threads explicites

MatrixProduct en MPI + OpenMP

Prog. **asynchrone et overlapped** : exemple 2 – avec le **kernel 1**

Circulation de $As[...]$ et calcul à partir de $As[...]$


```
#pragma omp parallel private(step)
{
  for (step = 0; step < NbStep; step++) {
    if (omp_get_thread_num() == 0) // 1 thread:
      MPI_sendrecv_replace(As,...); // MPI exchange of As
    else { // n-1 threads:
      // rmk: omp_get_thread_num() -> thread Id.
      //      omp_get_num_threads() -> current number of threads
      ligDebTh = .....; nbLigTh = ..... // Compute a sub-slice
      cblas_dgemm(...,nbLigTh,...,&As[ligDebTh][0],...); // of Cs
    }
  }
}
```

Comms. gérées par threads explicites

MatrixProduct en MPI + OpenMP

Prog. **asynchrone et overlapped** : exemple 2 – avec le **kernel 1**
Circulation de As[...] et calcul à partir de As[...]

```
#pragma omp parallel private(step)
{
  for (step = 0; step < NbStep; step++) {
    if (omp_get_thread_num() == 0)
      MPI_sendrecv_replace(AS, ...);
    } else {
      // rmk: omp_get_thread_num() -> thread Id.
      // omp_get_num_threads() -> current number of threads
      ligDebTh = .....; nbLigTh = ..... // Compute a sub-slice
      cblas_dgemm(..., nbLigTh, ..., &AS[ligDebTh][0], ...); // of Cs
    }
  }
}
```



Comms. gérées par threads explicites

MatrixProduct en MPI + OpenMP

Prog. **asynchrone et overlapped** : exemple 2 – avec le **kernel 1**
Circulation de As[...] et calcul à partir de AsBuf[...]


```
#pragma omp parallel private(step)
{
  for (step = 0; step < NbStep; step++) {
    #pragma omp single // Saving As into AsBuf
    {memcpy(AsBuf, As, ...);}
    if (omp_get_thread_num() == 0) // 1 thread:
      MPI_sendrecv_replace(As, ...); // MPI exchange of A
    } else { // n-1 threads:
      // rmk: omp_get_thread_num() -> thread Id.
      // omp_get_num_threads() -> current number of threads
      ligDebTh = .....; nbLigTh = ..... // Compute a sub-slice
      cblas_dgemm(..., nbLigTh, ..., &AsBuf[ligDebTh][0], ...); // of Cs
    }
  }
}
```

Comms. gérées par threads explicites

MatrixProduct en MPI + OpenMP

Prog. **asynchrone et overlapped** : exemple 2 – avec le **kernel 1**
Circulation de As[...] et calcul à partir de AsBuf[...]

```
#pragma omp parallel private(step)
{
  for (step = 0; step < NbStep; step++) {
    #pragma omp single
    {memcpy(AsBuf, As, ...);}
    if (omp_get_thread_num() == 0)
      MPI_sendrecv_replace(As, ...);
    } else {
      // rmk: omp_get_thread_num() -> thread Id.
      // omp_get_num_threads() -> current number of threads
      ligDebTh = .....; nbLigTh = ..... // Compute a sub-slice
      cblas_dgemm(..., nbLigTh, ..., &AsBuf[ligDebTh][0], ...); // of Cs
    }
  }
}
```




Comms. gérées par threads explicites

MatrixProduct en MPI + OpenMP

Prog. **asynchrone et overlapped** : exemple 2 – avec le **kernel 1**
Circulation de As[...] et calcul à partir de AsBuf[...]

```
#pragma omp parallel private(step)
{
  for (step = 0; step < NbStep; step++) {
    #pragma omp single // Saving As into AsBuf
    {memcpy(AsBuf, As, ...);}
    if (omp_get_thread_num() == 0) // 1 thread:
      MPI_sendrecv_replace(As, ...); // MPI exchange of A
    } else { // n-1 threads:
      // rmk: omp_get_thread_num() -> thread Id.
      // omp_get_num_threads() -> current number of threads
      ligDebTh = .....; nbLigTh = ..... // Compute a sub-slice
      cblas_dgemm(..., nbLigTh, ..., &AsBuf[ligDebTh][0], ...); // of Cs
    }
    #pragma omp barrier // Resync. all threads
  }
}
```



Comms. gérées par threads explicites

MatrixProduct en MPI + OpenMP

Prog. **asynchrone et overlapped** : exemple 2 – avec le **kernel 0**
Circulation de As[...] et calcul à partir de AsBuf[...]


```
#pragma omp parallel private(step)
{
  for (step = 0; step < NbStep; step++) {
    #pragma omp single // Saving As into AsBuf
    {memcpy(AsBuf, As, ...);}
    if (omp_get_thread_num() == 0) // 1 thread:
      MPI_sendrecv_replace(As, ...); // MPI exchange of A
    } else { // n-1 threads:
      // rmk: omp_get_thread_num() -> thread Id.
      // omp_get_num_threads() -> current number of threads
      ligDebTh = .....; nbLigTh = ..... // Compute a sub-slice
      for (i = ligDebTh; i < ligDebTh + nbLigTh; i++) // of Cs
        for (j = .....; j < .....; j++)
          Cs[i][j] += AsBuf[i][k]*Tbs[k][j]; // Bsfj][k]
    }
    #pragma omp barrier // Resync. all threads
  }
}
```

Comms. gérées par threads explicites

MatrixProduct en MPI + OpenMP

Prog. **asynchrone et overlapped** : **VARIANTE** ex 2 avec le **kernel 1**
Circulation de As[...] et calcul à partir de AsBuf[...]

```
#pragma omp parallel private(step)
{
  for (step = 0; step < NbStep; step++) {
    if (omp_get_thread_num() == 0) // 1 thread:
      ... // MPI comm. : send As, recv AsBuf
    } else { // n-1 threads:
      // rmk: omp_get_thread_num() -> thread Id.
      // omp_get_num_threads() -> current number of threads
      ligDebTh = .....; nbLigTh = ..... // Compute on As
      cblas_dgemm(..., nbLigTh, ..., &As[ligDebTh][0], ...);
    }
    ... // Resynchronize all threads if needed
    ... // Management of As and AsBuf arrays (As <-> AsBuf)
  }
}
```



Comms. gérées par threads explicites

MatrixProduct en MPI + OpenMP

Prog. **asynchrone et overlapped** : ex 3 – avec **section** et **task**, kernel 0
Circulation de As[...] et calcul à partir de AsBuf[...]

```
#pragma omp parallel private(step)
{
  for (step = 0; step < NbStep; step++) {
    #pragma omp single // Saving As into
    {memcpy(AsBuf,As,...);} // AsBuf

    #pragma omp sections
    {
      #pragma omp section // MPI exchange of A
      {MPI_sendrecv_replace(A,...);} // (one thread)

      #pragma omp section // Computation on Abuf
      {... AsBuf ...} // (on one thread)
    }
  }
}
```

Comms. gérées par threads explicites

MatrixProduct en MPI + OpenMP

Prog. **asynchrone et overlapped** : ex 3 – avec **section** et **task**, kernel 0
Circulation de As[...] et calcul à partir de AsBuf[...]

```
#pragma omp parallel private(step)
{
  for (step = 0; step < NbStep; step++) {
    #pragma omp single // Saving As
    {memcpy(AsBuf,As,...);} // AsBuf

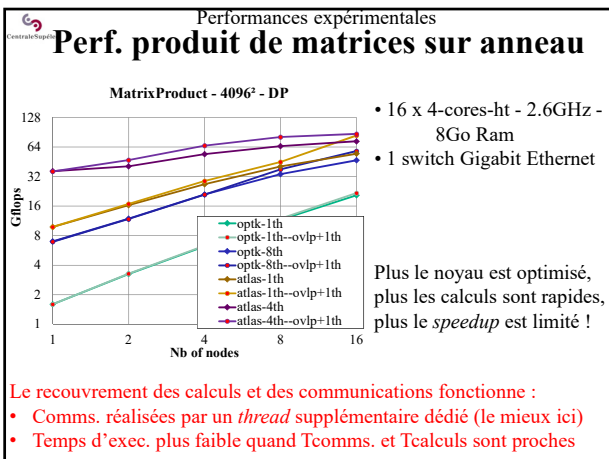
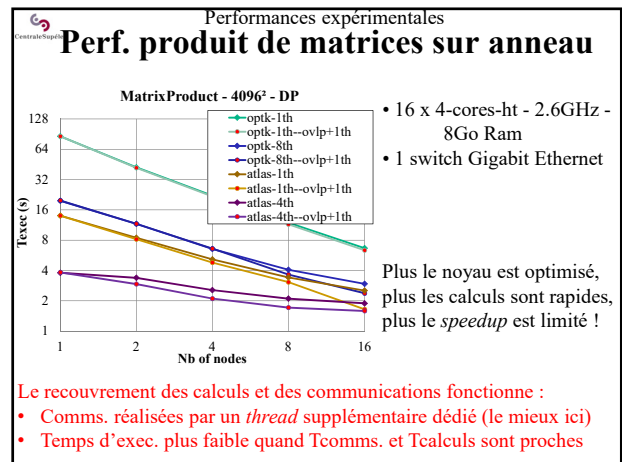
    #pragma omp sections
    {
      #pragma omp section // MPI exchange of As
      {MPI_sendrecv_replace(As,...);} // (one thread)

      #pragma omp section // Computation on
      {... AsBuf ...} // AsBuf
      for (i = 0; i < nbLigSlice; i++)
        #pragma omp task {... AsBuf ...} // (on all available threads)
    }
  }
}
```

To Do:

MPI-1 : communications asynchrones et recouvrement ...

4 – Performances expérimentales du recouvrement calculs/communications

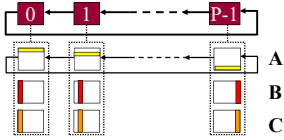


→ TP

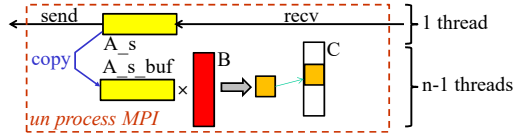
TP : produit de matrices avec recouvrement calculs/comms.

Même schéma initial de circulation des données (en P étapes)

Même opération : $C = A \times B$



Faire circuler A en parallèle des calculs de l'étape :



MPI-1: communications asynchrones et recouvrement calculs/communications

Questions ?