EVIDEN

Efficient HPL on top of runtime systems

Alycia Lisito
Eviden, Inria Bordeaux Sud-Ouest

© Eviden SAS

an atos business

Comment évaluer les machines ?

TOP500

- Créé en 1993 et mis à jour tous les 6 mois
- Classe les 500 supercalculateurs les plus puissants du monde
- Basé sur les performances du HPL
- A atteint l'ExaFlops/s !

© Eviden SAS

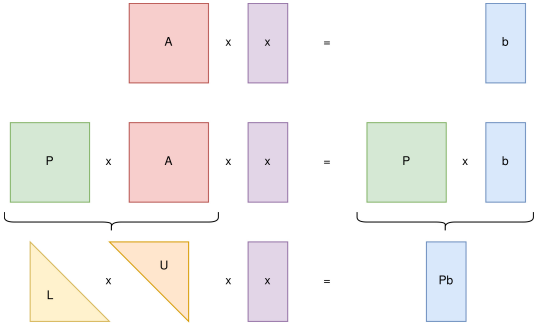
Comment évaluer les machines ?

TOP500

- Créé en 1993 et mis à jour tous les 6 mois
- Classe les 500 supercalculateurs les plus puissants du monde
- Basé sur les performances du HPL
- A atteint l'ExaFlops/s !

HPL (High Performance Linpack) :

- Résout **A x = b** grâce à une factorisation **LU** avec **pivotage partiel**
- Est une implémentation difficilement portable pour les architectures à mémoire distribuée



© Eviden SAS

État de l'art

Implémentations du HPL

- Netlib / ScaLapack
- Sur CPU
  - Intel CPU
  - AMD CPU
  - ARM
- Sur GPU
  - Intel GPU
  - AMD GPU (rocHPL)
  - Nvidia

Spécificité du HPL

- Difficilement portables car optimisé sur un type d'architecture
- Des implémentations principalement homogènes
- Beaucoup de synchronisations
- Optimisations existantes
  - Inner blocking
  - Lookahead
  - Noyaux BLAS optimisés

© Eviden SAS

Notre HPL

Un HPL sur architectures distribuées hétérogènes

Objectifs

- Un HPL portable sur tout type d'architecture
- Un HPL perfomant sur des machines hétérogènes

© Eviden SAS

Notre HPL

Un HPL sur architectures distribuées hétérogènes

Objectifs

- Un HPL portable sur tout type d'architecture
- Un HPL perfomant sur des machines hétérogènes

Notre HPL

- Avec le runtime StarPU dans Chameleon

StarPU et Chameleon

- Support d'exécution parallèle
- Bibliothèque d'algèbre linéaire dense parallèle
- Permet de traiter les algorithmes par tâches
- Support pour tout type de CPU et GPU déjà présent et validé à l'echelle

EVIDEN

© Eviden SAS

Notre HPL

Un HPL sur architectures distribuées hétérogènes

Objectifs

- Un HPL portable sur tout type d'architecture
- Un HPL perfomant sur des machines hétérogènes

Notre HPL

- Avec le runtime StarPU dans Chameleon
- Avec inner blocking
- Avec batching pour les synchronisations
- Adaptation de la granularité des tâches

StarPU et Chameleon

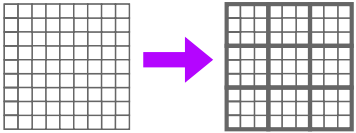
- Support d'exécution parallèle
- Bibliothèque d'algèbre linéaire dense parallèle
- Permet de traiter les algorithmes par tâches
- Support pour tout type de CPU et GPU déjà présent et validé à l'echelle

© Eviden SAS

Algorithme par tâches

Le principe

- La matrice est découpée en blocs
- Les algorithmes sont adaptés
- Chaque opération sur un bloc est considéré comme une tâche

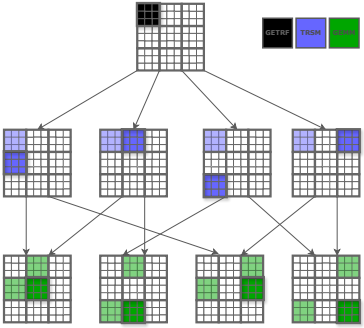


© Eviden SAS

Algorithme par tâches

Le principe

- La matrice est découpée en blocs
- Les algorithmes sont adaptés
- Chaque opération sur un bloc est considéré comme une tâche
- Les tâches sont dépendantes les unes des autres



© Eviden SAS

L'algorithme du GETRF

Du GETRF sans pivotage au pivotage partiel

Factorisation GETRF sans pivotage

Data: k, $A(k:nt,k)$
Result: $A(k:nt,k) \leftarrow L(k:nt,k)U(k,k)$
for $k = 0$ **to** $nt-1$ **do**
  $A_{k,k} \leftarrow$ GETRF$(A_{k,k})$
  for $n = k+1$ **to** $nt-1$ **do**
    $A_{n,k} \leftarrow$ TRSM_RU$(A_{k,k}, A_{n,k})$
    $A_{k,n} \leftarrow$ TRSM_LL$(A_{k,k}, A_{k,n})$
  end
  for $n = k+1$ **to** $nt-1$ **do**
    for $m = k+1$ **to** $nt-1$ **do**
      $A_{m,n} \leftarrow$ GEMM$(A_{m,k}, A_{k,n}, A_{m,n})$
    end
  end
end

Factorisation du panel k pour le GETRF avec pivotage partiel

Data: k, $A(k:nt,k)$
Result: $A(k:nt,k) \leftarrow P_k L(k:nt,k)U(k,k)$
$P_k \leftarrow$ **Id**
for $c = 0$ **to** $nb-1$ **do**
  for $m = k$ **to** $mb-1$ **do**
    $[ipiv(k), mpiv] \xleftarrow{reduce}$ SEARCH_MAX$(A_{m,k}, c)$
  end
  $[A_{k,k}, A_{mpiv,k}] \leftarrow$ SWAP_ROWS$(A_{k,k}, A_{mpiv,k}, c, ipiv_k)$
  for $m = k$ **to** $mb-1$ **do**
    $A_{m,k} \leftarrow$ SCALE_AND_UPDATE$(A_{m,k}, A_{k,k}, c)$
  end
end
$P_k \leftarrow$ IPIV_TO_PERM$(ipiv(:))$

© Eviden SAS

L'algorithme du GETRF

Du GETRF sans pivotage au pivotage partiel

Factorisation GETRF sans pivotage

Data: k, $A(k:nt,k)$
Result: $A(k:nt,k) \leftarrow L(k:nt,k)U(k,k)$
for $k = 0$ **to** $nt-1$ **do**
  $A_{k,k} \leftarrow$ GETRF$(A_{k,k})$
  for $n = k+1$ **to** $nt-1$ **do**
    $A_{n,k} \leftarrow$ TRSM_RU$(A_{k,k}, A_{n,k})$
    $A_{k,n} \leftarrow$ TRSM_LL$(A_{k,k}, A_{k,n})$
  end
  for $n = k+1$ **to** $nt-1$ **do**
    for $m = k+1$ **to** $nt-1$ **do**
      $A_{m,n} \leftarrow$ GEMM$(A_{m,k}, A_{k,n}, A_{m,n})$
    end
  end
end

Factorisation du panel k pour le GETRF avec pivotage partiel

Data: k, $A(k:nt,k)$
Result: $A(k:nt,k) \leftarrow P_k L(k:nt,k)U(k,k)$
$P_k \leftarrow$ **Id**
for $c = 0$ **to** $nb-1$ **do**
  for $m = k$ **to** $mb-1$ **do**
    $[ipiv(k), mpiv] \xleftarrow{reduce}$ SEARCH_MAX$(A_{m,k}, c)$
  end
  $[A_{k,k}, A_{mpiv,k}] \leftarrow$ SWAP_ROWS$(A_{k,k}, A_{mpiv,k}, c, ipiv_k)$
  for $m = k$ **to** $mb-1$ **do**
    $A_{m,k} \leftarrow$ SCALE_AND_UPDATE$(A_{m,k}, A_{k,k}, c)$
  end
end
$P_k \leftarrow$ IPIV_TO_PERM$(ipiv(:))$

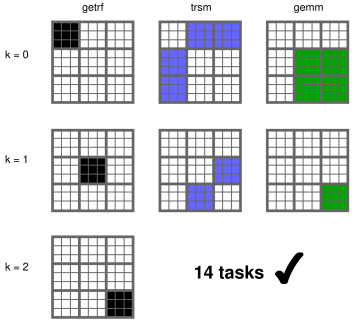
- **Très grand nombre de tâches**
- **Tâches de tailles très petites**

EVIDEN

© Eviden SAS

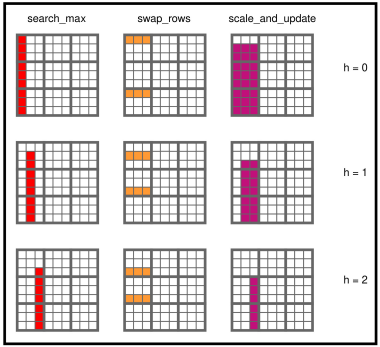
L'algorithme du GETRF

Getrf factorization without pivoting



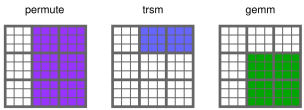
14 tasks ✓

Getrf factorization with partial pivoting



panel **k = 0**

35 tasks ✗



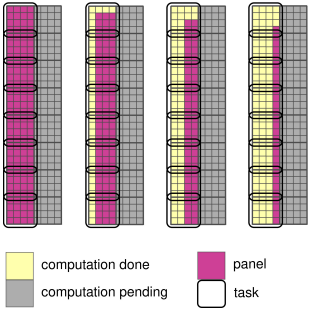
update **k = 0**

© Eviden SAS

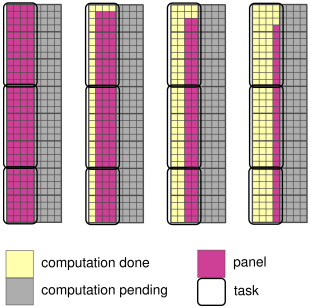
Gérer la granularité des tâches

Le batch dans Chameleon

Sans batching



Avec batching



- Avec pivotage, on a beaucoup de tâches de faible granularité
- Néfaste pour les performances avec un runtime
- On veut réduire le nombre de tâches et augmenter la granularité

- On rassemble les tâches par lot (batching)
- On retient la soumission jusqu'à avoir un lot de tâches prêtes
- Au dernier batch sur la colonne (flush)
- Même stratégie pour la permutation

EVIDEN

© Eviden SAS

Benchmark

The machines

- Inria HPC platform Plafrim
- Half diablo: 2 x 64 cores AMD Zen3

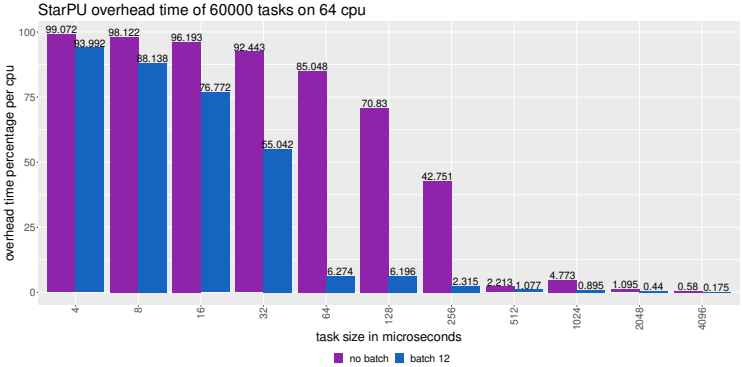
The modules

- mkl 2020
- gcc 11.2
- hwloc 2.10.0
- starpu 1.4.7 (prio scheduler)

© Eviden SAS

Gérer la granularité des tâches

L'overhead de StarPU



- Les plus petites tâches se trouvent en moyenne à 40 microsecondes
- L'effet de l'overhead de StarPU pour ces tâches n'est pas négligeable : environ 90% d'overhead sur le temps total de ces tâches

EVIDEN

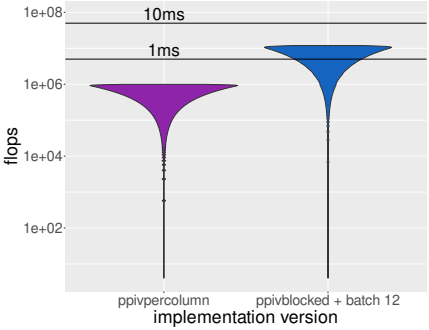
© Eviden SAS

Gérer la granularité des tâches

Le batch: les résultats

	ppivpercolumn	*ppviblocked batch 12*
Tasks nbr	11 482 299	1 020 100
Time (ms/task)	0.151703	1.079328
Flops/task	494 863	5 625 992

(a) Panel tasks statistics of a matrix of size 115200 with a block size of 576

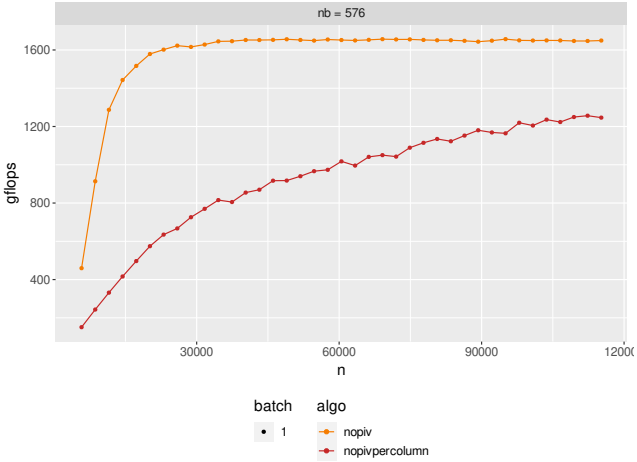


(b) Tasks size repartition

© Eviden SAS

Gérer la granularité des tâches

Implémentation du batching avec l'inner blocking

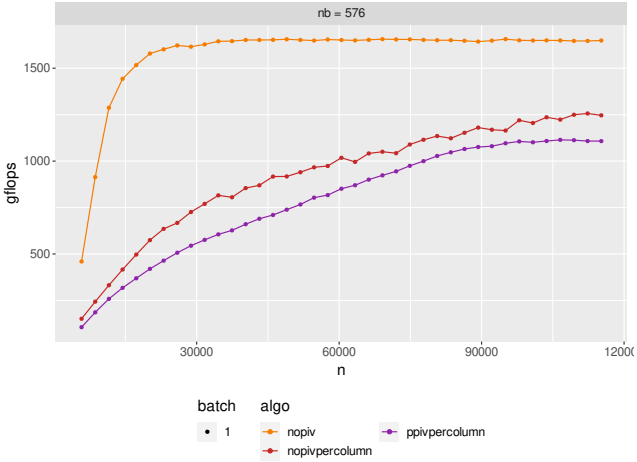


- Le *nopiv* atteint son pic ($1\,600\ GFlops/s$) rapidement
- Le *nopivercolumn* perd beaucoup en performances même sans pivotage

© Eviden SAS

Gérer la granularité des tâches

Implémentation du batching avec l'inner blocking

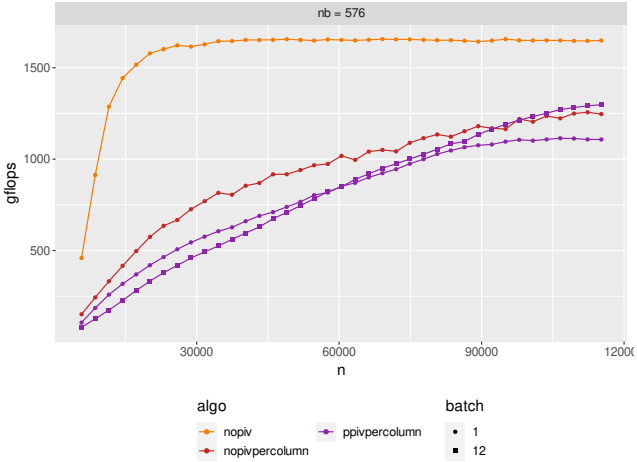


- Le *nopiv* atteint son pic ($1\,600\ GFlops/s$) rapidement
- Le *nopivercolumn* perd beaucoup en performances même sans pivotage
- Avec le pivotage (*ppivercolumn*), les perfomances sont encore plus mauvaises

© Eviden SAS

Gérer la granularité des tâches

Implémentation du batching avec l'inner blocking



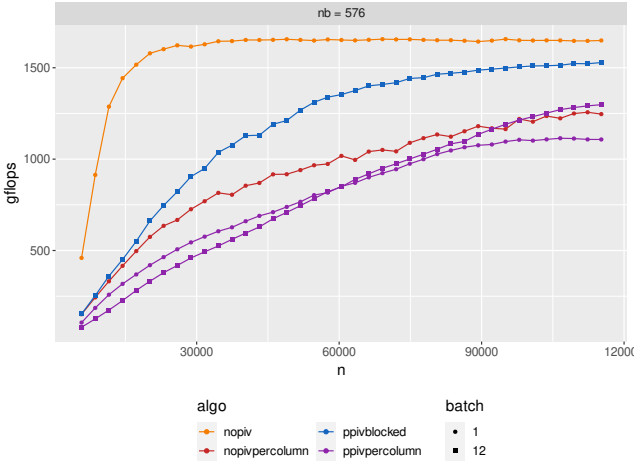
- Le *nopiv* atteint son pic ($1\,600\ GFlops/s$) rapidement
- Le *nopivercolumn* perd beaucoup en performances même sans pivotage
- Avec le pivotage (*ppivercolumn*), les perfomances sont encore plus mauvaises
- Avec le batching, les perfomances sont meilleures sur les grandes tailles

EVIDEN

© Eviden SAS

Gérer la granularité des tâches

Implémentation du batching avec l'inner blocking



- Le *nopiv* atteint son pic ($1\,600\ GFlops/s$) rapidement
- Le *nopivercolumn* perd beaucoup en performances même sans pivotage
- Avec le pivotage (*ppivercolumn*), les perfomances sont encore plus mauvaises
- Avec le batching, les perfomances sont meilleures sur les grandes tailles
- L'inner blocking (*ppivblocked*) permet de regagner en perfomances et plus rapidement
- Avec le inner blocking et batching on est à **93%** de la perf du getrf sans le pivotage

EVIDEN

© Eviden SAS

Conclusion

Contributions

- Une implémentation du HPL avec un runtime
- Du batching pour gérer la granularité des tâches
- L'inner blocking

Perspectives

- Gérer le batch en distribué
- Pouvoir gérer la taille des batchs de façon dynamique
- Étendre le batch à d'autres parties de l'algorithme comme les GEMMs par exemple pour les GPUs
- Faire l'interface de batching au niveau du runtime

© Eviden SAS