Agencement de données out-of-core pour des algorithmes à front d'onde en mémoire partagée

07 juin 2007

Équipe Projet AlGorille

Pierre-Nicolas Clauss
Jens Gustedt
Frédéric Suter

Plan

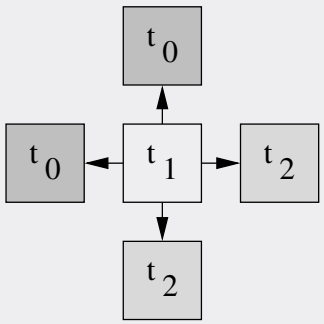
Données utilisées

Données out-of-core

- Plus de données que de mémoire RAM disponible
- Gestion efficace des (dé)chargements de données
- Deux approches
  - Système : modification du gestionnaire de l'OS
  - Algorithmique : structuration du code et mappage de fichiers

Algorithmes à front d'onde

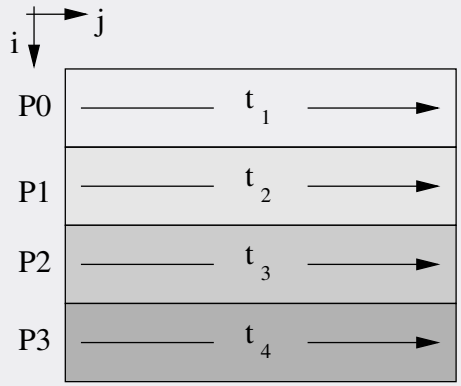
Dépendances de données

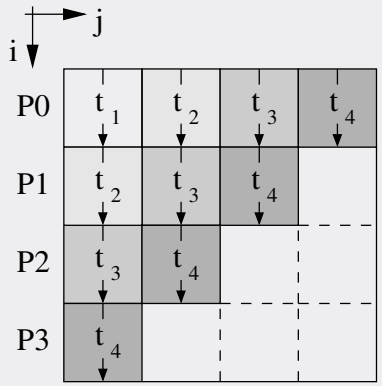


- Calculs (t_1) basés sur les valeurs des voisins proches
  - Voisins mis à jour précédemment (t_0)
  - Voisins mis à jour ultérieurement (t_2)
- Algorithmes itératifs : application jusqu'à convergence

Exécution parallèle de l'algorithme

Parallélisation par pipeline





- Découpage en blocs, petites communications
- Inversion des boucles du parcours (macro-pipeline)
- Deux phases
  - Phase de chargement
  - Régime constant : parallélisme maximal

Réalisation sur architecture distribuée

Recouvrement des phases

- Recouvrement des calculs et des communications avec les entrées/sorties
- Existence d'une taille de bloc optimale
  - Meilleur compromis entre phase de chargement et régime constant

Réalisation sur architecture distribuée

Problématique

- Chargements de données non-contiguës
  - Chargements inefficaces : pénalisation importante
  - Choisir une représentation des données optimisée
- Communications asynchrones
  - Désynchronisation des processus
  - Mémoire partagée : pas de communications

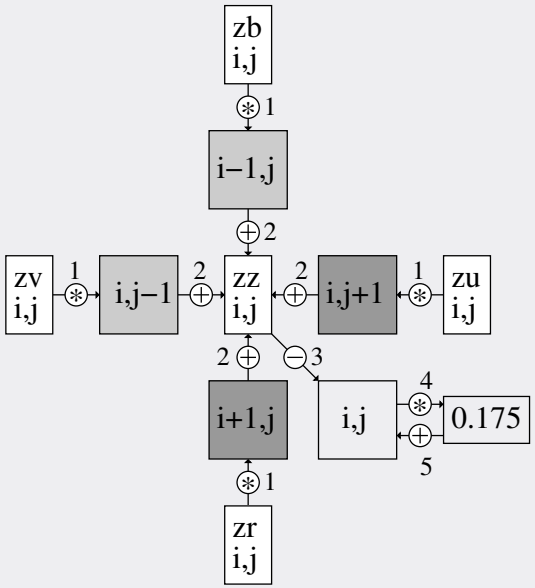
Plan

Contribution

- Description de l'algorithme utilisé
- Distribution des données
- Agencement des données
- Enchaînement des vagues

Algorithme utilisé

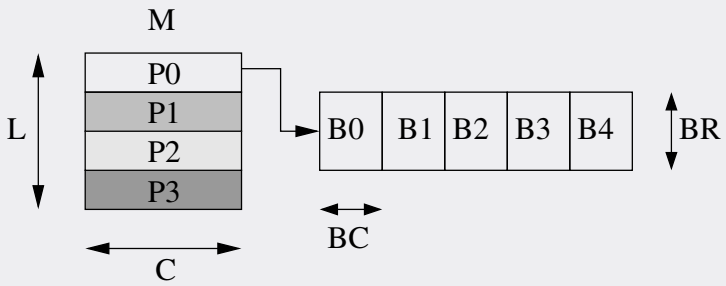
Livermore Kernel 23 - LinPack

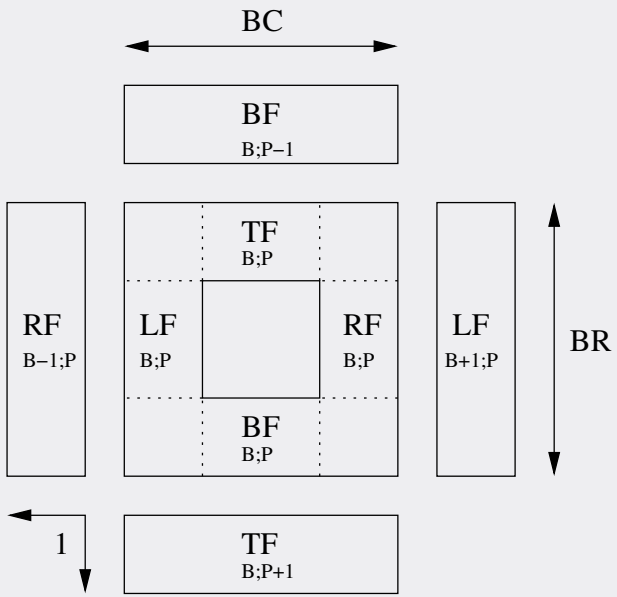


- Dépendance de données dans quatres directions
- 1 matrice de données et 5 matrices de coefficients

Distribution des données

Répartition par processeur

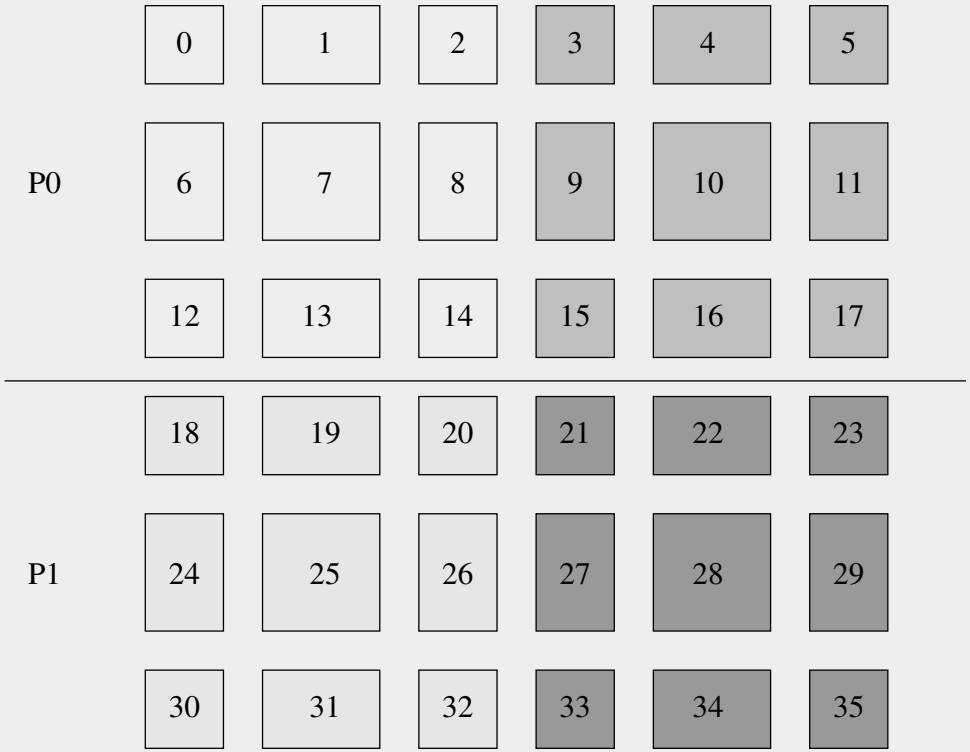




- Répartition par lignes sur les processeurs
- Découpage en blocs pour utiliser un pipeline
- 4 frontières par bloc

Représentation des données

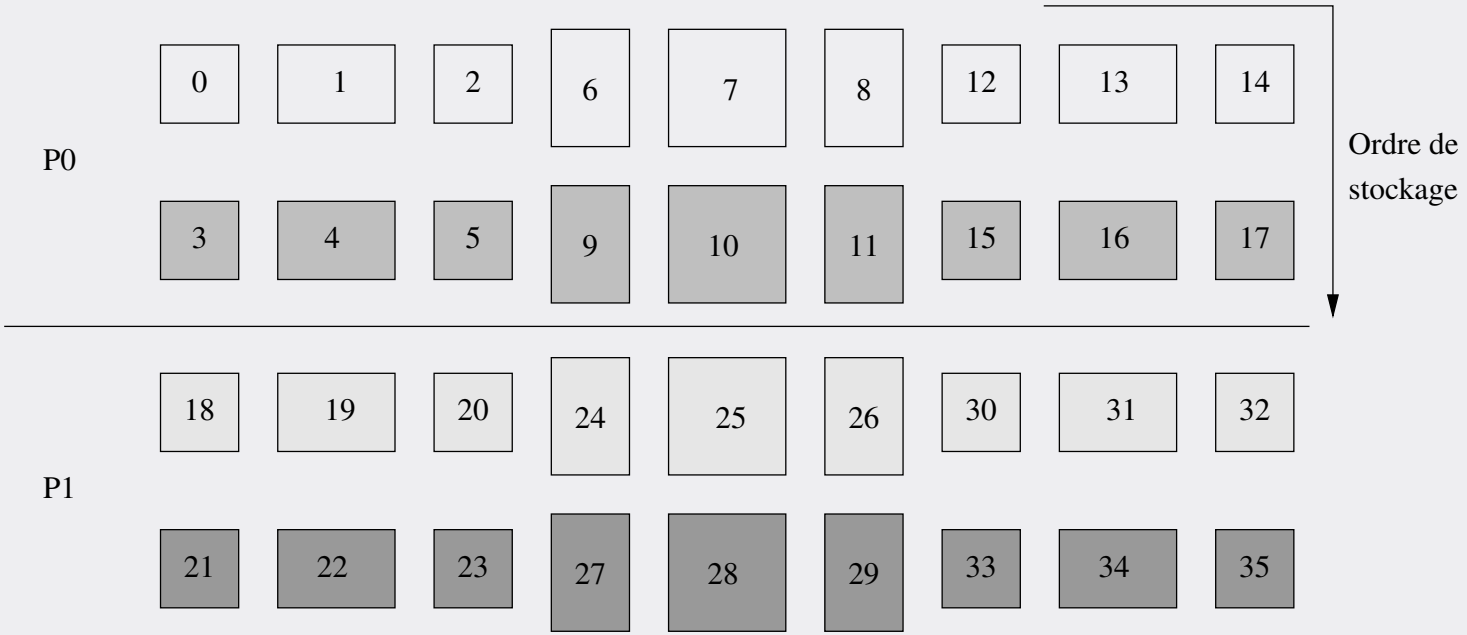
Agencement par lignes



- E/S de blocs et de frontières inefficaces

Représentation des données

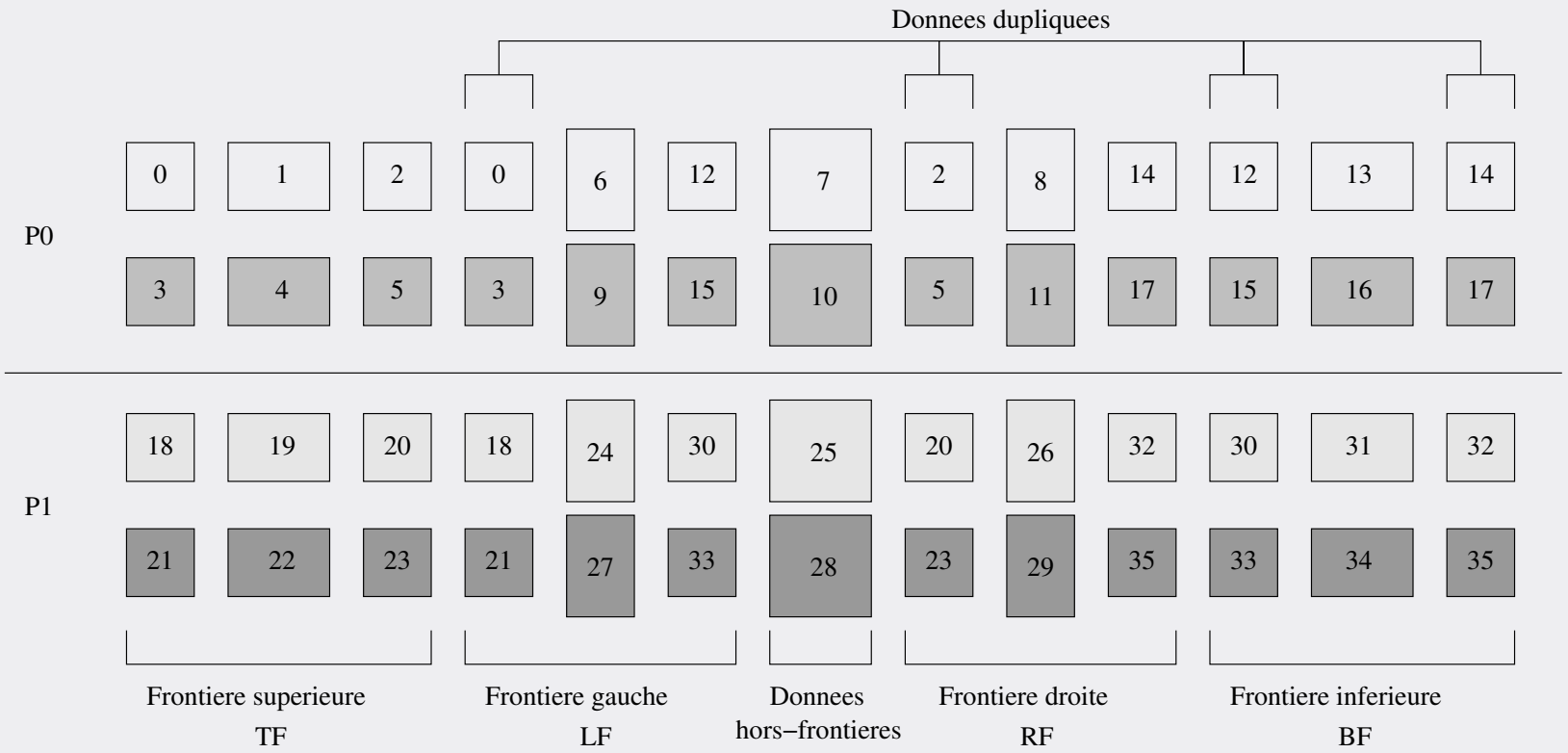
Agencement par blocs



- E/S de blocs et des frontières horizontales efficaces
- E/S des frontières verticales inefficaces
- Utilisée pour les matrices de coefficients

Représentation des données

Agencement optimisée



- E/S de blocs et de frontières efficaces
- Application d'une phase de pré-traitement plus lourde

Coût de l'agencement optimisé

Coût en espace disque

- Consommation mémoire supplémentaire :

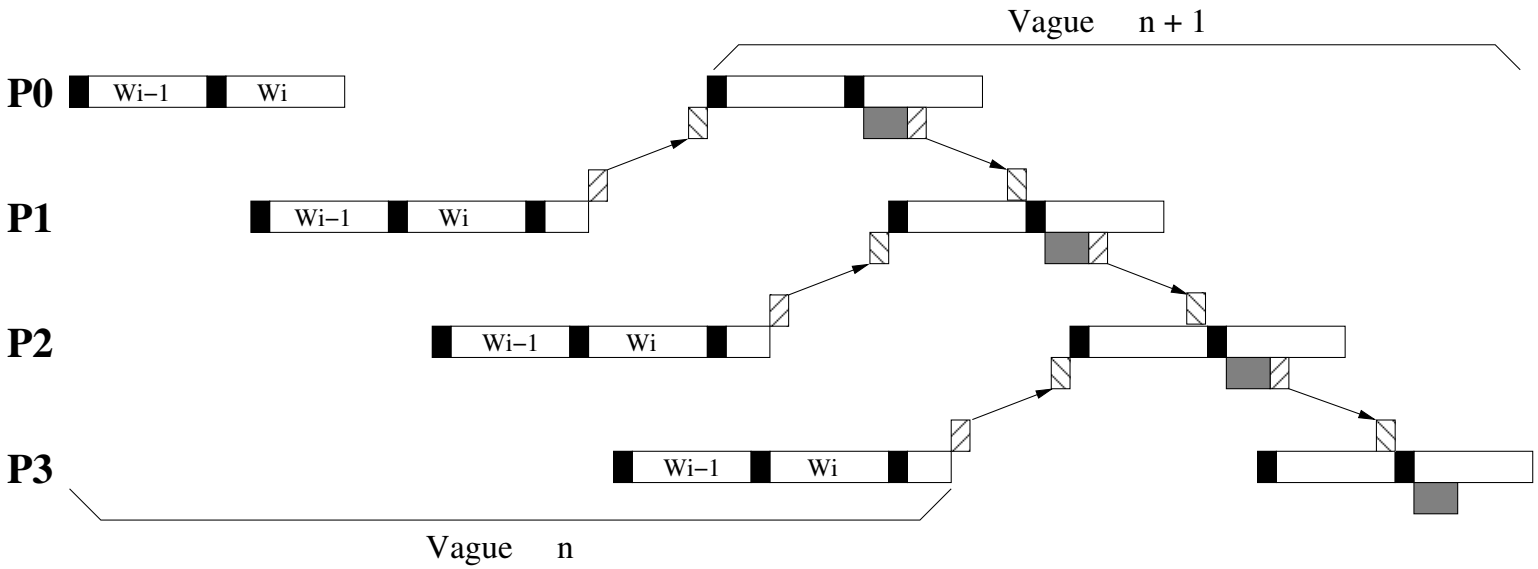
$$4 * T^2 * sizeof(element) * \frac{L}{BR} * \frac{C}{BC}$$

pour une épaisseur de frontière T

- Overhead pour une matrice de 16384x16384 doubles.

Taille d'un bloc	**Overhead**	
4096x4096	512 octets	2.38e-05 %
512x512	32 Ko	0.0015 %
64x64	2 Mo	0.098 %
8x8	128 Mo	6.25 %

Synchronisation en mémoire distribuée

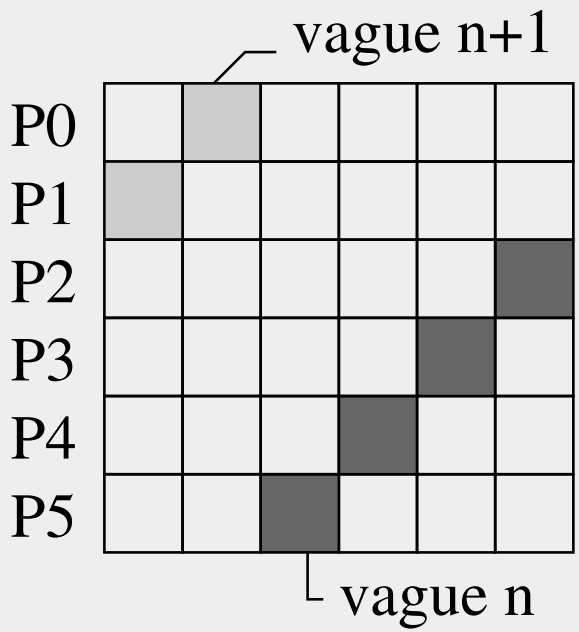


Phase de flush

- Enchaînement des itérations avec synchronisation
- Perte de temps pendant la synchronisation

Relxation de la synchronisation en mémoire partagée

Enchaînement des vagues



- Pas de communications coûteuses
- Régime constant sans phase de flush

Plan

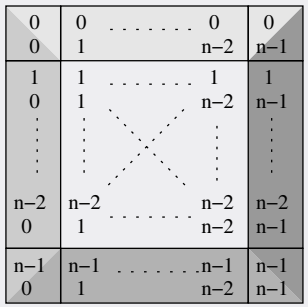
Implantation

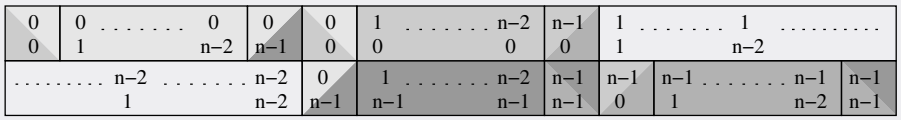
- Outils, gestion et transferts de données
- Résultats sur l'utilisation de l'agencement optimisé
- Résultats sur l'application du Livermore Kernel 23

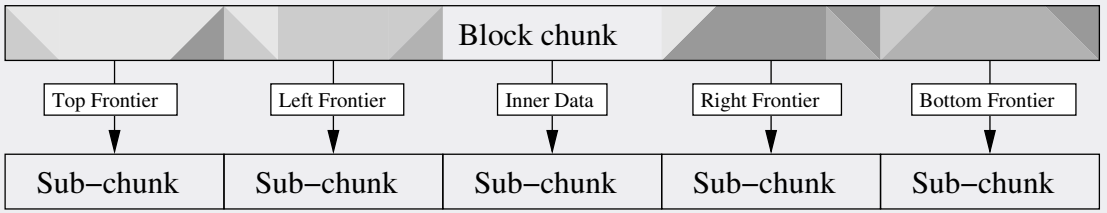
Outils

ParXXL

- C++, développée conjointement au LORIA et à Supélec
- Possibilité d'écrire pour des architectures distribuées et à mémoire partagée avec la même API
- Chunks
  - Zone de données permettant le mappage de tout ou partie d'un fichier
  - Extraction d'un sous-chunk contigu dans un chunk

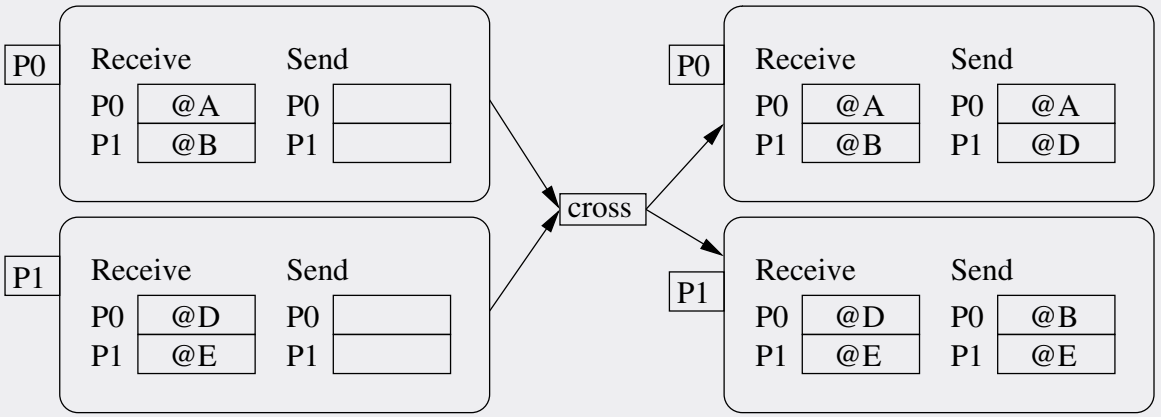






Transferts de données en mémoire partagée

Cross pointer

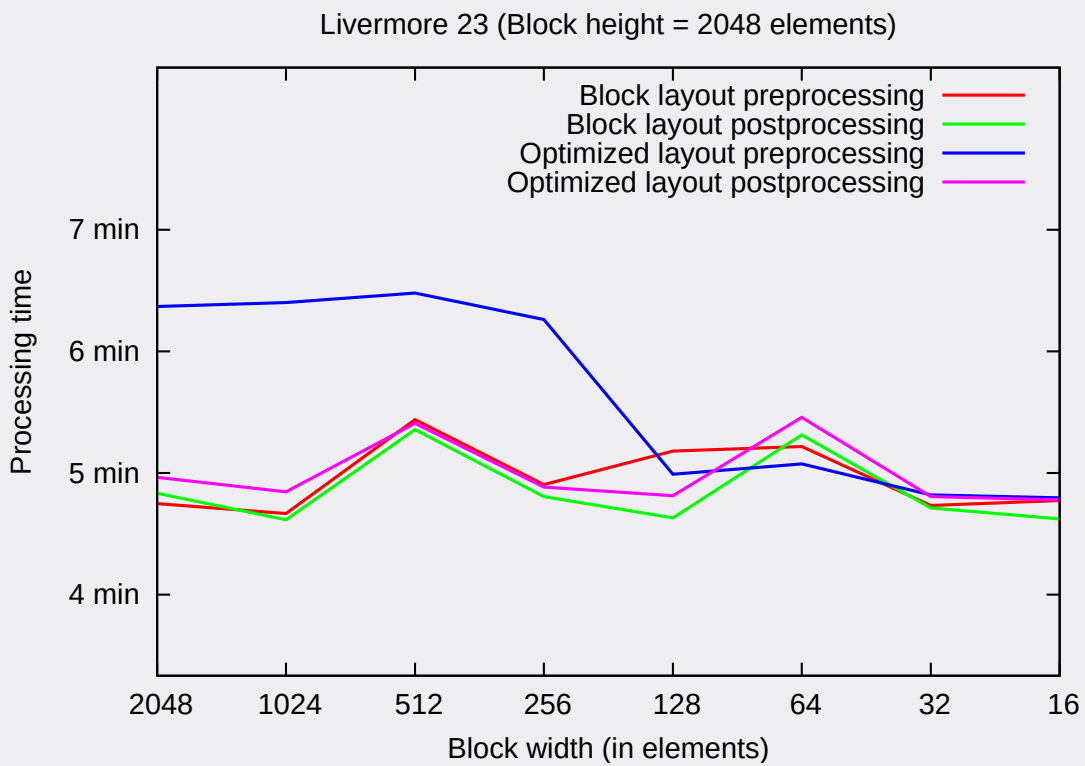


- Synchronisation forte uniquement à l'initialisation
- Communication par lecture/écriture dans/depuis un chunk
- Synchronisation locale par rendez-vous

Utilisation de l'agencement optimisé

Pré-traitement et post-traitement

- Matrices de 16384x16384 doubles

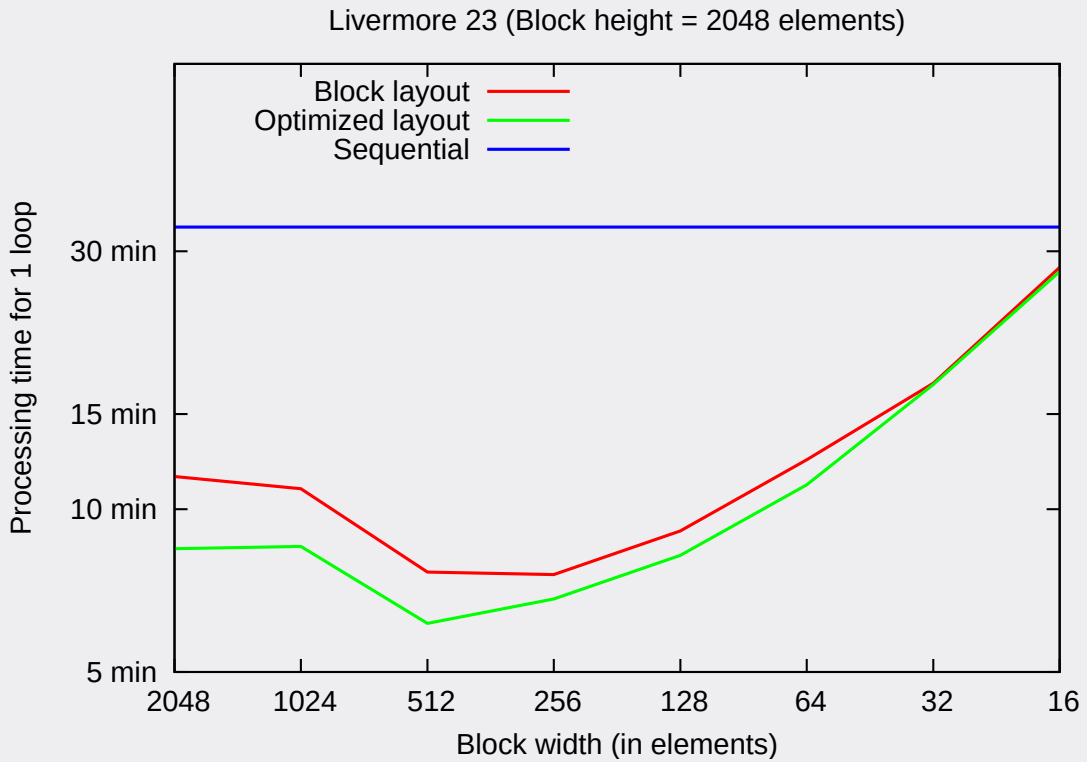


- Traitements sensiblements identiques pour les deux arrangements

Calculs

Application du Livermore Kernel 23

- Matrices de 16384x16384 doubles



- Existence d'une taille de bloc la plus efficace

Conclusion

Agencement des données

- Surcoûts en taille et en temps largement compensés
- Plus de pénalisation lors des entrées/sorties

Execution en mémoire partagée

- Enchaînement des vagues
- Régime constant jusqu'à la fin de la dernière itération

Perspectives

Agencement des données

- Prefetching agressif pour des traitements indépendants de la taille des blocs
- Répartition des données sur le(s) disque(s) dur(s)
- Généraliser l'agencement aux dimensions > 2

Modèle

- Modéliser l'algorithme pour calculer la taille de bloc optimale
- Comparer les performances avec une implantation sur architecture distribuée