DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr

PMem : Placement MEMoire

Répartir ses allocations sur une machine à plusieurs mémoires

Laboratoire Exascale - UVSQ & CEA | Rémi BARAT - Patrick CARRIBAULT, William JALBY

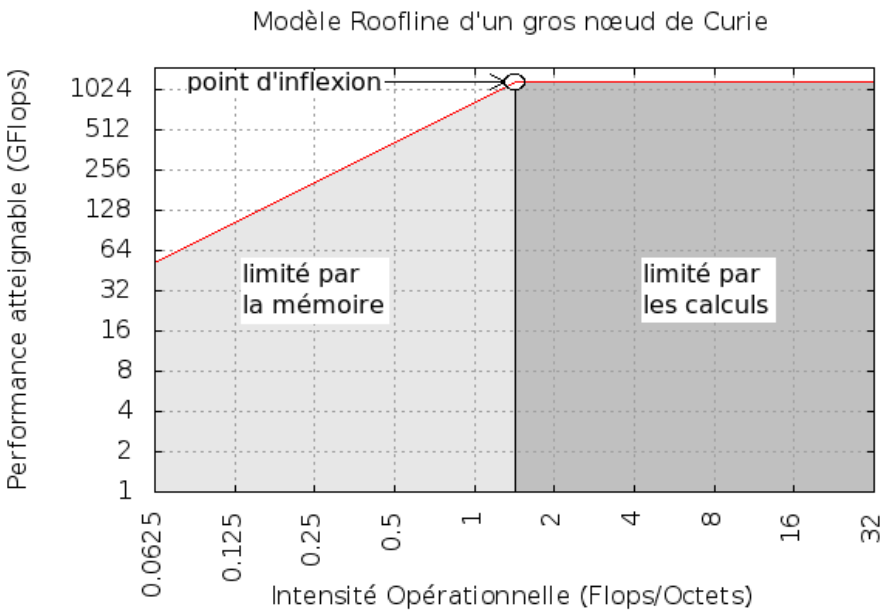
11 Décembre 2014

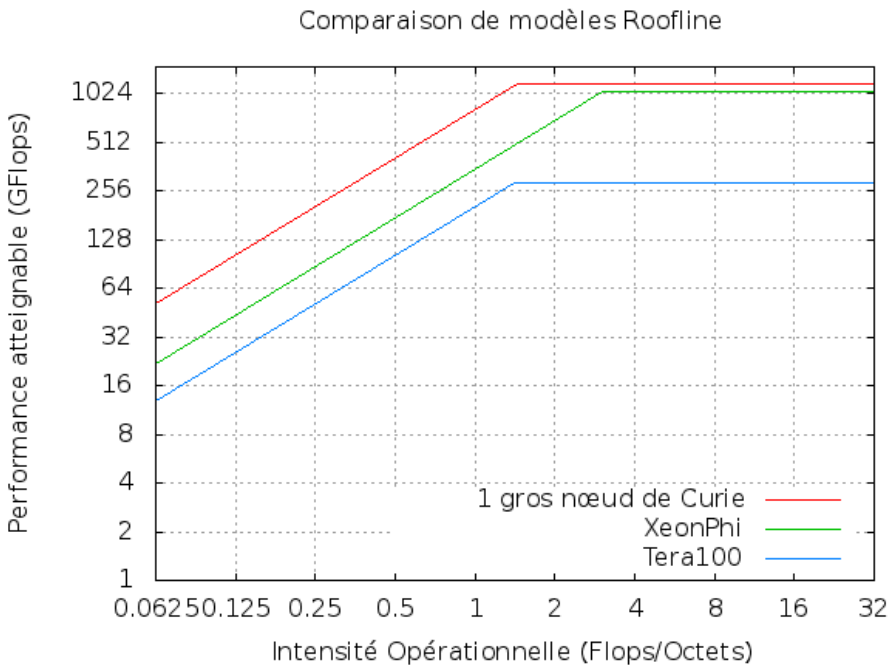
Exascale computing research

Plan

Plan

- Performance d'une application de plus en plus limitée par ses *accès mémoire*
- Modèle Roofline [1]
  - Prédiction de la performance maximale atteignable par une application
  - Intensité opérationnelle : $\frac{flop}{nb\ acces\ en\ memoire\ centrale}$



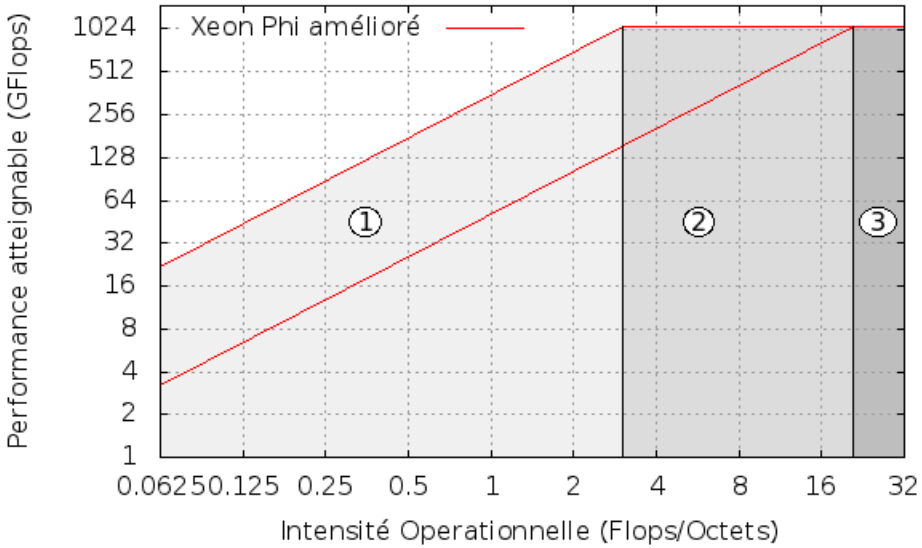


1. *Roofline : An Insightful and Visual Performance Model for Mutlicore Architectures*, S. Williams et al.

Nouvelle architecture : machine à plusieurs mémoires

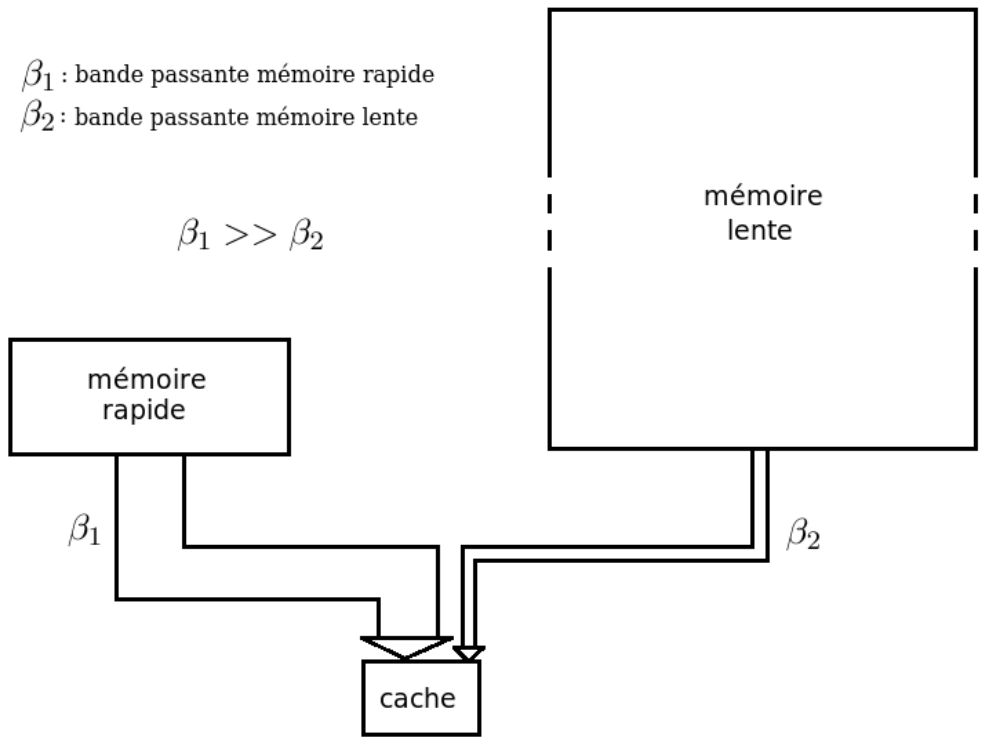
- Niveaux de cache : diminue le temps d'accès
- On peut aussi augmenter la bande-passante :
  Principe avancé par l'Intel KNL : plusieurs mémoires disponibles
  - Pas de duplication entre les mémoires
  - Choix laissé à l'utilisateur

Modèle Roofline d'un nœud Xeon Phi auquel on a rajouté une mémoire de Xeon.



β_1 : bande passante mémoire rapide
β_2 : bande passante mémoire lente

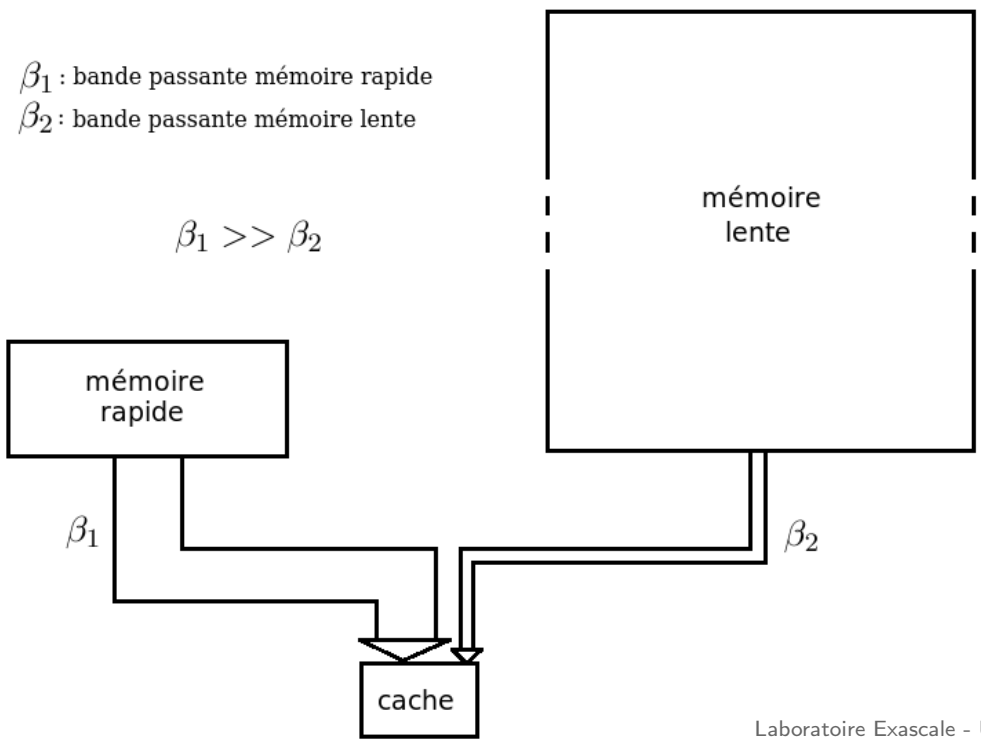
$\beta_1 >> \beta_2$



Placement des données

Caractériser les données qui bénéficieraient de la mémoire rapide

- algorithme
- implémentation
- résultats : efficacité, fiabilité



Plan

Algorithme

Principe

- Gestion des Threads OpenMP
- Allocations accédées dans les sections parallèles*
  (Par plusieurs threads en même temps)
  $\Rightarrow$ Mémoire rapide (débit important)

*section parallèle : commence par #pragma omp parallel

1re version

$\forall$ allocation de l'utilisateur : nombre d'accès en section parallèle

Améliorations

- Accès par plusieurs threads
- Notion temporelle (recouvrement par du calcul)
- Accès en mémoire (distinction cache hits/misses)

Plan

Plan

Outils et Framework existants

Frameworks

- Valgrind, Pin (Intel), DynamoRIO : instrumentation binaire, dynamique (DBA tools)
- MAQAO (UVSQ) : instrumentations statique/dynamique couplées.
- autres...

Frameworks	Avantages	Limites	Outils existants
Valgrind	Robustesse, prise en main	surcoût temporel, multi-thread séquentiel	MemCheck, Cachegrind, DRD
Pin	Linux+Win, multi-thread séquentiel	Robustesse	Intel Parallel Inspector, Intel Parallel Advisor
DynamoRIO			Dr. Memory
MAQAO	Analyse statique & dynamique, surcoût temporel		MTL

Plan

Fonctionnement de Valgrind

Un DBI (Dynamic Binary Instrumentation) framework

- Instrumentation du binaire à l'exécution
- Wrappage des malloc/new,
- Wrappage des fonctions de la libgomp (GOMP_parallel_start...)

Valgrind en parallèle...

- Code multithreadé ⇒ exécution en séquentiel en gardant les threads
- Code multiprocessus ⇒ un processus Valgrind par processus utilisateur :

Les outils Valgrind

- Développement d'un « plug-in » basés sur le cœur de Valgrind
- Existants : Memcheck, DRD, Cachegrind, Hellgrind...
- Mon outil Valgrind : PMem (Placement MEMoire)

Plan

PMem : Principe et Options

Objectif

Classement des allocations les plus accédées en région parallèle

Principe

Prise en compte des accès mémoire

- Dans une région parallèle
- Correspondant à une allocation dynamique (malloc, new)
- Zone accédée pour la 1re fois
  $\Rightarrow$ Même zone accédée par les autres threads : non comptée

Métrique fonction du nombre d'accès par différents threads dans les différentes zones d'une allocation dynamique.

- Marqueurs : instrumenter une partie du code
- Optimisations : limiter le surcoût temporel

PMem : Implémentation

Structure de donnée : Segment

- adresse, taille de la zone accédée
- nb d'accès dans cette zone
- possibilité d'extension/fusion entre Segments
- liste chaînée triée par adresse

Réduire le surcoût temporel : optimisations

Parcourir la liste à chaque accès est très coûteux

- Système de Checkpoint avec les bits de poids fort
- Système de Cache : pointeur vers les derniers Segments accédés
- Système de Buffer : insertion des accès par paquets triés

Réduction du surcoût de 10 à 100. Surcoût actuel : 100 à 1000.

Plan

Produit matriciel

```
double[][] matA, matB, matC;
double tmpA;
#pragma parallel for
for (i)
   for(k)
      tmpA = matA[i][k]
      for (j)
         matC[i][j] += tmpA * matB[k][j]
```

```
Free at      :   5c22040

=== Free of   5822040 of size 2000000
Loads     |       addr |     size | nb accesses
Th     0  |            |          |
          |    5822040 |   500000 |    62500
Th     1  |            |          |
          |    589c160 |   500000 |    62500
Th     2  |            |          |
          |    5916280 |   500000 |    62500
Th     3  |            |          |
          |    59903a0 |   500000 |    62500

=== Free of   5a0a500 of size 2000000
Loads     |       addr |     size | nb accesses
Th     0  |            |          |
          |    5a0a500 |  2000000 | 31250000
Th     1  |            |          |
          |    5a0a500 |  2000000 | 31250000
Th     2  |            |          |
          |    5a0a500 |  2000000 | 31250000
Th     3  |            |          |
          |    5a0a500 |  2000000 | 31250000
```

matA

matB

```
==================== End of Program ==============================

===> Most accessed allocations in parallel regions <===

======> Load <===
      addr |     size |  Alloc  | nb acc tot | nb acc th0 | nb acc th1 | nb acc th2 | nb acc th3 |
   5a0a500 | 2000000 |  malloc | 125,000,000 | 31,250,000 | 31,250,000 | 31,250,000 | 31,250,000 |
 allocated at 0x60000000C: ???
           at 0x4C2BA97: malloc (vg_replace_malloc.c:296)
           at 0x4007A4: main (3.3.prodMat_opt.c:26)

   5c22040 | 2000000 |  malloc | 125,000,000 | 31,250,000 | 31,250,000 | 31,250,000 | 31,250,000 |
 allocated at 0x600000010: ???
           at 0x4C2BA97: malloc (vg_replace_malloc.c:296)
           at 0x4007B4: main (3.3.prodMat_opt.c:27)

   5822040 | 2000000 |  malloc |    250,000 |     62,500 |     62,500 |     62,500 |     62,500 |
 allocated at 0x600000008: ???
           at 0x4C2BA97: malloc (vg_replace_malloc.c:296)
           at 0x400797: main (3.3.prodMat_opt.c:25)
```

matB

matC

matA

Coral Benchmark : LULESH

Description

- Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics
- Code d'hydrodynamique : boucles en temps et en espace
- Nombreux tableaux de petite taille (1000 octets), alloués/libérés régulièrement au début/à la fin de chaque itération.

Résultats

lulesh -p -s 5 (length of mesh along side 5^3)
itérations : 72, seule la 36^e est instrumentée
nb de régions parallèles relevées : 35207

- Surcoût temporel : $\times 1100$
- Allocation la plus accédée en parallèle :
  en lecture variable vnew « nouveau volume »)
  en écriture variable e_new « nouvelle énergie »)
  variable p_new « nouvelle pression »)

Résultats LULESH (1)

Les 5 données les plus accédées en parallèle en lecture

```
==================== End of Program ================================

===> Most accessed allocations in parallel regions <===

======> Load <===
      addr |    size |  Alloc  | nb acc tot | nb acc th0 | nb acc th1 | nb acc th2 | nb acc th3 |
  102ec850 |    1000 |  malloc |      6,498 |      1,666 |      1,839 |      1,839 |      1,154 |
 allocated at 0xB000043BC: ???
           at 0x4C2BA97: malloc (vg_replace_malloc.c:296)
           at 0x4099CD: main (lulesh.cc:2609)

  10352cd0 |     200 |  malloc |      4,210 |      1,034 |      1,274 |      1,234 |        668 |
 allocated at 0xB00004410: ???
           at 0x4C2BA97: malloc (vg_replace_malloc.c:296)
           at 0x40AD3C: main (lulesh.cc:2388)

  10353110 |     200 |  malloc |      3,320 |        900 |        980 |        940 |        500 |
 allocated at 0xB00004420: ???
           at 0x4C2BA97: malloc (vg_replace_malloc.c:296)
           at 0x40AD7C: main (lulesh.cc:2392)

  10352ef0 |     200 |  malloc |      2,785 |        607 |        847 |        847 |        484 |
 allocated at 0xB00004418: ???
           at 0x4C2BA97: malloc (vg_replace_malloc.c:296)
           at 0x40AD5C: main (lulesh.cc:2390)

  102ce690 |    1792 |  malloc |      2,544 |         23 |        507 |      1,008 |      1,006 |
 allocated at 0xC00004340: ???
           at 0x4C2BA97: malloc (vg_replace_malloc.c:296)
           at 0x58FB1D8: ??? (in /usr/lib/x86_64-linux-gnu/libgomp.so.1.0.0)
           at 0x58FEF84: ??? (in /usr/lib/x86_64-linux-gnu/libgomp.so.1.0.0)
           at 0x58FDFDB: GOMP_parallel_start (in /usr/lib/x86_64-linux-gnu/libgomp.so.1.0.0)
           at 0x4C2BE4C: GOMP_parallel_start (pm_gomp_intercepts.c:191)
           at 0x4096AE: main (lulesh.cc:1233)
```

vnew (new relative volume -- temp)

e_new (new energy)

delvc (?)

bvc (?)

allocation liée à la libgomp

Résultats LULESH (2)

Détails concernant les 2 données les plus accédées en parallèle en lecture

```
Free at      :  102ec850

=== Free of  102ec850 of size    1000
Loads     |       addr |    size | nb accesses
Th     0  |            |         |
          |   102ec850 |     264 |    1349
          |   102eca10 |      32 |      80
          |   102eca60 |      16 |      22
          |   102ecaa0 |      40 |     100
          |   102ecae8 |      16 |      22
          |   102ecb20 |      24 |      60
          |   102ecb80 |      24 |      33
Th     1  |            |         |
          |   102ec888 |      56 |    1274
          |   102ec950 |     256 |     328
          |   102eca70 |      16 |      22
          |   102ecac8 |      32 |      80
          |   102ecaf8 |      16 |      22
          |   102ecb38 |      24 |      60
          |   102ecb98 |      24 |      33
          |   102ecbe8 |       8 |      20
Th     2  |            |         |
          |   102ec8c0 |      56 |    1274
          |   102ec998 |      64 |      88
          |   102eca50 |     280 |     304
          |   102ecbb0 |      24 |      33
          |   102ecbd0 |      16 |      40
          |   102ecbf0 |      40 |     100
Th     3  |            |         |
          |   102ec8f8 |      32 |     728
          |   102ec9d8 |      56 |      77
          |   102eca90 |      16 |      22
          |   102ecb18 |       8 |      11
          |   102ecb50 |     232 |     316
```

vnew (new relative volume -- temp)

```
=== Free of  10352cd0 of size     200
Loads     |       addr |    size | nb accesses
Th     0  |            |         |
          |   10352cd0 |      56 |    1034
Th     1  |            |         |
          |   10352d08 |      56 |    1274
Th     2  |            |         |
          |   10352d40 |      56 |    1234
Th     3  |            |         |
          |   10352d78 |      32 |     668
Stores    |       addr |    size | nb accesses
Th     0  |            |         |
          |   10352cd0 |      56 |     560
Th     1  |            |         |
          |   10352d08 |      56 |     560
Th     2  |            |         |
          |   10352d40 |      56 |     560
Th     3  |            |         |
          |   10352d78 |      32 |     320
```

e_new (new energy)

Conclusion

Objectifs

- 1re approche de la machine à plusieurs mémoires.
- Algorithme pour conseiller un placement à l'utilisateur
- Conception d'un outil

Réalisations

- Conception d'un outil Valgrind : PMem
- Mise en place de marqueurs et optimisations pour en réduire le surcoût temporel
- Test sur Lulesh, résultats difficiles à vérifier.

Perpectives

- Fonctionnalités plus avancées pour PMem : non prise en compte des accès en cache, recouvrement par du calcul.
- Utiliser un framework plus efficace en parallèle (Pin, MAQAO...)
- Vérification des résultats en pratique