Motivations pour développer des approches de programmation parallèle de haut niveau

Sylvain Jubertie
sylvain.jubertie@univ-orleans.fr

LIFO

2 décembre 2014

1 Motivations

2 Développement et métriques

3 OSL : Orléans Skeleton Library

4 Conclusions/Perspectives

1 Motivations

2 Développement et métriques

3 OSL : Orléans Skeleton Library

4 Conclusions/Perspectives

- Pourquoi programmer pour des architectures parallèles ?

Architectures parallèles

- solution pour augmenter les performances
- smartphones → super-calculateurs
- unités vectorielles (SSE, AVX, NEON), multicore, NUMA, manycore (GPU, MIC), architectures à mémoire distribuée
- évolution : unités vectorielles plus larges (AVX512), plus de coeurs, accélérateurs
- limites : dissipation thermique, consommation

Performances théoriques

Sur un processeur 8 coeurs + AVX, calcul flottant SP :

- 1/64 des perf. théoriques si programmation séquentielle
- 1/8 des perf. théoriques si programmation vectorielle
- 1/8 des perf. théoriques si programmation threads

Différents niveaux de parallélisation à exploiter.

- Quel est l'impact de la parallélisation sur la consommation ?

Parallélisation et consommation

Projet TER Master 1 Info

- Quel est l'impact de la parallélisation sur la consommation ?
- Quel est l'impact de l'utilisation des unités vectorielles ?
- Quelle est la différence entre un programme séquentiel sur un processeur à x Ghz et un programme parallèle sur 4 coeurs à x/4 Ghz ?

Plateforme ARM

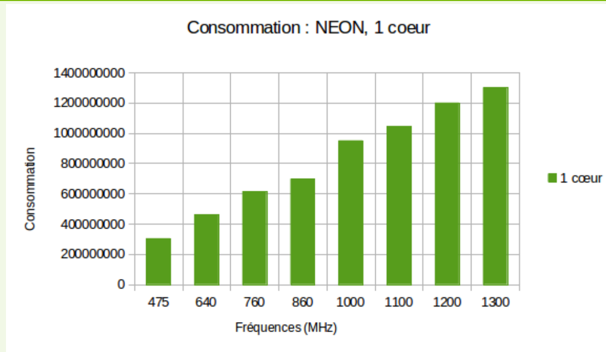
Tablette Nexus 7 :

- Tegra 3, 4 coeurs Cortex-A9 1,3Ghz
- instructions NEON

Résultats multiplication de matrice 1024^2 (en ms)

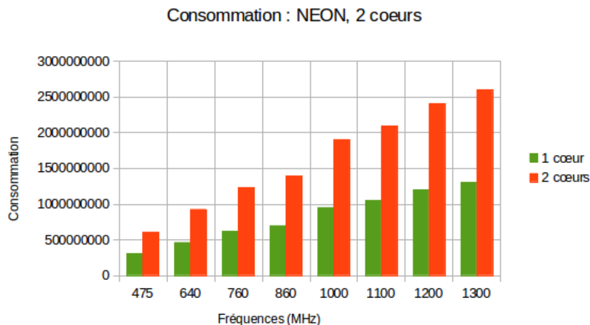
freq	seq	NEON	OMP 4c	OMP+NEON
475	23524	8571	5945	2248
640	17459	6363	4444	1665
760	14703	5356	3697	1402
860	12995	4732	3265	1261
1000	11176	4069	2815	1064
1100	10157	3702	2565	968
1200	9312	3393	2341	889

Consommation en fonction de la fréquence



Consommation non proportionnelle à la fréquence.

Consommation en fonction du nombre de coeurs



Consommation proportionnelle au nombre de coeurs.

Remarques

- Impossible d'utiliser tous les coeurs à la fréquence max.
  $\rightarrow$ surchauffe $\rightarrow$ extinction
- Pas de surconsommation des unités vectorielles...
- Meilleure conso. en baissant la fréquence et utilisant les 4 coeurs (tension+fréq.)
  jusqu'à -25% !!! mais juste pour le processeur...
- $P = k \times F \times U^2$!!!
- tension baissée par paliers

1 Motivations

2 Développement et métriques

3 OSL : Orléans Skeleton Library

4 Conclusions/Perspectives

- Quel est le rapport entre l'effort de développement d'une version parallèle et d'une version séquentielle d'un même programme ?
- Quel est le rapport entre performance de développement et performances ?

Programmation bas niveau

- unités vectorielles : compilateurs, OpenMP 4 ou Cilk, intrinsics, assembleur
- multicore : pthread, OpenMP ou Cilk
- NUMA : libnuma
- manycore : OpenCL, CUDA
- mémoire distribuée : implentations MPI

Niveaux de gris, version scalaire

```
1  for(j = 0 ; j < height ; ++j) {
2    for(i = 0 ; i < width2*4 ; i+=4) {
3
4      p = j * width * 4 +i;
5      c = (data[p]*307 + data[p+1]*604 + data[p+2]*113) >> 10;
6
7      data[p/4] = c;
8
9    }
10 }
```

Niveaux de gris, version SSE

```
1  for(j = 0 ; j < height ; ++j) {
2    for(i = 0 ; i < width2*4 ; i+=64) {
3
4      rgba0 = _mm_load_si128((__m128i*)(&data[j * width2*4 + i]));
5      rgba1 = _mm_load_si128((__m128i*)(&data[j * width2*4 + i + 16]));
6      rgba2 = _mm_load_si128((__m128i*)(&data[j * width2*4 + i + 32]));
7      rgba3 = _mm_load_si128((__m128i*)(&data[j * width2*4 + i + 48]));
8
9      // r * 307
10     r0 = _mm_mullo_epi32(_mm_and_si128(rgba0, cmask), v307);
11     r1 = _mm_mullo_epi32(_mm_and_si128(rgba1, cmask), v307);
12     r2 = _mm_mullo_epi32(_mm_and_si128(rgba2, cmask), v307);
13     r3 = _mm_mullo_epi32(_mm_and_si128(rgba3, cmask), v307);
14
15     // g * 607
16     g0 = _mm_mullo_epi32(_mm_and_si128(_mm_srli_si128(rgba0, 1), cmask), v604);
17     g1 = _mm_mullo_epi32(_mm_and_si128(_mm_srli_si128(rgba1, 1), cmask), v604);
18     g2 = _mm_mullo_epi32(_mm_and_si128(_mm_srli_si128(rgba2, 1), cmask), v604);
19     g3 = _mm_mullo_epi32(_mm_and_si128(_mm_srli_si128(rgba3, 1), cmask), v604);
20
21     // b * 113
22     b0 = _mm_mullo_epi32(_mm_and_si128(_mm_srli_si128(rgba0, 2), cmask), v113);
23     b1 = _mm_mullo_epi32(_mm_and_si128(_mm_srli_si128(rgba1, 2), cmask), v113);
24     b2 = _mm_mullo_epi32(_mm_and_si128(_mm_srli_si128(rgba2, 2), cmask), v113);
25     b3 = _mm_mullo_epi32(_mm_and_si128(_mm_srli_si128(rgba3, 2), cmask), v113);
26     ...
```

Niveaux de gris, version SSE

```
1         ...
2         // (r * 307) + (g * 607) + (b * 113)/1024
3         t0 = _mm_srli_epi32(_mm_add_epi32(r0, _mm_add_epi32(g0, b0)), 10);
4         t1 = _mm_srli_epi32(_mm_add_epi32(r1, _mm_add_epi32(g1, b1)), 10);
5         t2 = _mm_srli_epi32(_mm_add_epi32(r2, _mm_add_epi32(g2, b2)), 10);
6         t3 = _mm_srli_epi32(_mm_add_epi32(r3, _mm_add_epi32(g3, b3)), 10);
7
8         //
9         t0 = _mm_packus_epi16(_mm_packs_epi32(t0, t1), _mm_packs_epi32(t2, t3));
10
11        _mm_store_si128((__m128i*)(&data[j * width2 + i/4]), t0);
12
13      }
14    }
```

Bilan

- version SSE : 4x plus rapide
- mais nécessite un effort plus important... à mesurer

Métriques

Mesurer l'effort de développement en fonction de mesures sur le code :

- SLOC : Source Lines Of Code
- Halstead

J. Legaux, S. Jubertie, F. Loulergue : *Experiments in Parallel Matrix Multiplication on Multi-Core Systems*. ICA3PP 2012, september 2012, Fukuoka, Japan.

Mesures de Halstead

- N_1 nombre d'opérateurs
- N_2 nombre d'opérandes
- η_1 nombre d'opérateurs distincts
- η_2 nombre d'opérandes distinctes

Métriques de Halstead

- Vocabulaire : $\eta = \eta_1 + \eta_2$
- Longueur : $N = N_1 + N_2$
- Volume : $V = N \times \log_2 \eta$
- Difficulté : $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort : $E = D \times V$

Définitions

- opérateurs :
  - mots clés : `static`, `const`, `virtual`, `class`, `struct`,...
  - instructions : for, if, else, while, do, ...
  - opérateurs : arithmétiques, logiques, ...
  - ;
  - appels de fonctions
- opérandes : types, constantes, variables

Exemple multiplication de matrice

	η_1	η_2	η	N_1	N_2	N	V	D	E
seq	17	14	31	60	40	100	495	24,3	12k
SSE	26	24	50	408	328	736	4154	177,7	738k
blocked seq	27	30	57	232	184	416	2426	82,8	201k
blocked SSE	36	40	76	580	572	1152	7198	257,4	1852k

Remarques

- impossible de se passer de la programmation bas niveau
- structures de données spécifiques : alignement, distribution, ...
- combinaison difficile des différents niveaux de parallélisme
- langages spécifiques fournis à l'utilisateur
- ratio performance/effort faible (sauf OpenMP seul)

1 Motivations

2 Développement et métriques

3 OSL : Orléans Skeleton Library

4 Conclusions/Perspectives

OSL

Bibliothèque de squelettes algorithmiques :

- développée par Noman Javed
- C++, expression templates
- MPI
- structure de données `DArray<T>`
- modèle BSP (coût)
- aspects vérification Coq (F. Loulergue, N. Javed)

Squelettes algorithmiques

- M. Cole (1989)
- fonctions de haut niveau appliquées sur des structures de données
- composition de squelettes
- `map`, `zip`, `reduce`, `scan`, ...
- autres bibliothèques : Muesli, SkePU, SkeTo, ...

Produit scalaire

```
1 float sp = reduce( plus< float >,
2                    zip( multiplies< float >, d1, d1 ) );
```

Diffusion de chaleur 2D

```
1   DArray<double> bar(width*height, initu);
2   ...
3   bar = zip(plus<double>(),
4             bar,
5             map(bind(multiplies<double>(), ( diffuse * dt ) / (ds * ds), _1),
6                 zip(plus<double>(),
7                     map(bind(multiplies<double>(), -4, _1), bar ),
8                     zip(plus<double>(),
9                         mapIndex(rightb, shift(1,bfun,bar)),
10                        zip(plus<double>(),
11                            mapIndex(leftb, shift(-1,bfun,bar)),
12                            zip(plus<double>(),
13                                shift(-width, topb, bar),
14                                shift( width, bottomb, bar)
15                            )
16                        )
17                    )
18                )
19        )
20  );
```

Travaux réalisés

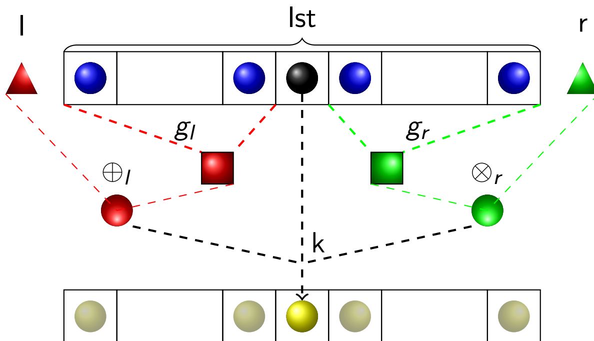
- refonte expression templates
- passage au C++11
- tableaux non équilibrés
- exceptions distribuées : `forwardExceptions`
- squelette de haut-niveau BH

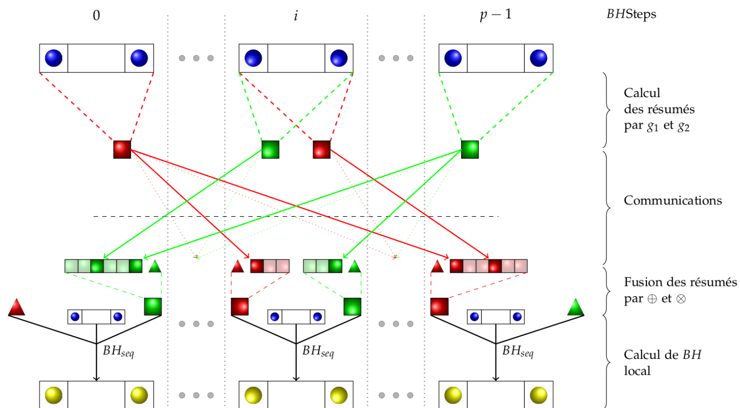
J. Legaux, F. Loulergue, S. Jubertie : *Managing Arbitrary Distributions of Arrays in Orléans Skeleton Library*. HPCS 2013, july 2013, Helsinki, Finland.
J. Legaux, F. Loulergue, S. Jubertie : *OSL : An Algorithmic Skeleton Library with Exceptions*. ICCS 2013, june 2013, Barcelona, Spain.

Squelette de haut-niveau BH

- idée : fournir un squelette de plus haut niveau
- BSP Homomorphism

J. Legaux, S. Jubertie, F. Loulergue : *Development Effort and Performance Trade-off in High-Level Parallel Programming*. HPCS 2014, july 2014, Bologna, Italy.

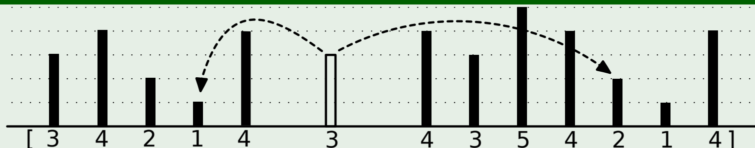

l
lst
r
g_l
g_r
$\oplus_l$
$\otimes_r$
k


0
i
$p-1$
BHSteps
Calcul des résumés par g_1 et g_2
Communications
Fusion des résumés par $\oplus$ et $\otimes$
Calcul de BH local
BH_{seq}
BH_{seq}
BH_{seq}

Problème ANSV :All Nearest Smaller Values

Etant donnée $xs = [x_1; x_2; \ldots; x_n]$ une liste d'éléments appartenant à un do- maine totalement ordonné. Pour chaque x_j , on cherche l'élément inférieur à x_j le plus proche à sa gauche et celui le plus proche à sa droite. Si un tel élément n'existe pas, nous définissons la valeur $-$ comme solution.

Exemple



ANSV BH

```
1  ANSVFunc a;
2  std::vector<int> empty(0);
3  DArray<int> in(input);
4
5  DArray<std::pair<int, int> > res =
6       osl::bh(a,
7               new candidatesL(),
8               new candidatesR(),
9               empty,
10              empty,
11              in);
```

Performance

version / np	**1**	**2**	**4**	**8**	**16**	**32**	**48**
BH	30.43 s	15.23 s	8.12 s	3.95 s	2.38 s	1.32 s	1.03 s
BSP	0.41 s	0.53 s	0.27 s	0.30 s	0.16 s	0.086 s	0.043 s
BH+BHSeq	0.62 s	0.34 s	0.15 s	0.078 s	0.039 s	0.022 s	0.017 s

Métriques

Algorithm / Metrics	η_1	η_2	η	N_1	N_2	N	V	D	E
BH									
CandidatesL	19	9	28	42	25	67	322	26,4	8500
Pickup	18	16	34	67	50	117	595	28,1	16660
								Total	25160
BH with sequential functions									
Sequential ANSV with summaries	24	17	41	107	69	176	943	48,7	45930
Sequential pickup	16	8	24	30	26	56	257	26	6682
								Total	77772
BSP									
Sequential ANSV	23	15	38	87	50	137	719	38,3	27537
Main function	35	66	101	579	416	995	6625	110,3	730737
								Total	758274

1 Motivations

2 Développement et métriques

3 OSL : Orléans Skeleton Library

4 Conclusions/Perspectives

Programmation parallèle

- programmation bas niveau : effort important, évolution limitée, code adhoc
- bibliothèques : effort réduit, langage imposé, optimisations spécifiques
- code LPC2E : utilisateurs non formés aux aspects bas-niveau

A. Spallicci, P. Ritter, S. Jubertie, S. Cordier, S. Aoudia : *Towards a self-consistent orbital evolution for EMRIs*. IX Lisa Conference, may 2012, Paris, France.

Métriques

- ordre de grandeur réaliste
- vérification par expériences sur des étudiants (DUT, L3, M1, M2)

OSL

- réduction de l'effort de développement côté utilisateur (ég. valid. SkelGIS)
- scalabilité
- métaprogrammation efficace
- impact du langage important sur effort : C++11
- mécanisme d'exceptions distribuées

Limitations OSL

- développement adhoc
- intrication entre langage, optimisations logicielles, structures de données, optimisations matérielles
- manque d'optimisations
- difficulté reportée dans la conception de la bibliothèque
- débogage difficile (templates)

Perspectives

- génie logiciel/prog. objet : découpler les aspects langages (Boost::Proto), transformations (fusion), optimisations, structures de données
- aspects distribués :
  - structures de données distribuées
  - optimisation automatique des communications
  - recouvrement automatique des calculs/comm.

Idéal

- langage de métaprogrammation ne reposant pas sur les templates C++
- analyseur du code généré pour vérifier les optimisations (fusions, move semantics)
- code vérifié : transformations, vectorisation, ...