Analyse des performances potentielles des algorithmes numériques : Un outils basé sur la mesure du parallélisme d'instructions

Eric Petit

Travail effectué à l'UPVD
en collaboration avec Bernard Goossens, Philippe Langlois, et David Parello

20 mai 2010

1 De l'incertitude de la mesure de performance

2 La mesure d'ILP

3 Étude de cas : la sommation correcte

4 Conclusion et perspective

Contexte et motivation

Motivation : constat de l'échec de la mesure de performance

- Évaluation des performances d'un algorithme d'évaluation précise de polynôme en double précision IEEE-754 (Louvet-Langlois)

		$\frac{CompHorner}{Horner}$	$\frac{DDHorner}{Horner}$	$\frac{DDHorner}{CompHorner}$
P4 - 3GHtz	GCC 4.2	3	8.9	3.0
	ICC 9.1	3.2	9.7	3.4
Athlon 64 2GHtz	GCC 4.1.2	3.2	8.7	3.0
Itanium2 1.4Ghtz	GCC 4.2	2.9	7	2.4
	ICC 9.1	1.5	5.9	3.9

- Variabilité importante des résultats selon l'architecture et le compilateur

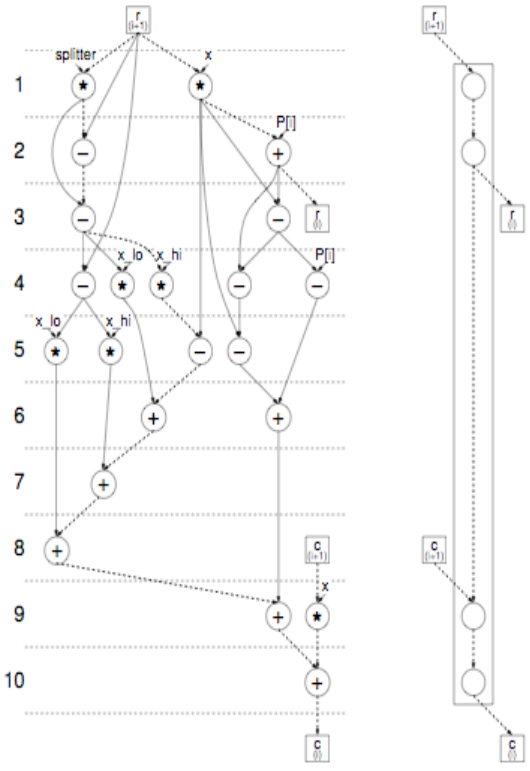
Contexte et motivation

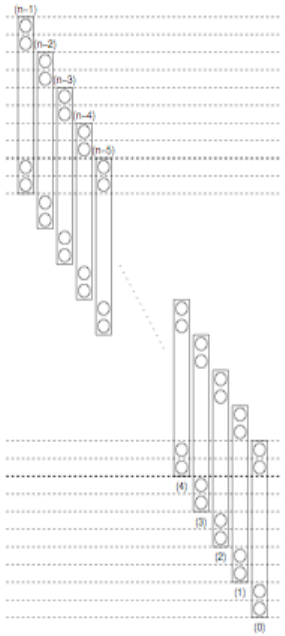
Motivation : constat de l'échec de la mesure de performance

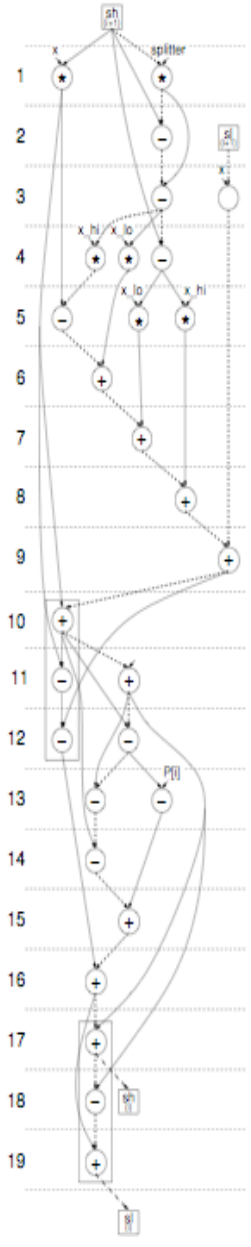
- Évaluation des performances d'un algorithme d'évaluation précise de polynôme en double précision IEEE-754 (Louvet-Langlois)

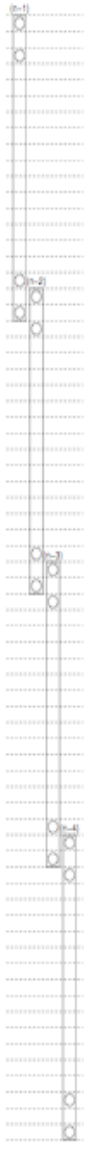
		$\frac{CompHorner}{Horner}$	$\frac{DDHorner}{Horner}$	$\frac{DDHorner}{CompHorner}$
Mesured	results	2.8-3.2	8.7-9.7	3-3.5
Flop count	Horner=2n	22n+5	28n+4	
Expected	results	11	14	1,3

- Question ouverte en 2005 (Rump)
- Élément de réponse en 2007 (Louvet)









CompHorner ILP=11

DDHorner ILP=1,65

Contexte et motivation

Constats

- Compter les opérations flottantes ne suffit pas
- Chronométrer n'explique pas les performances
- Le potentiel de parallélisme a une influence critique sur la performance
- Le programmeur "lambda" ne possède pas les ressources nécessaires pour évaluer son algorithme (outils et expertise)
- Toute la chaîne, de l'algorithme au contexte d'exécution introduit une incertitude dans la mesure

Contexte et motivation

Performance et compilation

- Nombreuses optimisations possibles
- On ne sait pas ce qui a marché ou pas
- Code généré en asm dur à lire (même pour les "initiés")
- Vision linéaire (parallélisme ?)
- Certaines optimisations trop peu souvent utilisées (e.g. SSE)

=> Compilateur/options différentes = perf. différentes
=> Algo. différents/Compilateur= perf. différentes

Contexte et motivation

Performance et architecture

- Partages des ressources entre plusieurs cœurs (e.g. cache, (I)MC)
- Variation dynamique de la fréquence du processeur
- Hiérarchie mémoires complexes (latence, taille, profondeur, prédicteurs, politique de remplacement...)
- Différents niveaux de parallélisme (ILP, vecteur, pipeline, TLP,...)
- Degré superscalaire / instructions dispo. variable (SSE, FMA,...)

=> Performances difficiles à prédire, non pérennes

Objectifs du projet PerPi

Performance relative : critère comparatif

- Aider au choix d'un algorithme
- Extraire les informations permettant la prédiction des performances
- Mesurer le parallélisme potentiel des applications
- La mesure d'ILP est une première étape

Objectif du projet PerPi

Performance absolue : optimisation

- Assister le choix des optimisations et leur mise en œuvre
- Faire ressortir les points de congestion
- Faire ressortir le parallélisme inexploité
- Constater graphiquement l'effet d'une optimisation sur le code

1 De l'incertitude de la mesure de performance

2 La mesure d'ILP

3 Étude de cas : la sommation correcte

4 Conclusion et perspective

ILP

- Représente le parallélisme potentiel entre les instructions d'un programme
- Contraint seulement par les vraies dépendances (RAW)
- Les fausses dépendances (WAR et WAW) sont des dépendances de nom, pas de données

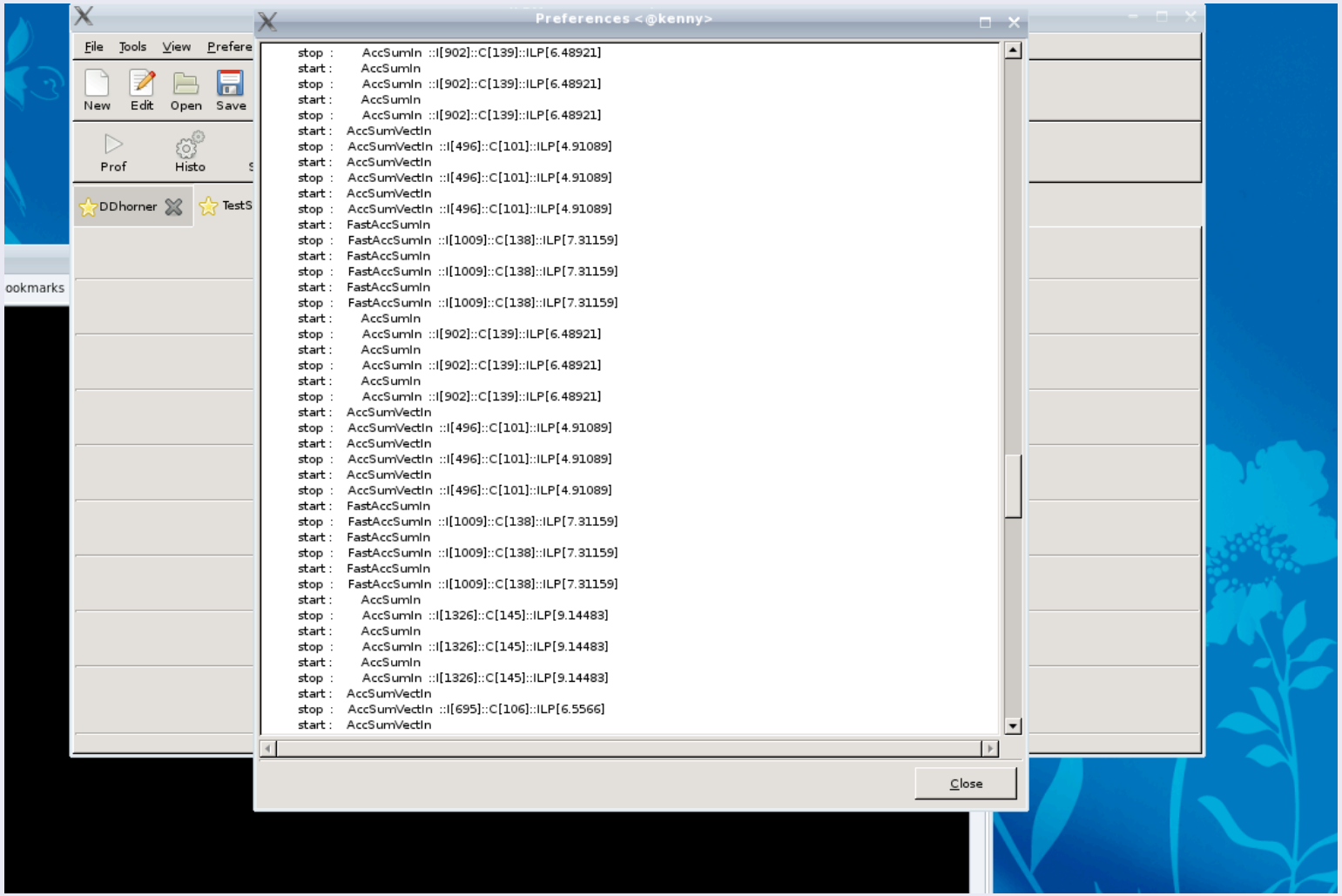
Machine idéale de Hennessy-Patterson

- Une instruction est ordonnancée le cycle après son dernier prédécesseur dans le graphe de flot de donnée
  - Latence de 1 cycle pour toutes les opérations
  - Infinité de registres
  - Infinité d'unités de calcul
  - Prédicteurs de branchements idéaux
  - Mémoire parfaite

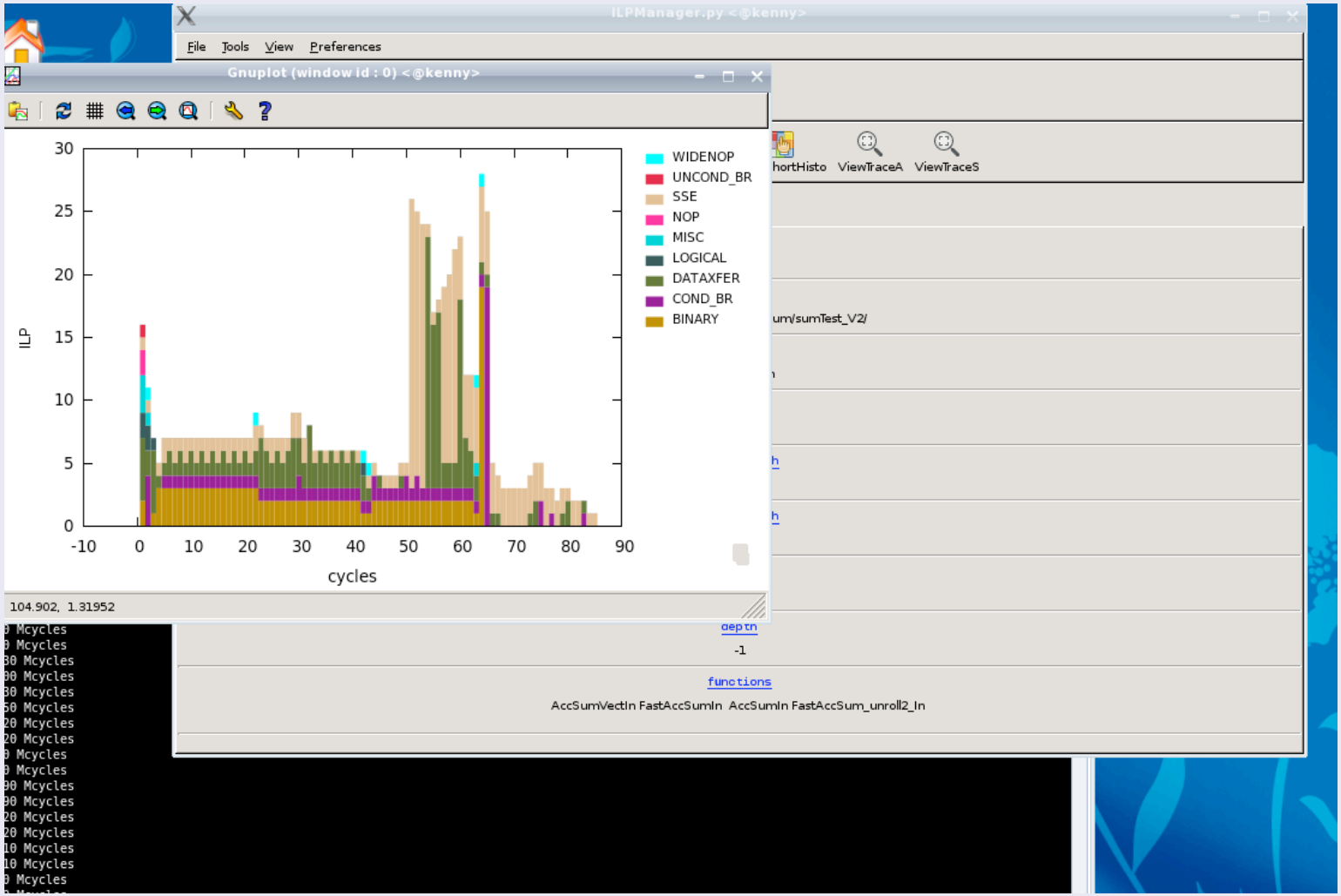
L'outil PerPi – version (très) préliminaire

- Basé sur l'outil Pin d'Intel
- Calcule l'ordonnancement idéal des instructions x86 exécutées
- Nombre de cycles, nombre d'instructions, ILP
- Beaucoup plus rapide que les solutions à base de simulateur
- Passe à l'échelle

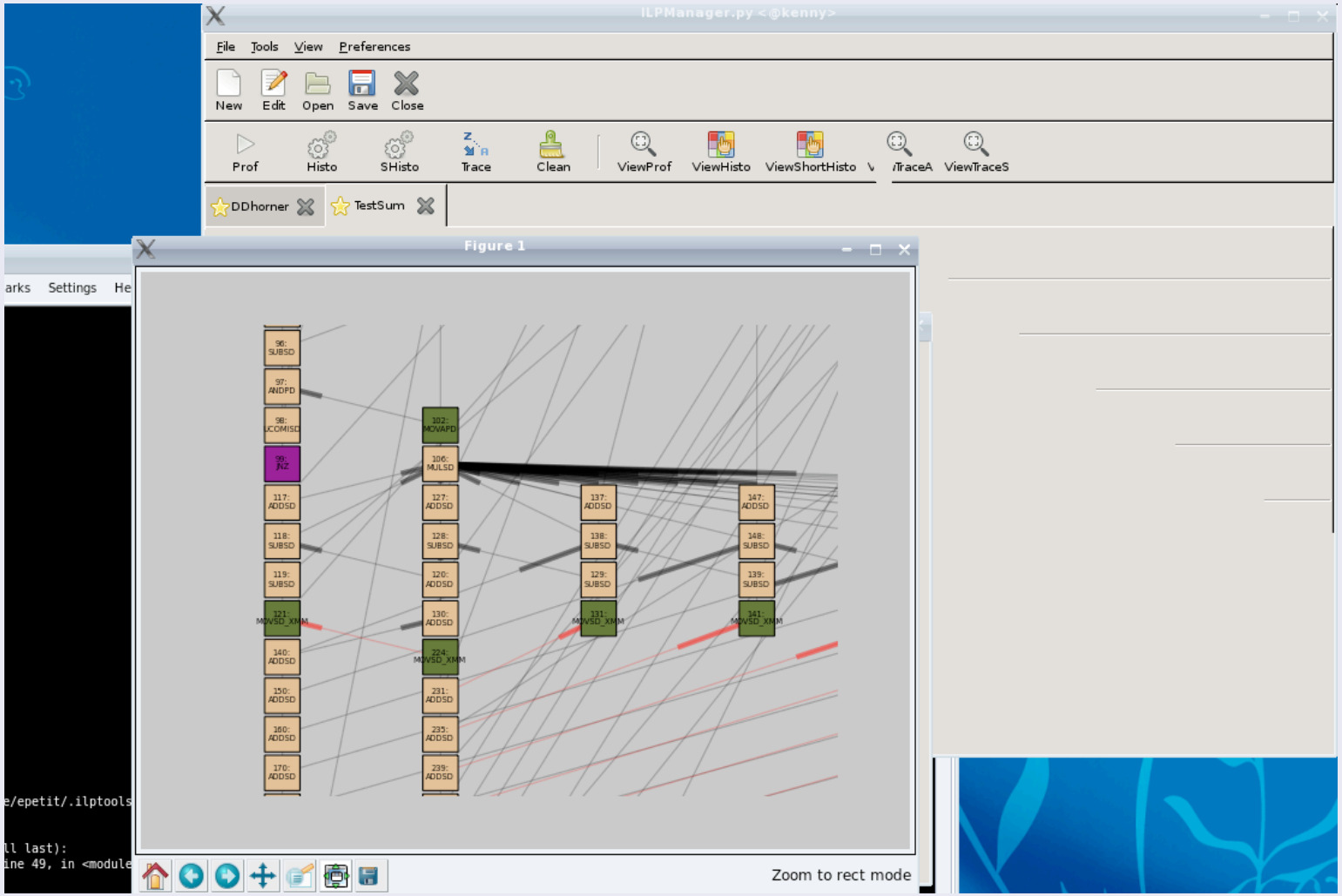
L'outil PerPi – version (très) préliminaire



L'outil PerPi – version (très) préliminaire



L'outil PerPi – version (très) préliminaire



1 De l'incertitude de la mesure de performance

2 La mesure d'ILP

3 Étude de cas : la sommation correcte

4 Conclusion et perspective

Sommation correcte : AccSum et FastAccSum

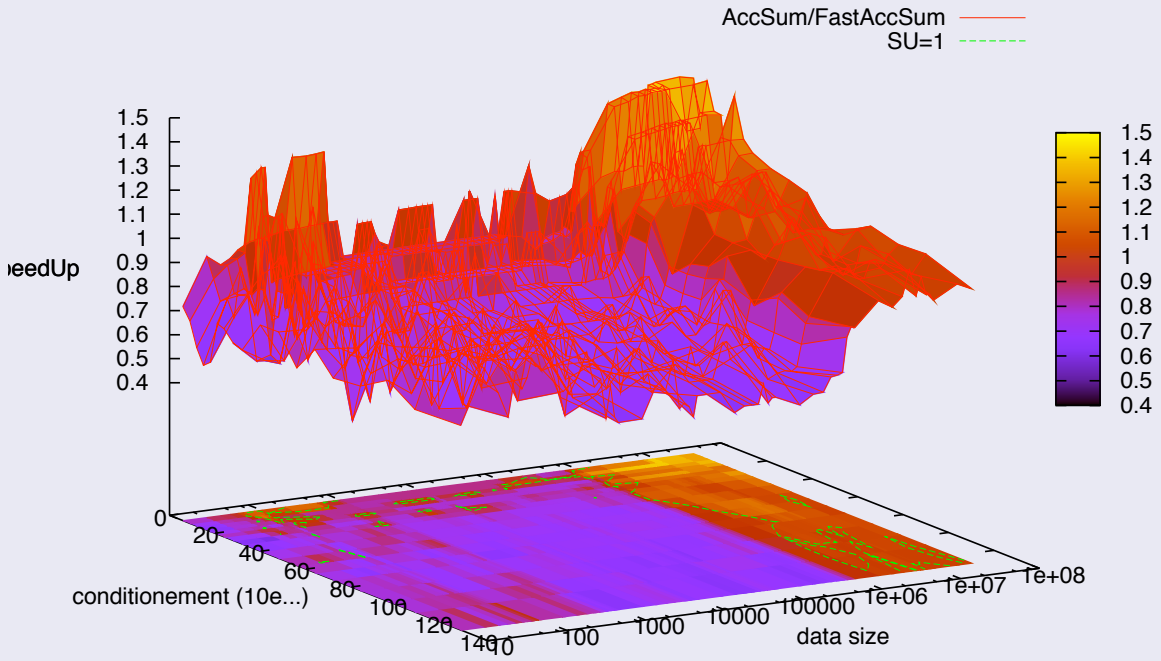
Somme correcte (accurate)

- Une somme est correcte en double précision si son résultat est la l'arrondi en double précision du résultat exact
  - Calcul itératif de l'erreur et compensation
  - Termine quand l'erreur est nulle en double précision
  - #Itération fonction du conditionnement de l'entrée
- Rump a proposé 2 solutions, AccSum et FastAccSum. Le second est "censé" être 30% plus rapide.
- Plus de calcul d'erreurs dans FastAccSum => moins d'itérations et pas la peine de calculer un max a chaque itération

Sommation correcte : FastAccSum

Premiers résultats

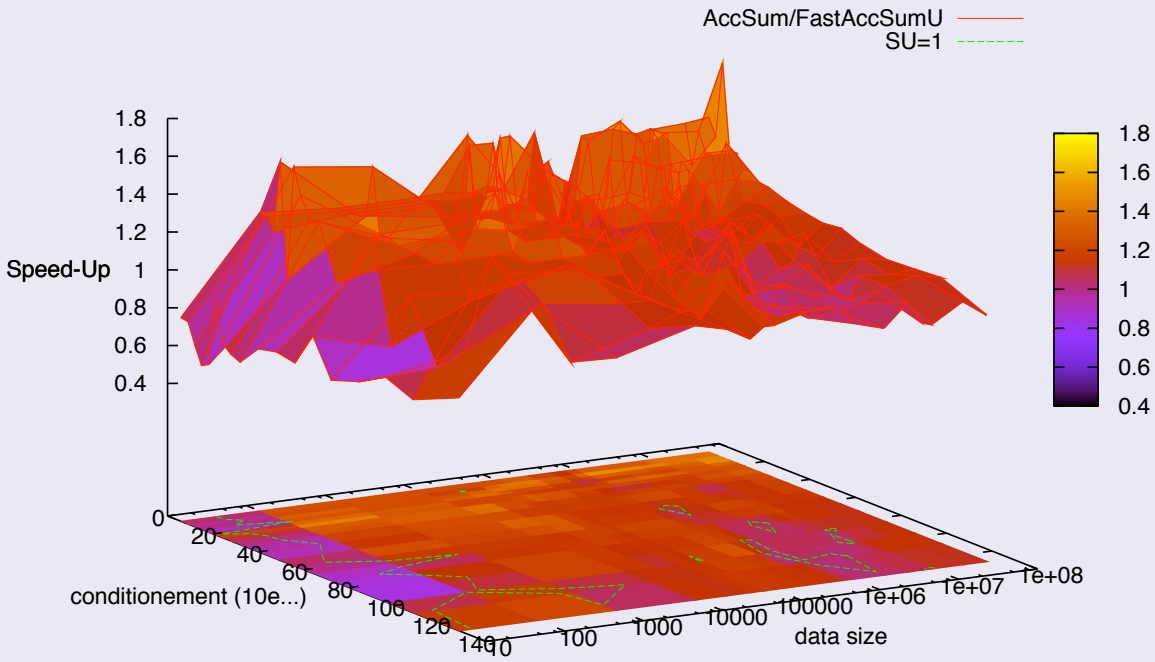
- FastAccSum souvent plus lent que AccSum ! (mesure avec tsc)



Sommation correcte : FastAccSum

Premiers résultats

- Rump propose de dérouler la boucle pour améliorer l'ILP



Sommation exacte : FastAccSum

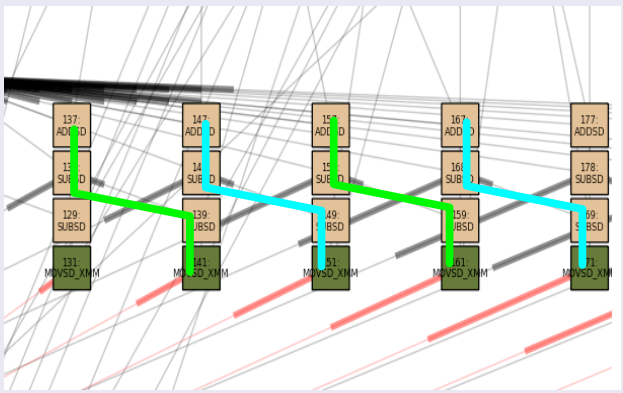
Premiers résultats

- On "voit" un SU de 50% environ, ce qui correspond à la réalité

Sommation exacte : AccSum

Cependant :

- On a optimisé FastAccSum pour exploiter le paralléllisme, pourquoi pas AccSum ?

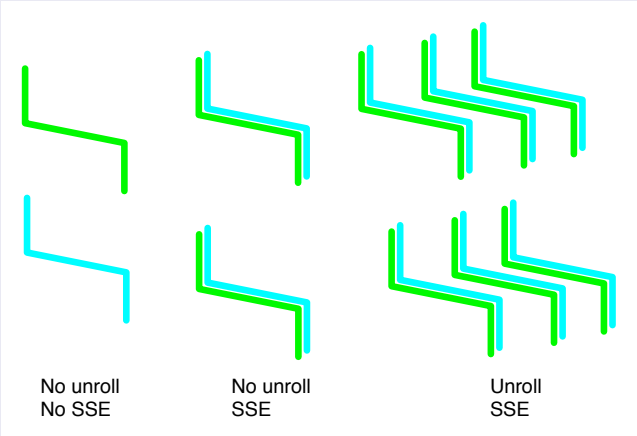


- La boucle principale est parfaitement parallélisable, on peut vectoriser et dérouler pour optimiser les flops

Sommation exacte : AccSum

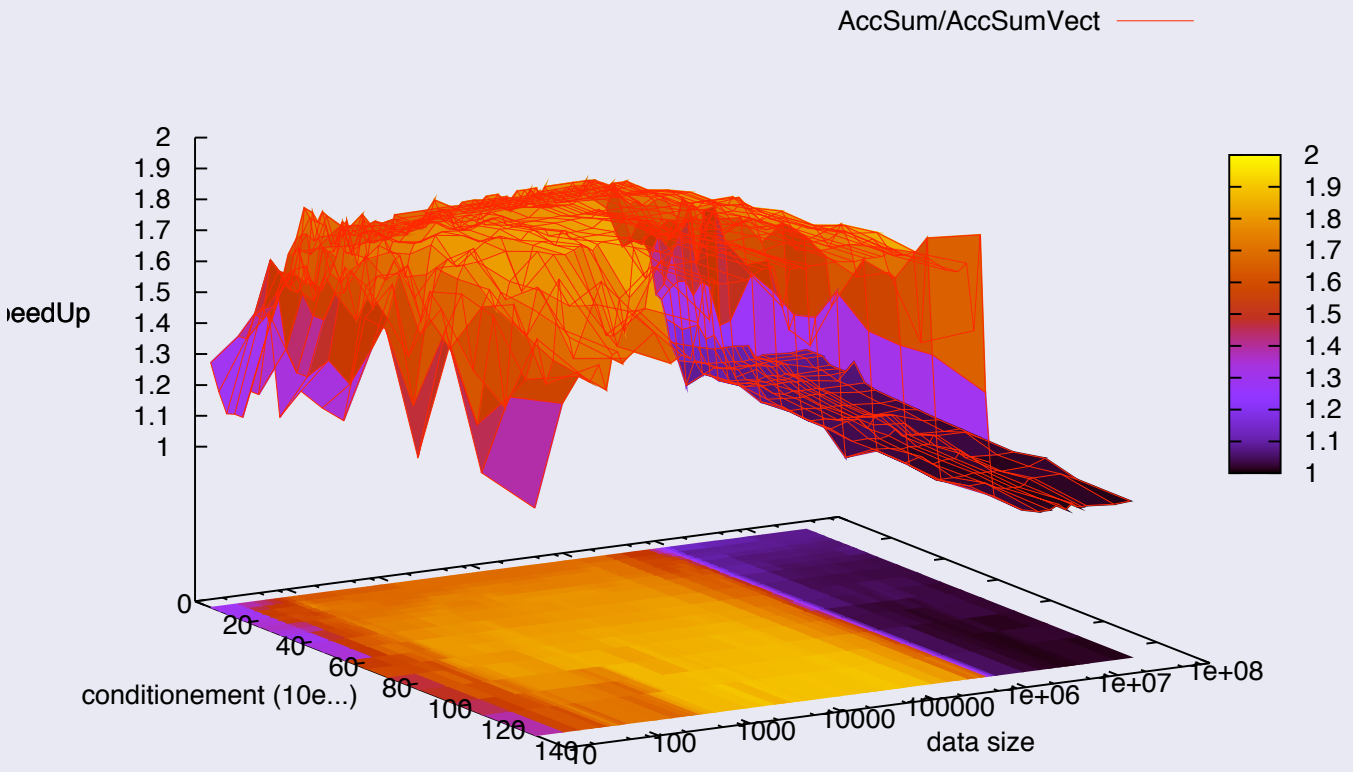
Optimisation de AccSum

- On vectorise => 2*plus d'opérations flottantes dp en 1 fois
- On déroule pour permettre de masquer au mieux les latences
- Environ 5 GFlops sur 6 possibles (mesure avec PAPI sur la somme globale, pas que la boucle optimisée)



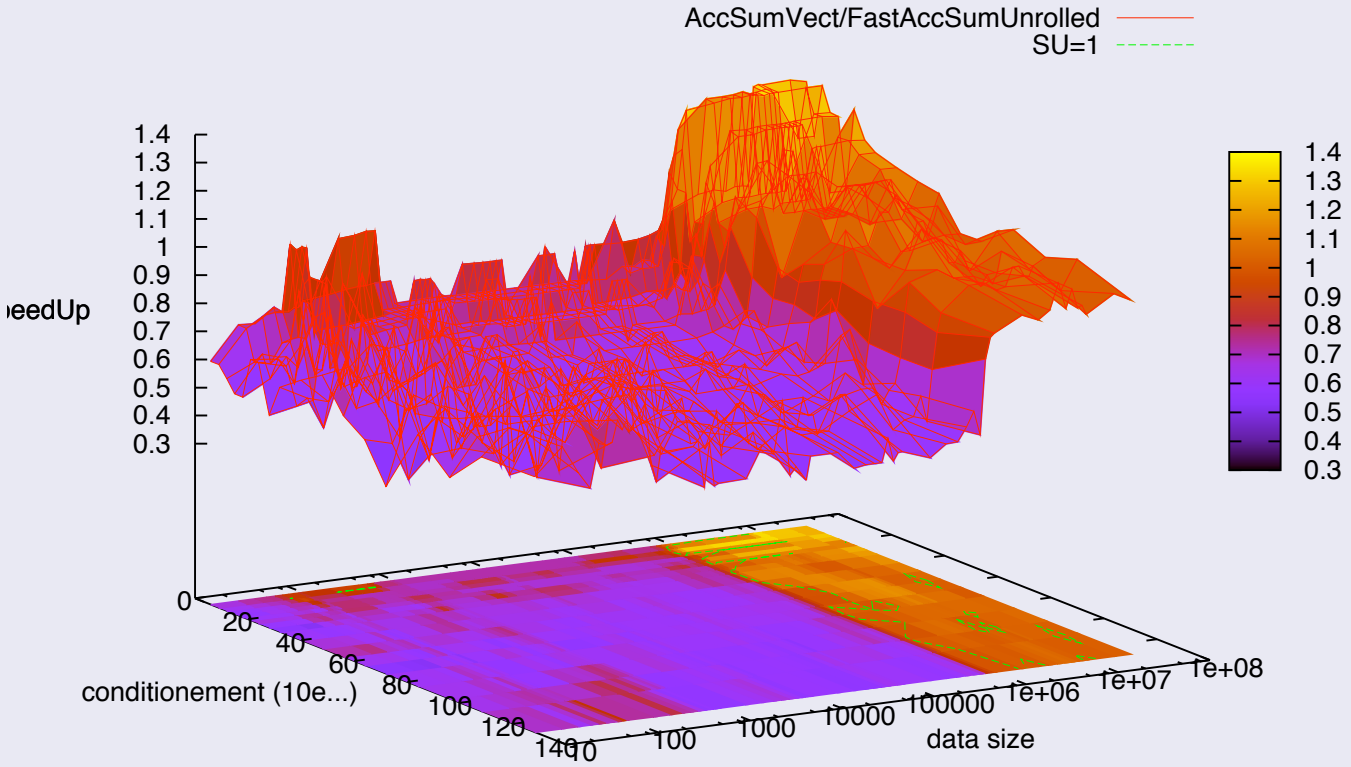
Sommation exacte : AccSum et FastAccSum

Optimisation de AccSum



Sommation exacte : AccSum et FastAccSum

Optimisation de AccSum



1 De l'incertitude de la mesure de performance

2 La mesure d'ILP

3 Étude de cas : la sommation correcte

4 Conclusion et perspective

AccSum vs FastAccSum

- Complexité supérieure, temps d'exécution inférieur
- Cette conclusion est-elle objective ?
- => nécessité de définir une affinité profil d'une application / environnement d'exécution (archi, taille des données,...)

L'ILP dans PerPi

- On est proche du résultat obtenu à la main par Louvet pour la réprésentation de l'ordonnancement des calculs
- Il est possible d'observer l'effet des optimisations et de donner un objectif de SU
- Il est possible de voir des optimisations potentielles
- On voit les chaînes de dépendances bloquantes.

La mesure de performance : perspective

- Écart-type de l'ILP ? (parallélisme uniforme ou noyaux massivement parallèles)
- Longueur et largeur des chaînes d'instructions indépendantes ? (Vecteur, TLP fin (OMP) ou large (MPI))
- Tracé sous contraintes (max ipc, max register, max op...) pour l'estimation de la performance (Intel IACA)
- Interface avec les mesures "on-site" (tsc, PAPI, TAU,...)

Merci de votre attention, avez-vous des questions ?