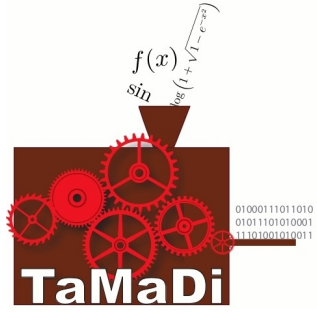
Déploiement de l'algorithme L sur GPU

Premiers résultats

Pierre Fortin, Mourad Gouicem, Stef Graillat

Équipe PEQUAN, LIP6/UPMC

Réunion ANR TaMaDi
Sofia-Antipolis, 22-23 Février 2010

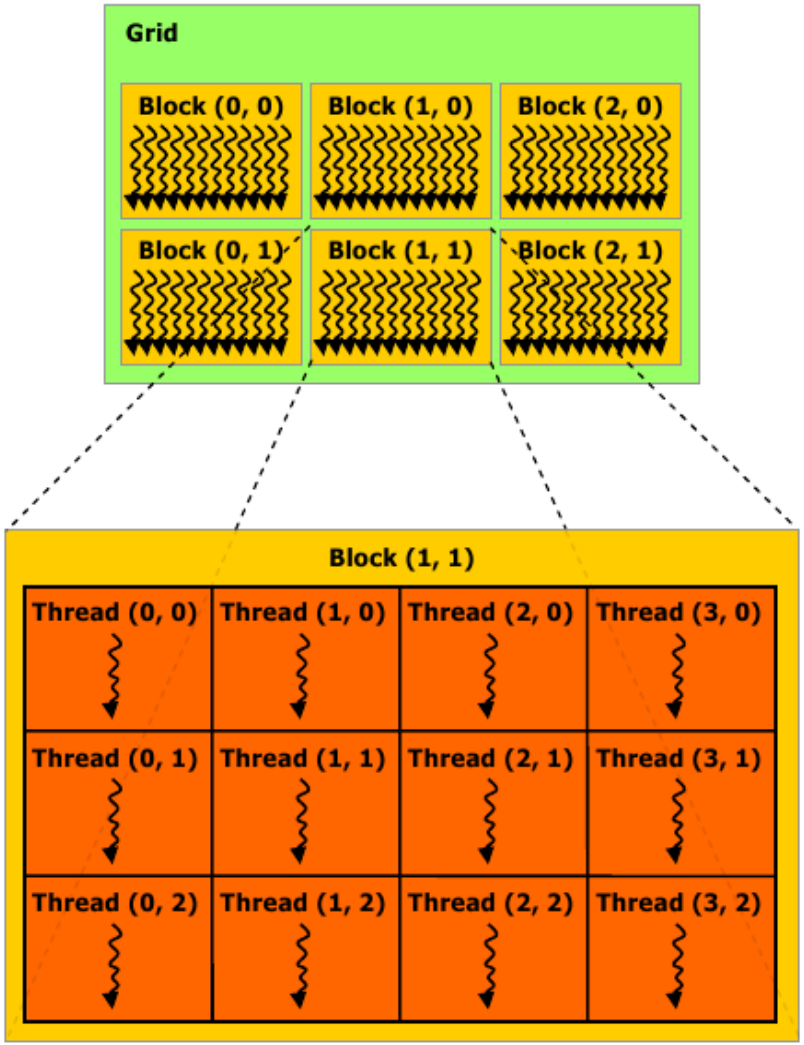


Architecture des GPU

Source : CUDA Programming Guide

- Architecture « many-core »
- Exécution partiellement SIMD
- 14 Stream Multiprocessor (SM) sur Fermi (C2050)
- Chaque SM possède 32 « CUDA cores »
  soit $14 \times 32 = 448$ « CUDA Cores » sur le C2050
- Registres de 32-bit (32 768 par SM)

Hiérarchie des *threads*



Programmation

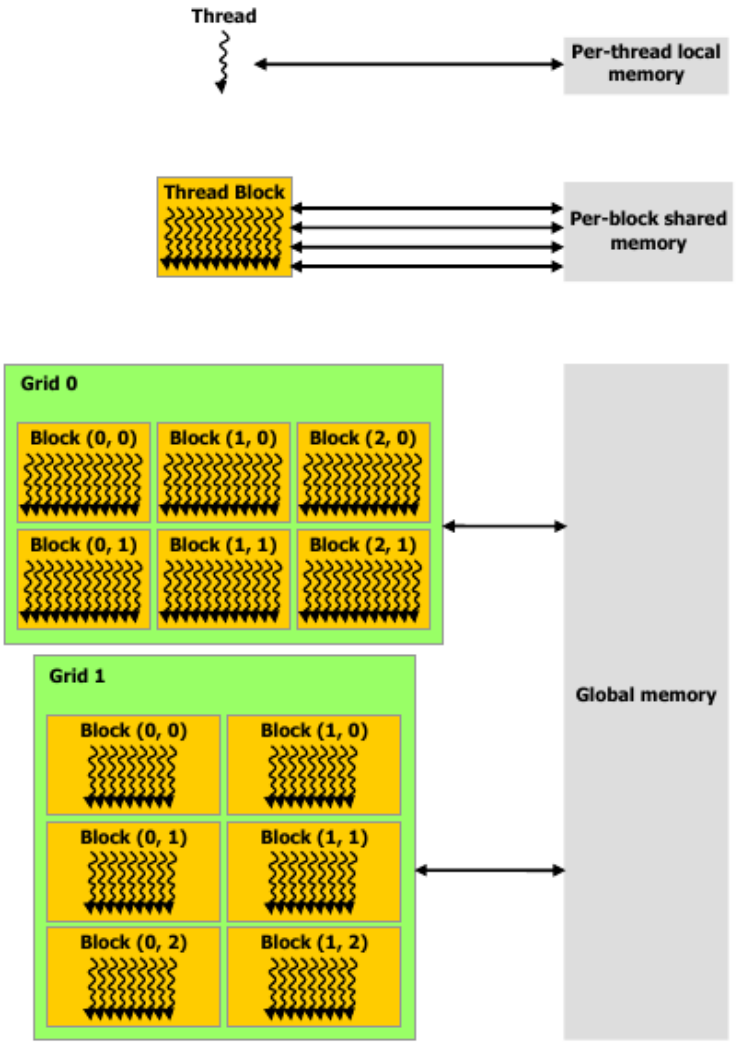
- Bloc composé de *threads*
- Grille composée de blocs

Schéma d'exécution

- 1 bloc est affecté à 1 SM
- SM exécutent chaque bloc par warps
- Un warp est un groupe de 32 *threads*

Source : CUDA Programming Guide

Hiérarchie mémoire



Source : CUDA Programming Guide

Schéma mémoire

- Mémoire partagée :
  - rapide (latence < 10 cycles)
  - locale à un SM
  - petite quantité (48 Ko par SM)
- Mémoire globale :
  - lente (latence > 400 cycles)
  - accessible à tous les *threads*
  - grande quantité (2,8 Go avec ECC)
  - attention aux accès amalgamés (*coalesced*).

Arithmétique sur C2050

Arithmétique flottante

- IEEE-754 compliant
- 32-bit Add, Mul, FMA : 32 opérations/cycle/SM
- 64-bit Add, Mul, FMA : 16 opérations/cycle/SM
- 32-bit Inverse : 4 opérations/cycle/SM
- Performances crêtes
  - simple précision : 1.03 Tflop/s
  - double précision : 515 Gflop/s

Arithmétique sur C2050

Arithmétique entière

- 32-bit Add, opérations logiques : 32 opérations/cycle/SM
- 32-bit Mul, FMA, décalages, comparaison : 16 opérations/cycle/SM
- Division et modulo émulés (< 20 instructions)
- 64-bit aussi
  - 64-bit Add, opérations logiques : 16 opérations/cycle/SM
  - 64-bit Mul, FMA, décalages, comparaison : 4 opérations/cycle/SM

Algorithme L

But

Soit $f(x)$ une fonction trancendante
Soit $P(x)$ une approximation polynomiale de $f(x)$ avec une erreur ϵ

- Trouver les arguments x (*pires cas*) $\ll$ proches $\gg$ d'un flottants ou du milieu de 2 flottants

Génération d'approximations affines

Intervalle	Nombre de points	degré approximation
I	2^{40}	$d > 2$

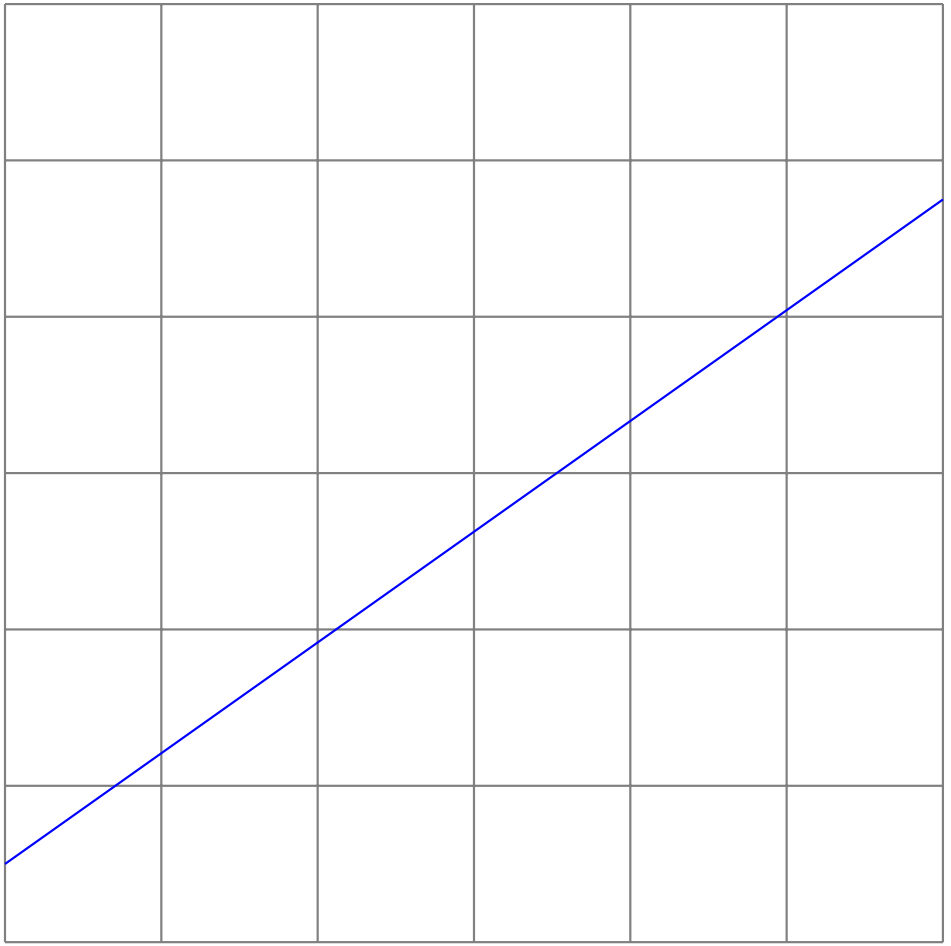
1. f est approchée par P sur I (développement de Taylor) .
2. Répartition du calcul sur plusieurs machines

Génération d'approximations affines

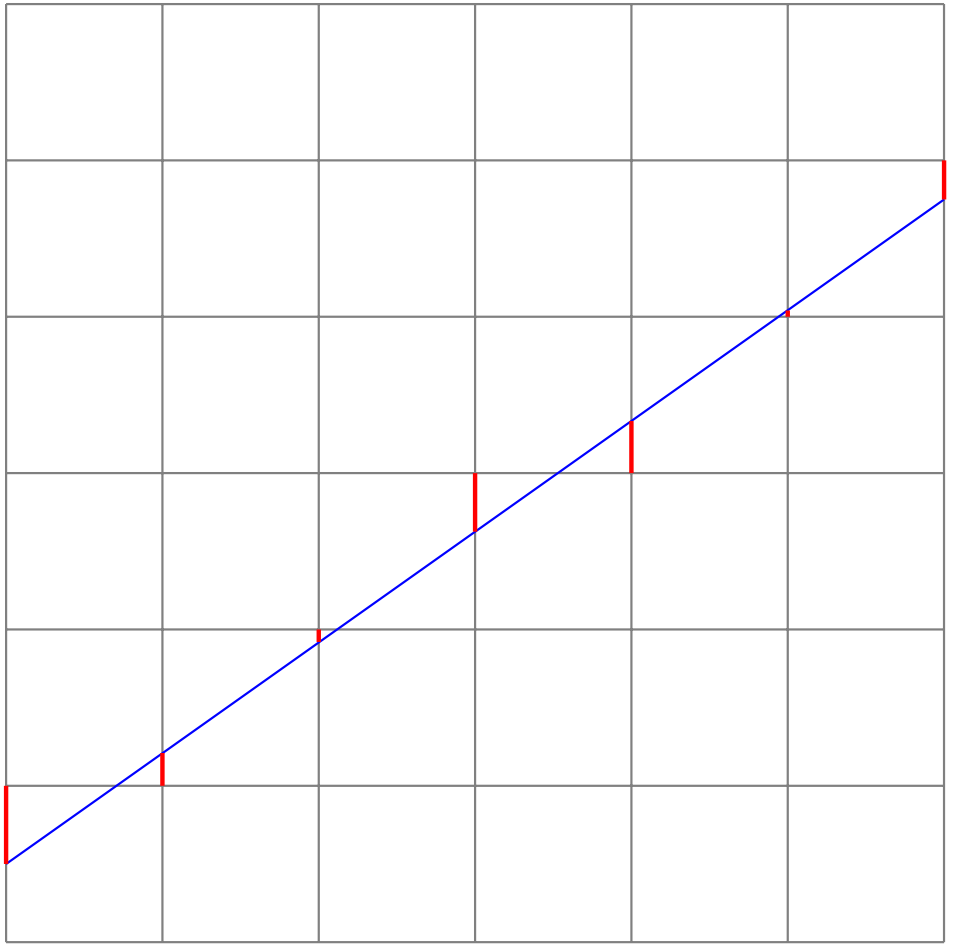
Intervalle	Nombre de points	degré approximation
I	2^{40}	$d > 2$
J	2^{15}	2

1. f est approchée par P sur I (développement de Taylor) .
2. Répartition du calcul sur plusieurs machines
3. Réduction du degré de l'approximation.

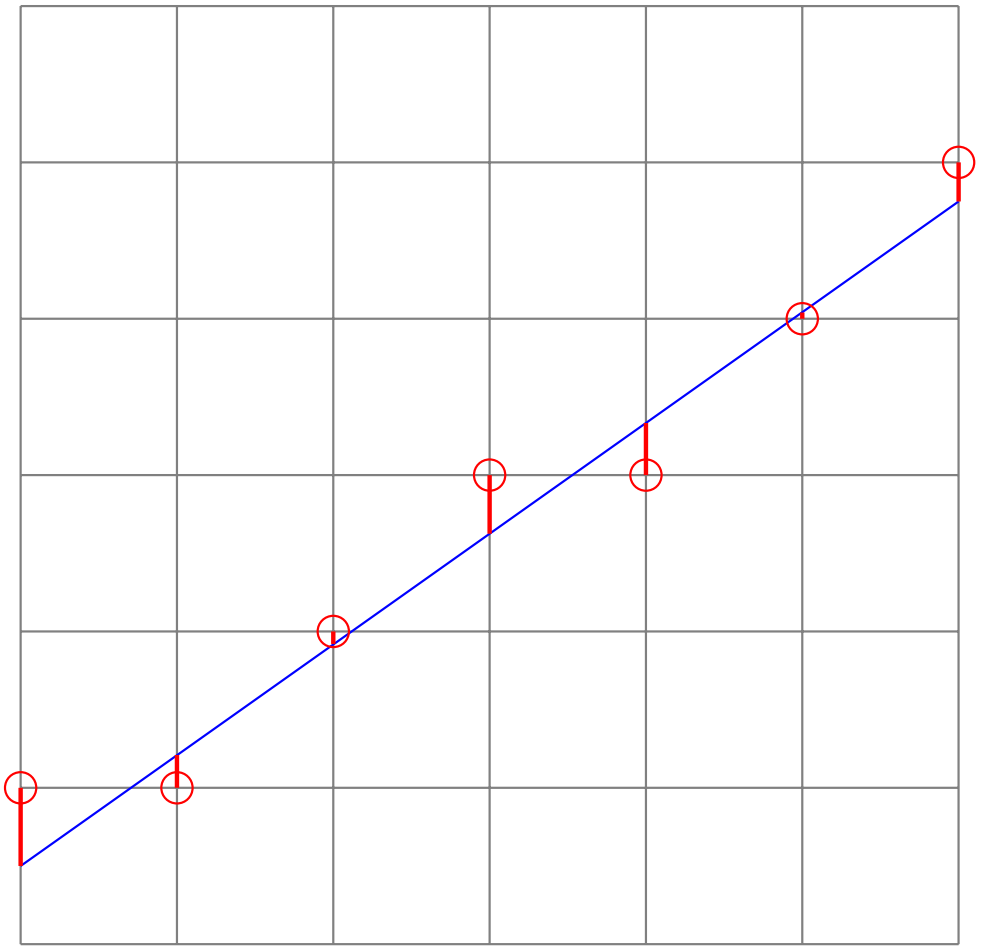
Test $\{b - a \cdot x\} < \epsilon$



Test $\{b - a \cdot x\} < \epsilon$



Test $\{b - a \cdot x\} < \epsilon$



- Algo naïf : $O(N)$ comparaison
- Algorithme-L (basé sur le théorème des 3 distances) : $O(log(N))$ comparaison

Test $\{b - a \cdot x\} < \epsilon$

Stratégie

- **Phase 1** : Algorithme-L sur J
- **Phase 2** :
  1. On divise J en 8 sous intervalles K.
  2. Algorithme-L sur chaque K
- **Phase 3** : Recherche exhaustive (Algo naïf) sur les K ayant échoués le test.

Exponentielle sur un intervalle I : 243 pires cas.

Phase	Nb intervalles
1	$2^{25} \approx 33.55$M
2	109 048
3	2182

Algorithme-L par soustraction [Lefèvre1997]

initialisation : $x \leftarrow a\,;\ y \leftarrow 1\,;\ d \leftarrow b\,;$
$u \leftarrow 1\,;\ v \leftarrow 1\,;$

if $d < d_0$ **then return** Fail;
while *True* **do**
  if $d < x$ **then**
    while $x < y$ **do**
      if $u + v \geq N$ **then return**
      Success;
      $y \leftarrow y - x\,;\ u \leftarrow u + v\,;$
    if $u + v \geq N$ **then return**
    Success;
    $x \leftarrow x - y\,;\ v \leftarrow v + u;$
  else
    $d \leftarrow d - x;$
    if $d < d_0$ **then return** Fail;
    while $y < x$ **do**
      if $u + v \geq N$ **then return**
      Success;
      $x \leftarrow x - y\,;\ v \leftarrow u + v\,;$
    if $u + v \geq N$ **then return**
    Success;
    $y \leftarrow y - x\,;\ u \leftarrow u + v;$

Algorithme-L par division [Lefèvre2005]

initialisation : $x \leftarrow a\,;\ y \leftarrow 1\,;\ d \leftarrow b\,;$
$u \leftarrow 1\,;\ v \leftarrow 1\,;$

if $d < d_0$ **then return** Fail;
while *True* **do**
  if $d < x$ **then**
    while $x < y$ **do**
      if $u + v \geq N$ **then return**
      Success;
      $y \leftarrow y - x\,;\ u \leftarrow u + v\,;$
    if $u + v \geq N$ **then return**
    Success;
    $x \leftarrow x - y\,;\ v \leftarrow v + u;$
  else
    $d \leftarrow d - x;$
    if $d < d_0$ **then return** Fail;
    while $y < x$ **do**
      if $u + v \geq N$ **then return**
      Success;
      $x \leftarrow x - y\,;\ v \leftarrow u + v\,;$
    if $u + v \geq N$ **then return**
    Success;
    $y \leftarrow y - x\,;\ u \leftarrow u + v;$

initialisation : $x \leftarrow a\,;\ y \leftarrow 1\,;\ d \leftarrow b\,;$
$u \leftarrow 1\,;\ v \leftarrow 1\,;$

if $d < d_0$ **then return** Fail;
while *True* **do**
  if $d < x$ **then**
    $q \leftarrow \lfloor x/y \rfloor;$
    $y \leftarrow y - q \times x\,;$
    $u \leftarrow u + q \times v;$
    if $u + v \geq N$ **then return**
    Success;
    $x \leftarrow x - y\,;\ v \leftarrow u + v;$
  else
    $d \leftarrow d - x;$
    if $d < d_0$ **then return** Fail;
    $q \leftarrow \lfloor y/x \rfloor;$
    $x \leftarrow x - q \times y\,;$
    $v \leftarrow v + q \times u;$
    if $u + v \geq N$ **then return**
    Success;
    $y \leftarrow y - x\,;\ u \leftarrow u + v;$

Comportement des variantes

Version soustraction

- Coût d'une itération faible
- Si $|x - y| \ll$ petit $\gg$: nombreux tours de boucle faible
- Si $|x - y| \ll$ grand $\gg$: nombre de tours de boucle élevé

Version division

- Coût constant mais élevé

Version hybride

- Si $|x - y| < 2^3$: version soustraction
- Si $|x - y| \geq 2^3$: version division

Déploiement sur GPU

Avancement

- Calcul des approximations affines toujours sur CPU.
- Déploiement du test $\{b - a \cdot x\} < \epsilon$
  - 1 *thread* par intervalle $J \Rightarrow 2^{25}$ *threads*
  - Mise en place des accès amalgamés
  - Taille de bloc optimale : 256 *threads*/bloc

Temps du test $\{b - a \cdot x\} < \epsilon$

Variante	Temps CPU	Temps GPU	**Gain**
Soustraction	4.34	0.35	**12.4**
Hybride	5.1	0.44	**11.6**
Division	13.54	0.71	**19.1**

Diminution de la divergence

Divergence

- SM = Exécution SIMD

Diminution de la divergence

Divergence

- SM = Exécution SIMD
- 2 *threads* sont divergents au sein d'un *warp* s'ils empruntent 2 branches différentes

Diminution de la divergence

Divergence

- SM = Exécution SIMD
- 2 *threads* sont divergents au sein d'un *warp* s'ils empruntent 2 branches différentes $\Longrightarrow$ sérialisation des branches.

Diminution de la divergence

Divergence

- SM = Exécution SIMD
- 2 *threads* sont divergents au sein d'un *warp* s'ils empruntent 2 branches différentes $\Longrightarrow$ sérialisation des branches.

Mesure de la divergence

Nous mesurons le nombre de tests divergents sur 1 SM.

Variante	Temps	Divergence
Soustraction	0.35	6,93 M
Hybride	0.46	6.29 M
Division	0.76	1,63 M

Diminution de la divergence

Gain espéré

- Test avec 2^{25} fois le même intervalle
- Seule la phase 1 est mesurée
- Gain par rapport à la version GPU

Variante	Temps	Gain
Soustraction	0.04	7.5
Hybride	0.08	3.5
Division	0.15	2.76

Diminution de la divergence

initialisation : $x \leftarrow a\,;\ y \leftarrow 1\,;\ d \leftarrow b\,;$
$u \leftarrow 1\,;\ v \leftarrow 1\,;$

if $d < d_0$ **then return** Fail;
while *True* **do**
  if $d < x$ **then**
    while $x < y$ **do**
      if $u + v \geq N$ **then return**
      Success;
      $y \leftarrow y - x\,;\ u \leftarrow u + v\,;$
    if $u + v \geq N$ **then return**
    Success;
    $x \leftarrow x - y\,;\ v \leftarrow v + u;$
  else
    $d \leftarrow d - x;$
    if $d < d_0$ **then return** Fail;
    while $y < x$ **do**
      if $u + v \geq N$ **then return**
      Success;
      $x \leftarrow x - y\,;\ v \leftarrow u + v\,;$
    if $u + v \geq N$ **then return**
    Success;
    $y \leftarrow y - x\,;\ u \leftarrow u + v;$

Diminution de la divergence

initialisation : $x \leftarrow a\,;\ y \leftarrow 1\,;\ d \leftarrow b\,;$
$u \leftarrow 1\,;\ v \leftarrow 1\,;$

if $d < d_0$ **then return** Fail;
while *True* **do**
  if $d < x$ **then**
    while $x < y$ **do**
      if $u + v \geq N$ **then return** Success;
      $y \leftarrow y - x\,;\ u \leftarrow u + v\,;$
    if $u + v \geq N$ **then return** Success;
    $x \leftarrow x - y\,;\ v \leftarrow v + u;$
  else
    $d \leftarrow d - x;$
    if $d < d_0$ **then return** Fail;
    while $y < x$ **do**
      if $u + v \geq N$ **then return** Success;
      $x \leftarrow x - y\,;\ v \leftarrow u + v\,;$
    if $u + v \geq N$ **then return** Success;
    $y \leftarrow y - x\,;\ u \leftarrow u + v;$

initialisation : $x \leftarrow a\,;\ y \leftarrow 1\,;\ d \leftarrow b\,;$
$u \leftarrow 1\,;\ v \leftarrow 1\,;$

if $d < d_0$ **then return** Fail;
while *True* **do**
  $swap\_bool \leftarrow d \geq x;$
  if $swap\_bool = True$ **then**
    $d \leftarrow d - x;$
    if $d < d_0$ **then return** Fail;
    $SWAP(x, y)\,;\ SWAP(u, v);$
  while $x < y$ **do**
    if $u + v \geq N$ **then return** Success;
    $y \leftarrow y - x\,;\ u \leftarrow u + v\,;$
  if $u + v \geq N$ **then return** Success;
  $x \leftarrow x - y\,;\ v \leftarrow v + u;$
  if $swap\_bool = True$ **then**
    $SWAP(x, y)\,;\ SWAP(u, v);$

Diminution de la divergence

Variante	Avec swap		Impact	
	Temps	Div.	Gain	Div.
Soustraction	0.37	4,90 M	0.9	-29.3%
Hybride	0.39	4,82 M	1.2	-23.4%
Division	0.63	0,97 M	1.2	-40.5%

- swap = coût supplémentaire, surtout pour la soustraction
- Travaux en cours
  - Diminution du coût du swap
  - Profiling : étudier les autres sources de divergence
  - Déroulage de boucle

Détail des temps

Variante	Calcul Approx. (CPU)	Transfert Coeffs	Calcul pires cas	Transfert pires cas	Total
Soustraction	7.69	0.62	0.35	0.04	8.70
Hybride	7.77	0.61	0.44	0.04	8.89
Division	7.79	0.62	0.71	0.04	9.16

- Temps de transfert $>$ Temps de calcul sur GPU
- De 63% sur CPU à 85% sur GPU
- Génération des approximations bloquant

Génération des approximations affines sur GPU

Méthode 1

$$P(x) \mapsto P(x+k),\ k \text{ taille de l'intervalle}$$

Translation du polynôme.

Méthode 2

Translation par la méthode des « différences divisées ».

Interpolation de Newton + somme de coefficients (table des différences)

Génération des approximations affines sur GPU

Méthode 1

$P(x) \mapsto P(x+k)$, k taille de l'intervalle

$P(x) \mapsto P(x+y \cdot k)$, y indice de l'intervalle

Translation du polynôme.

Méthode 2

Translation par la méthode des « différences divisées ».

Interpolation de Newton + somme de coefficients (table des différences)

Intrinsèquement séquentielle

Génération des approximmations affines sur GPU

Proposition 1

2^{25} *threads* qui translatent leurs coefficients par la méthode 1.

Proposition 2

- On forme i' paquets de j intervalles J
- qui calculent les coefficient du premier intervalle du paquet (méthode 1 ou interpolation de Newton)
- puis qui génèrent j intervalles par sommation de coefficients en parallèle

Problèmes

Multi-précision sur GPU

- Multi-précision à implanter sur GPU.
- Borner la taille des coefficients ?
- Trouver le nombre de bits optimal pour les coefficients
  $\Rightarrow$ meilleur compromis performance / precision

Meilleure approximation en norme infini

- pour se passer de la multi-précision ?
- pour augmenter $\#J$?


Vincent Lefèvre.
An Algorithm that Computes a Lower Bound on the Distance Between a Segment and, 1997.


Vincent Lefèvre.
Moyens arithmétiques pour un calcul fiable.
PhD thesis, école normale supérieure de Lyon, 2000.


Vincent Lefèvre.
New Results on the Distance between a Segment and $\mathbb{Z}^2$. Application to the Exact Rounding.
17th IEEE Symposium on Computer Arithmetic (ARITH'05), pages 68–75, 2005.


Nvidia.
CUDA C Programming Guide.
version 3.2 edition, 2010.