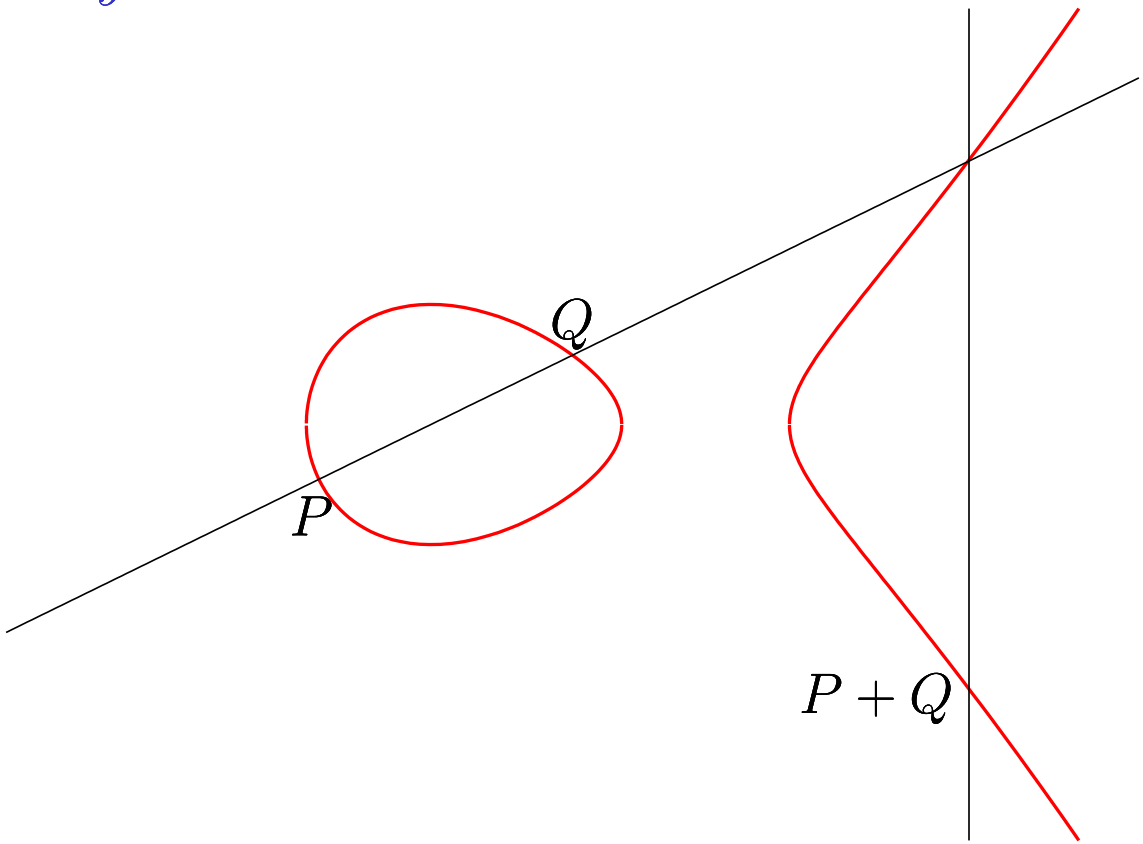
ECM sur GPU

Cyril Bouvier

30 juin 2011

Courbes elliptiques

$$E : y^2 = x^3 + Ax + B$$



L'élément neutre est le point à l'infini en coordonnées projectives $(0 : 1 : 0)$.

ECM

Dans toute la suite N est l'entier que l'on veut factoriser.
L'étape 1 d'ECM :

- on choisit une courbe elliptique E avec un point P et une fonction rationnelle ϕ qui envoie l'élément neutre de E sur 0 ;
- on choisit un entier B_1 et on pose

$$s = \prod_{\substack{\pi \leq B_1 \\ \pi \text{ premier}}} \pi^{\lfloor \log(B_1)/\log(\pi) \rfloor};$$

- on calcule sP en utilisant la loi de groupe de E sur $\mathbb{Q}$ où l'on réduit modulo N après chaque calcul intermédiaire ;
- et on espère que $\text{pgcd}(\phi(sP), N)$ nous donne un facteur de N.

Courbes de Montgomery

- Courbes de Montgomery : pour $b(a^2 - 4) \neq 0$

$$by^2 = x^3 + ax^2 + x$$

- On utilise uniquement les coordonnées projectives X et Z. Un point est noté $(X :: Z)$, et on prend $\phi(X :: Z) = Z$.
- Les formules d'addition sont indépendantes des coefficients de la courbe et les formules de doublement ne dépendent des coefficients de la courbe que par $d = \frac{a+2}{4}$.

Paramétrisation de Montgomery

- Paramétrisation de Montgomery : pour tout $c \ 2 \ \mathbb{Q}nf \quad \frac{9}{8}, \quad 1, 0g,$

$$(16c + 18)y^2 = x^3 + (4c + 2)x^2 + x.$$

- On prend comme point de départ $(X_0 :: Z_0) = (2 :: 1)$.
- On peut choisir c tel que $d = c + 1$ soit "petit". Transforme une multiplication entre deux grands entiers par une multiplication par un « petit » entier lors du doublement d'un point sur la courbe.

$$Z_2 = ((X + Z)^2 \quad (X \quad Z)^2)((X \quad Z)^2 + d \quad ((X + Z)^2 \quad (X \quad Z)^2))$$

$$X_2 = (X + Z)^2(X \quad Z)^2$$

Montgomery binary ladder

Fonction 1 Montgomery binary ladder

Input : P et s.

Output : Q tel que $Q = sP$

$P_1 = P$

$P_2 = 2P$

On écrit s en base 2, $s = (s_{k-1} \quad s_0)_2$

for $i = k \quad 2$ downto 0 **do**

if $s_i = 0$ **then**

$P_2 = P_1 \boxplus P_2$, $P_1 = 2P_1$

else

$P_1 = P_1 \boxplus P_2$, $P_2 = 2P_2$

$Q = P_1$

$$X_{P_1 \boxplus P_2} = Z_{P_1 \boxminus P_2} \quad ((X_{P_2} \quad Z_{P_2})(X_{P_1} + Z_{P_1}) + (X_{P_2} + Z_{P_2})(X_{P_1} \quad Z_{P_1}))^2$$
$$Z_{P_1 \boxplus P_2} = X_{P_1 \boxminus P_2} \quad ((X_{P_2} \quad Z_{P_2})(X_{P_1} + Z_{P_1}) \quad (X_{P_2} + Z_{P_2})(X_{P_1} \quad Z_{P_1}))^2$$

Montgomery binary ladder

Fonction 1 Montgomery binary ladder

Input : P et s.

Output : Q tel que $Q = sP$

$P_1 = P$

$P_2 = 2P$

On écrit s en base 2, $s = (s_{k-1} \quad s_0)_2$

for $i = k \quad 2$ downto 0 **do**

if $s_i = 0$ **then**

$P_2 = P_1 \boxplus P_2$, $P_1 = 2P_1$

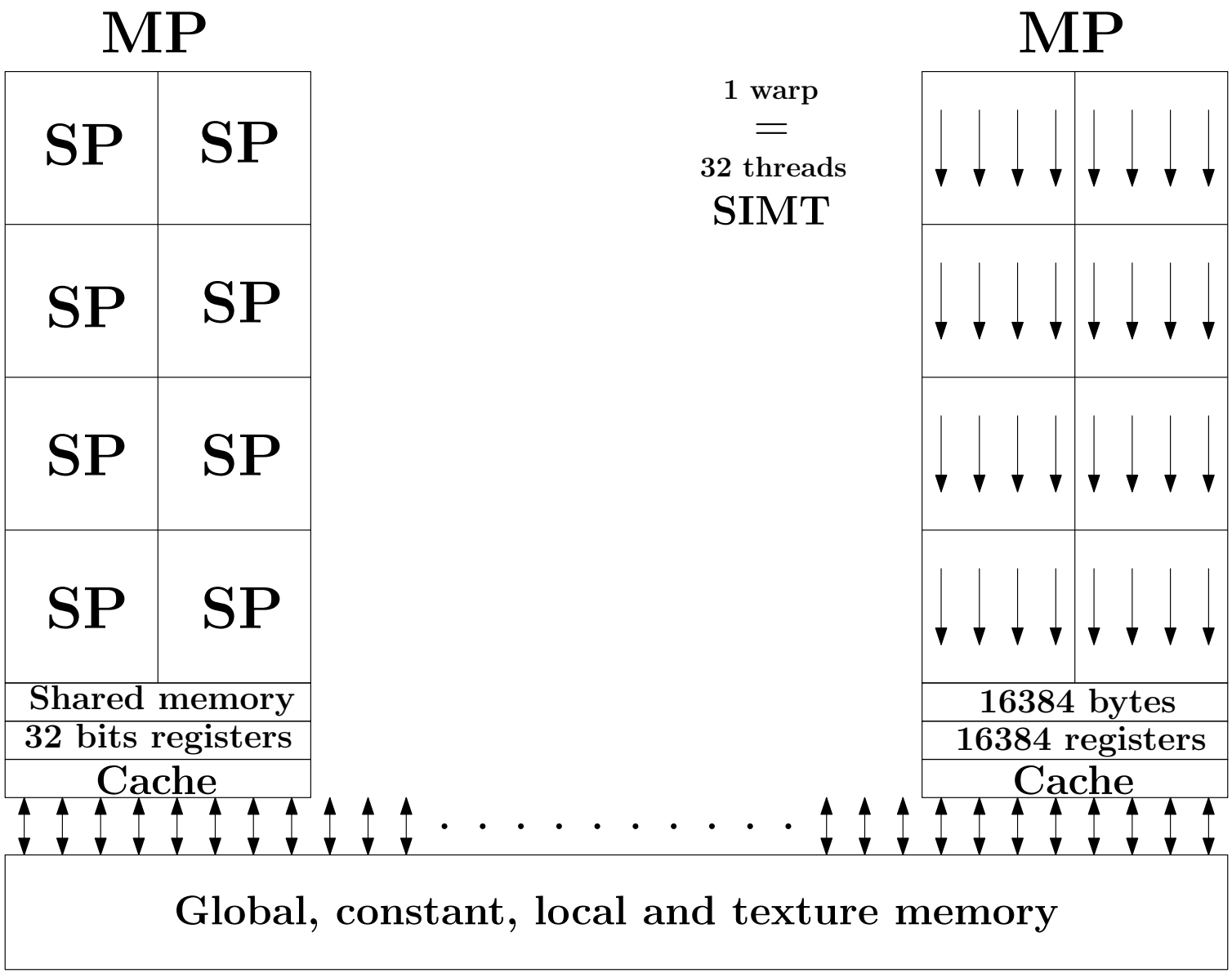
else

$P_1 = P_1 \boxplus P_2$, $P_2 = 2P_2$

$Q = P_1$

$$X_{P_1 \boxplus P_2} = 1 \quad ((X_{P_2} \quad Z_{P_2})(X_{P_1} + Z_{P_1}) + (X_{P_2} + Z_{P_2})(X_{P_1} \quad Z_{P_1}))^2$$
$$Z_{P_1 \boxplus P_2} = 2 \quad ((X_{P_2} \quad Z_{P_2})(X_{P_1} + Z_{P_1}) \quad (X_{P_2} + Z_{P_2})(X_{P_1} \quad Z_{P_1}))^2$$

Cartes graphiques NVIDIA



Caractéristiques des cartes

Capacité de calcul	1.3	2.0
Nombre de threads dans un warp	32	
Nombre de cœurs par MP	8	32
Nombre maximum de threads par bloc	512	1024
Nombre maximum de warps par MP	32	48
Quantité de mémoire partagée par MP	16ko	48ko
Nombre de banques de mémoire partagée	16	32
Nombre de registres 32 bits par MP	16384	49152
Quantité de mémoire locale par thread	16ko	512ko
Quantité de mémoire constante	64ko	

Table: Caractéristiques de certaines cartes NVIDIA

Accès coalescents et conflits de banques

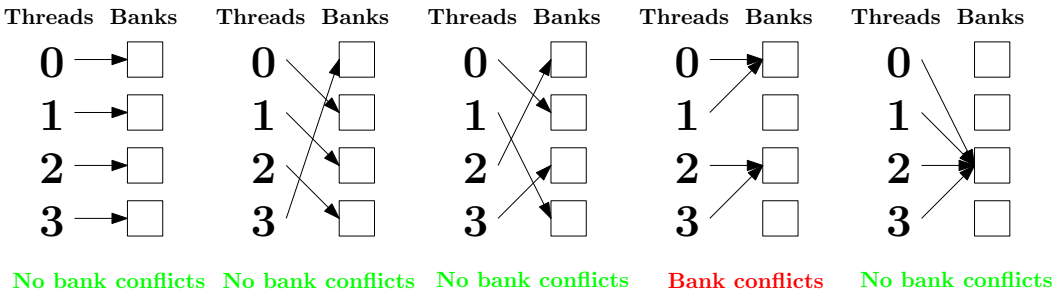


Figure: Différents accès à la mémoire partagée

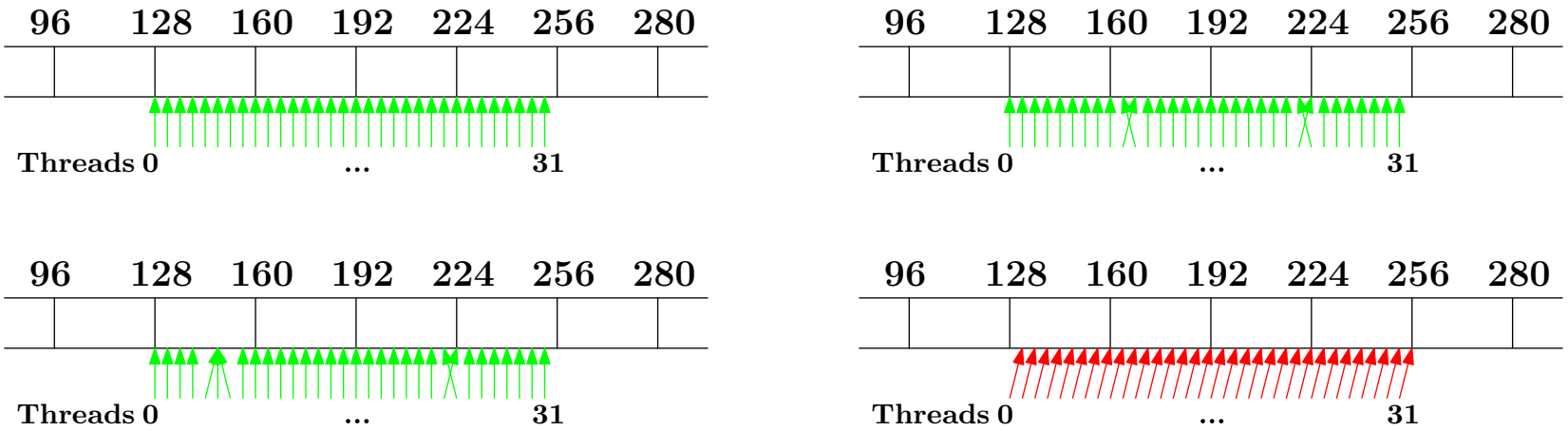


Figure: Différents accès à la mémoire globale

Arithmétique sur GPU

- On manipule des entiers de 1024 bits comme des tableaux d'entiers 32 bits non signés (unsigned int). On pose $B = 2^{1024}$.
- On utilise une représentation carry-save : (a,c) représente le nombre $a + 2^{32}c$. Les retenues sont stockées dans un tableau d'entiers 32 bits (int).
- On manipule les entiers sous leur représentation de Montgomery : on représente un entier a modulo N par $\overline{a} = (a \quad B) \mod N$. L'addition et la soustraction deviennent $\overline{a \quad b} = \overline{a} \quad \overline{b} \mod N$. La multiplication devient $\overline{a \quad b} = \overline{a} \quad \overline{b} \quad B^{-1} \mod N$.

Degré de parallélisme

- Premier niveau de parallélisme : un warp (32 threads) s'occupe du calcul sur une courbe. Sur une carte de capacité de calcul 1.3, on peut traiter 16 courbes par MP, sur une carte 2.0 on peut traiter 32 courbes par MP.
- Second niveau de parallélisme : à l'intérieur d'un warp, un thread s'occupe d'un chiffre (*ie* d'un unsigned int) du grand entier.

Répartition du calcul

- Le programme hôte sur le processeur se charge de tous les précalculs et initialisations (coefficients des courbes, mises sous forme de Montgomery, *etc*) et copie tout ce qui est nécessaire dans la mémoire globale de la carte graphique.
- Le GPU effectue, à chaque appel, une addition et un doublement sur toutes les courbes en parallèle. Pour cela il copie de la mémoire globale vers la mémoire partagée les coordonnées des points des courbes, effectue les calculs nécessaires modulo N et copie les nouvelles coordonnées de la mémoire partagée vers la mémoire globale.
- Enfin l'hôte se charge de copier les résultats à partir de la mémoire globale du GPU et du calcul des pgcd.

Appel du GPU en CUDA

```
//Initialisation et copie
...

dim3 dimBlock(SIZE_NUMBER,CURVES_BY_BLOCK);
dim3 dimGrid(number_of_curves/CURVES_BY_BLOCK);

for (j=mpz_sizeinbase(s,2)-2;j>=0;j--)
{
  if (mpz_tstbit(s,j)==1)
    Cuda_Ell_DblAdd<<<dimGrid,dimBlock>>>
        (d_xB,d_zB,d_xA,d_zA,firstinvd);
  else
    Cuda_Ell_DblAdd<<<dimGrid,dimBlock>>>
        (d_xA,d_zA,d_xB,d_zB,firstinvd);
}
//Copie des resultats et calcul des pgcd
...
```

Exemple de code CUDA pour le GPU : l'addition

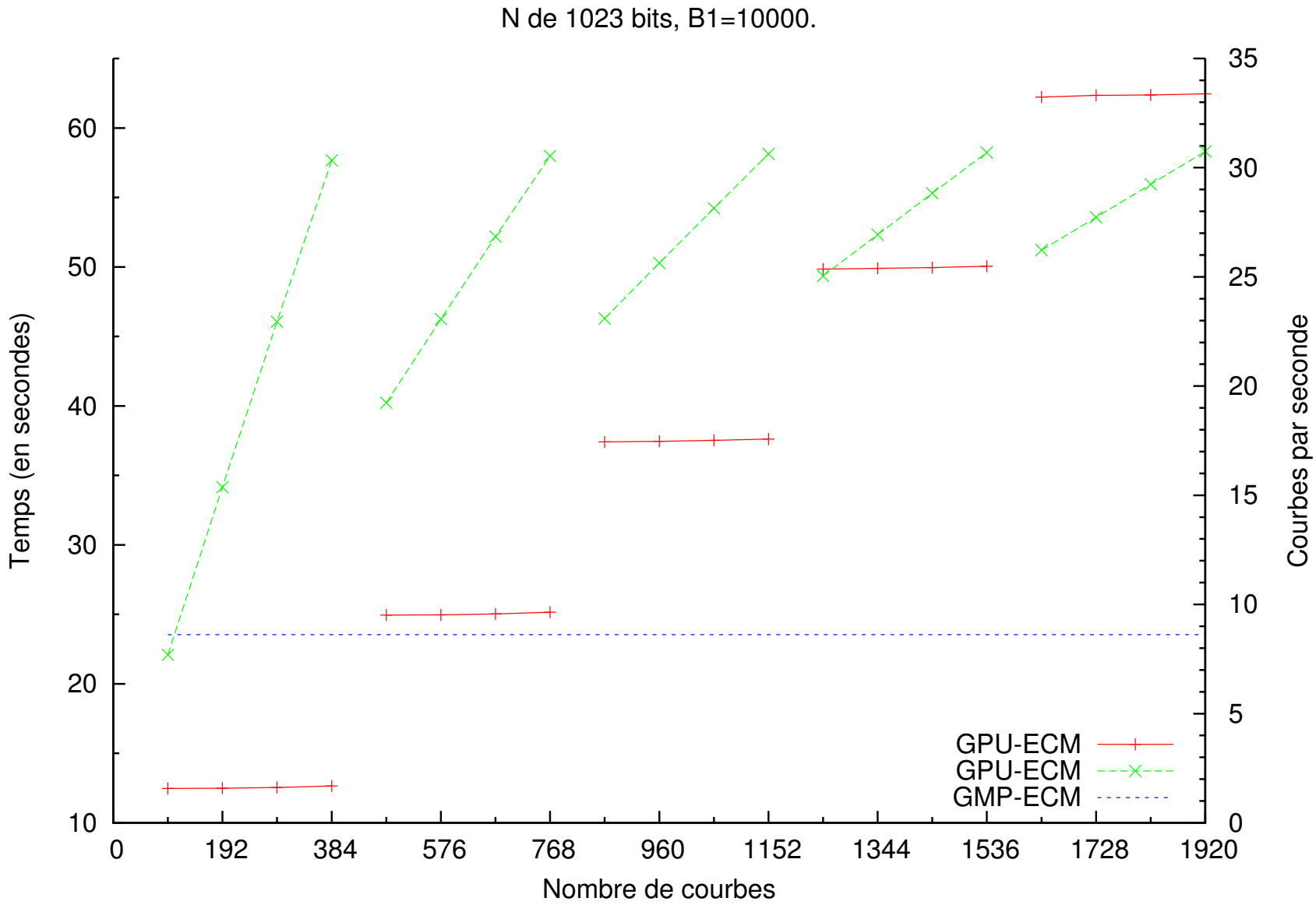
```
//r+2^32*c <- a + b
__device__ void Cuda_Add
(biguint_t r, dbigint_t cy , biguint_t a, biguint_t b)
{
__asm__("add.cc.u32 %0, %1, %2"
        : "=r"(r[threadIdx.x])
        : "r"(a[threadIdx.x]) ,"r"(b[threadIdx.x]));
__asm__("addc.s32 %0, 0, 0;" :"=r"(cy[threadIdx.x]));
}
```

Résultats

B_1	10^6	10^7	10^8	10^9
Taille de s (en bits)	1442099	14424844	144266969	1442697344
Mémoire nécessaire pour stocker s	180ko	1,80Mo	18,0Mo	180Mo
Temps de calcul de s	40ms	790ms	12,3s	3m01s
GPU-ECM (384 courbes)	20min54	3h29	35h	348h

Table: Temps de calcul de s.

Résultats


N de 1023 bits, B1=10000.
Temps (en secondes)
Courbes par seconde
Nombre de courbes
0
192
384
576
768
960
1152
1344
1536
1728
1920
10
20
30
40
50
60
0
5
10
15
20
25
30
35
GPU-ECM
GPU-ECM
GMP-ECM

Résultats

	GMP-ECM	GPU-ECM	GPU-EECM
Taille de N (en bits)	1023		210
Débit (courbes/sec)	10,5	37,1	4928
Débit normalisé (1)	10,5	37,1	83,1
Débit normalisé (2)	10,5	37,1	40,1

Table: Différentes implémentations de l'étape 1 d'ECM ($B_1 = 8192$).