Températures, erreurs matérielles et GPU

David Defour, Eric Petit

ComPAS'2013 : RenPar'21 / SympA'15/ CFSE'9
Grenoble, France, du 16 au 18 janvier 2013

Températures, erreurs matérielles et GPU

David Defour, Eric Petit

Université de Perpignan Via Domitia,
Laboratoire DALI - 54 avenue Paul Alduy
64000 Perpignan- France
david.defour@univ-perp.fr

Université de Versailles Saint-Quentin,
Laboratoire ITACA - 45 avenue des Etats-Unis
78035 Versailles - France
eric.petit@uvsq.fr

Résumé
Les co-processeurs massivement parallèles offrent une grande puissance de calcul en intégrant un nombre croissant de coeurs. De plus, les technologies de gravure sont de plus en plus denses, les fréquences plus élevées et les voltages plus faibles. La combinaison de ces facteurs tend à augmenter significativement la probabilité d'erreurs de calcul dues aux erreurs physiques. Dans cet article, nous présentons les résultats d'expériences caractérisant l'impact du vieillissement des cartes de type GPU NVIDIA sur l'apparition d'erreurs physiques. En se basant sur une plateforme logicielle de micro-tests écrit en OpenCL, nous sollicitons de manière intensive et ciblée certains éléments de l'architecture pour caractériser les éventuelles erreurs de calcul. Afin d'accélérer le processus de vieillissement et donc d'apparition d'erreurs, nous utilisons une technique standard, consistant à faire fonctionner les processeurs à des températures élevées. Ce papier introduit les phénomènes physiques et l'état de l'art liés aux méthodes de caractérisation des erreurs physiques. Nous présentons ensuite notre adaptation de ces protocoles au cas des GPU NVIDIA ainsi que nos premiers résultats. Ces derniers montrent comment et où ces erreurs se produisent.

Mots-clés : GPU, erreur physique, température, vieillissement, electro-migration

---

1. Introduction

L'apparition de fautes dans les composants intégrés est un phénomène connu [11, 12, 13]. Les nouveaux coprocesseurs matériels comme les GPU ou les Xeon Phi (MIC) n'échappent pas à la règle. Parmi les facteurs pouvant influer sur l'existence et l'apparition d'erreurs physiques se trouvent l'abaissement du voltage de fonctionnement, la fréquence croissante, la finesse de gravure, la densité et le nombre croissant de transistors [8, 7, 10]. Les fautes liées au matériel pouvant survenir lors de l'exécution d'un programme sont de trois sortes [7] :

- **Les fautes permanentes** : elles sont soit présentes depuis la gravure en usine de la puce, soit liées à un dommage permanent dû au vieillissement.
- **Les fautes transitoires** : ponctuelles et non reproductibles, elles sont dues à des interférences extérieures.
- **Les fautes intermittentes** : ce sont des fautes temporaires mais reproductibles dans les mêmes conditions. Elles sont dues à un vieillissement ou des variations dans la fabrication de la puce.

Si ces problèmes sont communs avec les autres types de processeurs, ces accélérateurs matériels font appel à un grand nombre de coeurs vectoriels pour traiter le travail en parallèle. On constate que l'impact de ces erreurs dans la cas des applications graphiques, en particulier des erreurs intermittentes, est limité. Cependant, on assiste à une montée en puissance de ce type de solutions pour l'accélération de

codes de calcul intensif. L'utilisation de ces coprocesseurs pour le calcul scientifique, en particulier ceux du Calcul Haute Performance, HPC, pose alors la question de la fiabilité des résultats produits et de leur tenu dans le temps [9, 28]

Certaines études ayant déjà constaté des dysfonctionnements dans la hiérarchie mémoire des GPU [9], des solutions dites *Error-Correcting Code*, ECC, sont déjà proposées pour les GPU Tesla NVIDIA. Cette technologie ne permet pas une fiabilité totale et les taux d'erreurs restent encore significatifs dans les systèmes à grande échelle [22]. Mais à notre connaissance, aucune étude n'a été réalisée sur l'apparition de ces erreurs au sein des autres éléments composant un processeur graphique (unités de calcul, mémoire partagée, registre ...). Dans cet article nous présentons une étude préliminaire visant à mettre en évidence et à caractériser l'apparition d'erreurs physiques sur les processeurs de type GPU.

Pour y parvenir, nous avons mis en place un protocole expérimental pour l'étude de différentes architectures dites *manycore* des fautes intermittentes et permanentes. Nous avons choisi de développer la plateforme dans un langage portable de haut niveau : OpenCL [14]. Le refroidissement des architectures testées a été contraint de façon logicielle et matérielle afin d'augmenter la température de fonctionnement pour permettre d'accélérer le vieillissement des circuits.

La première section de l'article est consacrée à l'état de l'art : définitions et sources des défauts, les protocoles de tests existant, la détection d'erreurs en ligne et les travaux préliminaires sur GPU. La seconde section présente le matériel étudié et son impact sur la méthode. La troisième section présente le protocole expérimental, logiciel et matériel, ainsi que les tests effectués. Enfin nous présentons les résultats obtenus avant de conclure sur l'efficacité de notre méthode et les travaux futurs.

2. État de l'art

Cette section présente successivement les sources possibles d'erreurs matérielles, les méthodes de détection et enfin les travaux antérieurs dans ce domaine sur les GPU.

2.1. Sources des erreurs physiques

Dans le cycle de vie d'un processeur, la première source d'erreur de fonctionnement vient du processus de fabrication. Le test intégral de circuit n'étant pas réalisable sur des circuits de cette taille [1, 25] certaines erreurs restent silencieuses aux tests en usine. Elles apparaissent alors lors de leur utilisation [15, 16]. Sur les GPU, une étude basée sur les statistiques du projet Folding@home [9] fait état de deux tiers des GPU produisant des erreurs détectables, dont 2% des GPU avec une probabilité supérieure à 10^{-4} que le résultat d'un calcul soit faux ; ce taux devient négligeable dans la gamme NVIDIA Tesla. Par ailleurs au sein d'un même processeur la qualité de gravure peut ne pas être uniforme. Certaines parties seront donc plus sujettes aux erreurs intermittentes. Ces dernières sont en effet principalement dûes à des variations de tension et de température qui impactent les délais de transition et l'état logique des transistors [7].

Les bus mémoires des GPU travaillent à des fréquences élevées, et sont particulièrement exposés aux erreurs dûes aux interférences générées par les rayonnements ou les délais induits par des variations de tension [6]. La solution proposée par NVIDIA, basée sur un mécanisme de correction d'erreur de type ECC jusqu'au niveau registre est efficace mais son impact sur la bande passante est significatif (environ 25% sur une carte Tesla 2050 ou 2070). Sur ce type d'architecture où la performance est directement liée au débit de la mémoire, la sanction est immédiate et bien souvent cette option est désactivée sur les systèmes et remplacée par du contrôle en ligne et de la redondance [17, 18].

Ces dernières années, l'abaissement de la tension de fonctionnement a ralenti par rapport à la finesse de gravure. Il en résulte un vieillissement accéléré des processeurs [7, 20] induit par le phénomène d'electro-migration. Ce dernier correspond à la migration de la matière constituant les conducteurs. Avec des tailles de transistors approchant celle de l'atome, ils deviennent d'autant plus facilement défaillant.

2.2. Protocoles standard de test et détection d'erreurs en ligne

Dans l'industrie, les protocoles de test, la classification des résultats, et les unités de mesure sont définies par le *Joint Electron Device Engineering Council*, JEDEC. Pour chaque défaut, le JEDEC définit un ou plusieurs protocoles de test pour les mesurer, de manière accélérée ou non [11, 12, 13]. La standardi-

sation des protocoles permet aux fondeurs de proposer des caractéristiques comparables qui aideront les ingénieurs à choisir le composant adapté à leur contexte (température, humidité, vibration, atmosphère...). Par exemple Xilinx résume dans un document [27] les résultats des tests pour une partie de leurs composants.
La lecture des différents protocoles de tests met en avant la température comme facteur de vieillissement accéléré des circuits. Pour le type d'erreurs visées dans le cadre de ce travail (fautes intermittentes), le JEDEC propose des *stress-tests* avec des températures variant de 120 à 160 degré.

2.3. Travaux préliminaires sur GPU

Il n'existe pas à notre connaissance d'étude sur la fiabilité des GPU relativement aux erreurs intermittentes. L'étude menée par Haque [9] sur les statistiques de Folding@home ne permet ni de déterminer la source des erreurs ni de les caractériser finement. Elle met cependant en évidence la possibilité que des erreurs se produisent au niveau du GPU lui-même et pas forcément dans la mémoire.
Dans cette section nous faisons un rapide tour d'horizon des différents programmes de tests existant pour GPU et décrivons pourquoi ceux-ci sont inadaptés à notre étude. Dans un deuxième temps, nous présentons l'état de l'art des travaux sur la fiabilisation des systèmes utilisant les GPU.

2.3.1. Programes de test sur GPU

Il existe aujourd'hui un certain nombre de benchmarks de type GPGPU comme RODINIA [2], SHOC [4] ou PARBOIL [24]. Ces benchmarks se composent de coeurs de calcul d'applications pour lesquelles le GPU peut être utilisé. Ils permettent de mesurer ou de comparer la performance d'un système ou l'efficacité d'une optimisation. Mais ils ne permettent pas de déterminer si une erreur est apparue, et encore moins d'en définir le type et les caractéristiques précisément. De plus, dans les benchmarks classiques, une erreur intermittente peut passer inaperçue si celle-ci n'impacte pas le résultat final.
Il existe également des benchmarks synthétiques pour GPU composés de microkernels qui ciblent des unités architecturales en particulier [26]. Mais les calculs effectués n'ont aucun sens et par conséquent ils n'intègrent pas de contrôle d'erreurs. De plus, à notre connaissance aucun n'est écrit en OpenCL et ne répond à notre objectif de portabilité.
S'il existe des simulateurs de GPU, comme barra [3], ceux-ci ne permettent pas la caractérisation des erreurs sur les architectures puisque l'évaluation nécessite le support physique. En revanche, ils peuvent s'avérer utile pour évaluer la couverture et le surcoût des méthodes matérielles de détection et de correction d'erreurs.

2.3.2. Détection et correction d'erreurs matérielles sur GPU

Il existe principalement deux approches pour la gestion des erreurs matérielles : la prévention et la détection-correction d'erreur.
Afin de prévenir l'apparition d'erreur, outre le procédé de fabrication, il est possible d'agir sur les paramètres d'utilisation du GPU. Température de fonctionnement, fréquence de la mémoire, du GPU, capacité de refroidissement, etc. NVIDIA semble utiliser ce genre de levier pour la protection des cartes comme en témoigne ses deux brevets [19, 5]. Ils agissent sur la température et le voltage, pour éviter les défaillances et prévenir le vieillissement. Les tests présentés en section 5.1, montrent les éléments caractéristiques de ces mécanismes tels que nous les avons mesurés.
L'autre levier pour fiabiliser les systèmes est la détection et la correction d'erreur. Il existe deux types d'approches :
– La détection matérielle d'erreur. Par exemple l'approche [23] utilise de la redondance et des comparateurs afin de détecter les différences et corrige par un vote à la majorité.
– La détection logicielle d'erreur. Par exemple l'outils hauberk [29] est une méthode logicielle de détection et correction d'erreurs basée sur de la duplication et *checksum* pour les codes hors des boucles

3. Description d'une carte graphique NVIDIA

Les architectures cibles de ce travail sont des coprocesseurs vectoriels. Ces processeurs que l'on appellera *device* dans la suite de cet article, n'interviennent qu'en soutient du processeur *hôte* qui est chargé de contrôler le fonctionnement du device. Nous nous sommes intéressés aux cartes graphiques à base de GPU NVIDIA et pouvant être assemblées par un constructeurs tiers (MSI, Asus, ...). Ces cartes se

composent de plusieurs éléments comme le GPU, le BIOS graphique, la mémoire graphique, des convertisseurs analogiques-digitaux, des circuits d'accélérations dédiés, une interface avec la carte mère, des mécanismes de refroidissement et des puces de gestion de la puissance.

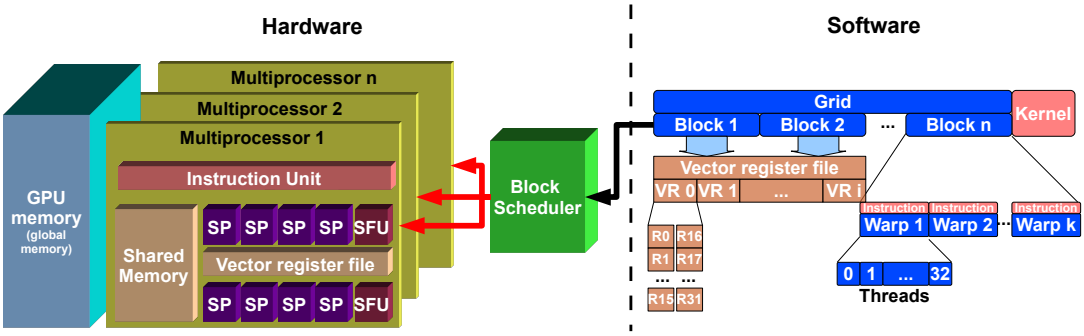


FIGURE 1 – Description de l'architecture G80.

Les processeurs graphiques se caractérisent par une conception interne destinée à exploiter le parallélisme de donnée. Ils intègrent, à cet effet, un grand nombre d'unités de calcul fonctionnant en mode vectoriel. Ces processeurs sont capables de recouvrir les longues latences des instructions mémoires en exploitant plusieurs milliers de threads légers en parallèle. Un exemple de ce type d'architecture est décrit dans la figure 1.

Ce type de processeur peut être vu comme un ensemble de multiprocesseur (*TPC*). Chaque multiprocesseur se compose de plusieurs processeurs de calcul (*PE*) fonctionnant en mode SIMD. A chaque cycle, tous les PE d'un TPC exécutent la même instruction, mais sur des données différentes. Ces multiprocesseurs intègrent plusieurs types de mémoire comme un banc de registres, une mémoire partagée par tous les PE d'un TPC, ainsi qu'une mémoire de constante et de texture accessible en lecture seule.

Les multiprocesseurs sont également composés d'un grande variété d'unités de calcul permettant d'évaluer en matériel des multiplications et accumulations, des FMA, d'unités destinées à l'évaluation de fonctions mathématiques (ex : sin, cos, exp, log), d'unités d'interpolation d'attributs, d'unités de texture et de filtrage. Ces unités supportent aussi bien des entiers que des flottants. Le type de flottant supporté dépend de la génération considérée. Par exemple, chaque multiprocesseur des architectures G80 sont capables d'exécuter des opérations flottantes en simple précision en 4 cycles d'horloge pour un vecteur ou warp de longueur 32.

4. Plateforme de test

Nous avons défini une méthodologie adaptée aux caractéristiques des processeurs graphiques afin d'en analyser leur fiabilité et plus particulièrement d'en définir leur comportement face aux erreurs intermittentes caractéristiques du vieillissement des composants. En particulier, nous voulons au delà de la simple constatation d'erreurs, être capable d'en caractériser leur type, leur fréquence d'apparition, l'unité incriminée et les conséquences sur le matériel.

Pour y parvenir, nous proposons une infrastructure de test basée sur du *Software Based Self-test*. Cette infrastructure se compose d'une partie matérielle et d'une partie logicielle. La partie matérielle permet de définir les conditions dans lesquelles les tests sont réalisés comme par exemple, l'organisation architecturale ou la température de fonctionnement. La partie logicielle est chargée de stresser l'accélérateur, de vérifier l'absence ou la présence des erreurs et de fournir les résultats de l'analyse.

4.1. Partie matérielle

La plateforme matérielle utilisée pour les tests est une machine de type Workstation HP Z800, munie d'un CPU Intel Xeon E5645 cadencé à 2.4 GhZ avec 12 Mo de cache et de 3 modules de RAM de 4 Go chacun. Le CPU est connecté au reste du système par un contrôleur de type Intel X58, combiné à un contrôleur Intel ICH10. Pour les tests, les GPU étaient placés dans un boitier externe de type Tesla D870

connecté au système via un pont dédié (Nforce 100). L'environnement logiciel est une Ubuntu 12.04 munie de la plateforme OpenCL 1.1 CUDA 4.2.1 (driver propriétaire NVIDIA 295.41). Cette configuration est décrite dans la figure 2.

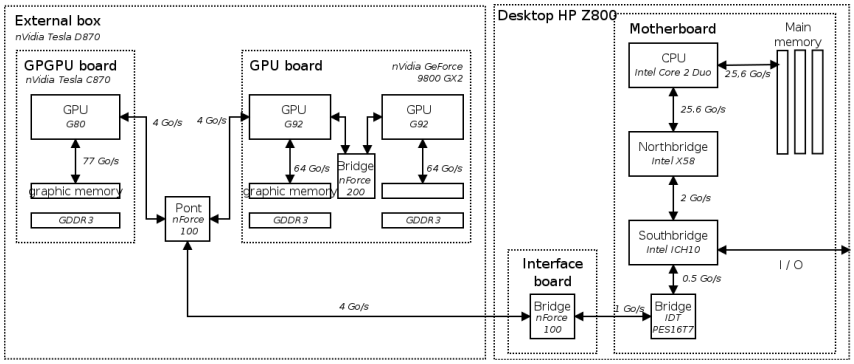


FIGURE 2 – Description de la plateforme de test.

Afin d'accélérer le processus de vieillissement des GPU, nous les avons fait fonctionner à des températures supérieures à leur température de fonctionnement habituelle. Certaines cartes utilisent une puce de contrôle de la température et du fonctionnement du ventilateur de type ONSEMI ADT7473 [21]. Pour ce type de carte, certains paramètres liés à la température sont modifiables par mise à jour du BIOS de la carte graphique à l'aide des outils d'*overcloking* (ex : nvflash, nibitor). Malheureusement, nous avons constaté lors de nos tests que ces modifications n'étaient pas disponibles pour l'ensemble des GPU testés. Pour celles dont nous avons réussit à en modifier le comportement, les puces, en utilisation intensive, n'atteignaient pas la plage de température visée (160-170 °C). Nous avons donc réalisé une enceinte thermique autour du GPU. Pour réguler la température, nous avons laissé le mécanisme de refroidissement passif et avons simplement modifié le fonctionnement du ventilateur. Les relevés de températures étaient effectués pas l'intermédiaire des sondes de températures internes et vérifiés avec un thermocouple P3400 placé au plus près du GPU.

4.2. Partie logicielle

Bien que la plateforme logicielle que nous avons implémentée permette de cibler n'importe quel périphérique compatible OpenCL, indépendamment du constructeur (NVIDIA, AMD, Intel, ...), nous nous sommes limités pour les tests aux architectures G80 de chez NVIDIA.

OpenCL permet de faire la distinction entre le processeur hôte (processeur central faisant office de chef d'orchestre) et les devices (CPU, GPU, ou autre) dont la mission est d'exécuter des noyaux de calcul intensif. Il est donc possible de distinguer l'application de tests écrite en C et tournant sur l'hôte, des micro-kernels écrits en OpenCL dont la vocation est de stresser un ou des composants en particulier. L'avantage de ce modèle de test est de rendre la partie vérification des résultats et contrôle peu intrusive, puisqu'elle sera réalisée par l'hôte.

Nous souhaitons nous intéresser principalement aux fautes intermittentes qui peuvent survenir lors des calculs effectués par le GPU. Le programme de test que nous avons mis en place se décompose en deux parties. Une partie exécutée par l'hôte et une partie exécutée par le GPU.

4.2.1. Partie hôte

Le programme hôte de notre protocole de test minimise les interférences entre l'exécution des programmes sur le device à tester et la vérification des résultats. Pour y parvenir, le programme hôte respecte la philosophie OpenCL. Ainsi on créé en premier un contexte d'exécution, auxquels on va associer le GPU à tester. Ensuite on créé une queue de commandes pour le GPU sélectionné. La mémoire est allouée sur le périphérique et les transferts des données d'entrées sont initiés. On charge les programmes

à exécuter sur le GPU (kernel), puis ils sont compilés sans aucune optimisation. On définit les arguments d'appel des kernels et l'environnement d'exécution (nombre de work-item et de work-group), on lance le kernel et enfin on transfert le résultat du GPU vers le CPU. Ce schéma d'exécution permet de :
– sélectionner le processeur à tester si plusieurs périphériques OpenCL sont présents sur la machine,
– fixer la durée du test,
– fixer le nombre d'itération à effectuer,
– sélectionner le type de kernel à exécuter,
– sélectionner les motifs d'exécution
– opérer un contrôle supplémentaire par rapport à la température de fonctionnement maximum.
Le contrôle supplémentaire de la température calibre un temps de latence entre chaque exécution afin que ces périodes d'inactivité entre kernels laissent le temps au processeur graphique de dissiper de l'énergie.
Les cibles architecturales sont composées de plusieurs unités de calcul. Dans le modèle choisi, chaque unité se comporte comme une ligne d'exécution indépendante exécutant les mêmes instructions. Le programme sélectionne automatiquement une configuration d'exécution dans laquelle chaque coeur de calcul exécute au moins un thread. Ce modèle permet ainsi d'isoler la ligne à l'origine de l'erreur en cas de défault.
Afin de minimiser les transferts de données, nous avons choisi de transférer au début du test un jeu de données initial pour chaque kernel à tester. Ensuite, une phase d'initialisation est lancée. Celle-ci se compose d'une exécution "à vide" pour chaque kernel afin de produire sur le périphérique le résultat attendu pour chaque coeur de calcul. Après l'initialisation, la phase de test est lancée. Si aucune limite de temps n'a été spécifiée, celle-ci consiste en une boucle sans fin composée d'appels successifs aux kernels de test sur les jeux de données placées initialement en mémoire. A l'issue de chaque exécution, les résultats produits sont comparés avec les résultats de références (ceux qui ont été générés lors de la phase d'initialisation). Si une erreur apparaît, alors celle-ci est détectée et signalée à l'hôte qui se charge de rapporter la ligne correspondant à l'erreur et sa date d'apparition.

4.2.2. Partie device

Nous avons créé un benchmark synthétique composé de plusieurs kernels. Parmi les spécificités architecturales testées, nous avons inclu un test relatif aux unités de calcul flottantes (*MADKernel*), un pour les bancs de registres (*REGKernel*) et un test de la mémoire partagée (*SHMKernel*). Chacun de ces kernels se compose d'une série d'instructions répétées un nombre de fois paramétrable. Les instructions et les données d'entrées sont choisies de façon à ce qu'une erreur intermédiaire puisse se propager jusqu'au résultat final. Seuls 3 kernels de test ont été écrits, mais nous avons voulu le modèle évolutif. Il est donc possible d'ajouter d'autres kernels de test respectant le modèle d'exécution (un jeu de donnée produit de façon déterministe un jeu de résultat en sortie).
A titre d'exemple, le kernel permettant de tester la mémoire partagée est décrit dans le Listing 1. Nous pouvons voir que pour ce kernel, différents motifs d'écriture/lecture sont tour à tour testés. Si une erreur se produit, alors celle-ci est consignée dans le tableau vecA, en indiquant la position de l'erreur. Enfin, nous remarquons que ce code n'utilise pas le résultat de référence vecB, puisque seuls les motifs présents dans le tableau pattern sont testés.

Listing 1 – Kernel de test de la mémoire partagée

```
#define NBPATTERN 4
__constant    uint pattern[NBPATTERN]={0xFFFFFFFF, 0xAAAAAAAA, 0x55555555, 0x00000000};

__kernel void ShMemKernel(__global   int      * vecA,     // Is there an Error and where
                          __global   float * vecB,        // Reference result
                          __global   float * vecC,        // Input Number
                          __global   float * vecD,        // Input Number
                          const uint nbiter)
{
    uint gid   = get_global_id(0);
    uint lid   = get_local_id(0);
    uint lsize = get_local_size(0);
    int i, j, k;
    __local int localBuffer[SIZE_OF_SHARED_MEM];
```

```
    vecA[gid] = 0;

    for(i=0; i<nbiter; i++){
        for(k=0; k < NBPATTERN; k++){
            // 1: Write the value to the shared memory
            for(j=lid; j<SIZE_OF_SHARED_MEM; j+=lsize)
                localBuffer[j]=pattern[k];

            // 2: Read the value from shared memory
            for(j=lid; j<SIZE_OF_SHARED_MEM; j+=lsize){
                if(localBuffer[j] != pattern[k])
                    vecA[gid] = j;
            }
        }
    }
}
```

5. Résultats expérimentaux

Pendant 3 mois, nous avons réalisé des tests sur différentes cartes graphiques afin d'en analyser le comportement lorsque celles-ci sont soumises à un stress interne (micro-kernel de test) et à un stress externe (fortes températures). Nous avons dans un premier temps étudié les différents mécanismes de protections thermiques de cartes NVIDIA de différentes générations. A partir de ces tests nous avons constaté que seules les cartes de type C870 et 9800GX2 pouvaient atteindre des températures de fonctionnement de l'ordre de 130 ˚C sans modifications matérielles pour inhiber le Thermal Shutdown Protection (TSP). Nous avons donc fait fonctionner ces exemplaires à 130 ˚C pendant 1 mois et n'avons constaté aucune erreurs. Ensuite, nos efforts se sont concentrés sur les cartes C870 qui étaient les seules cartes testées pouvant fonctionner à des températures supérieures à 160 ˚C sans activer le TSP.

5.1. Adaptation de la fréquence

Parmi les différents mécanismes de protection thermique, les constructeurs intègrent dans les cartes graphiques un mécanisme visant à réduire la fréquence de fonctionnement des coeurs de calcul lorsqu'une certaine température est atteinte [19]. Afin de caractériser le comportement de ce mécanisme nous avons soumis les cartes graphiques à différentes températures sans altérer le firmware d'origine. Les résultats sont consignés dans le tableau 1, qui contient pour chacune des cartes testées les températures à partir desquels la fréquence de fonctionnement est baissée et dans quelle proportion, ainsi que la température de TSP à partir de laquelle la carte graphique est désactivée.

A partir des GF11x, un mécanisme logiciel intervient en complément des autres mécanismes pour diminuer la fréquence de fonctionnement lors de l'exécution de certains logiciels comme FurMark, MSI Kombustor, ou OCCT qui peuvent conduire le GPU à surchauffer. Ce mécanisme est basé uniquement sur le nom de l'exécutable et un simple changement de nom permet de le contourner. A partir des GTX570/580, un mécanisme additionel d'ajustement des performances a été ajouté. Il surveille la puissance et le voltage sur chaque rails 12V de la carte afin que ces paramètres restent dans les spécifications de la carte.

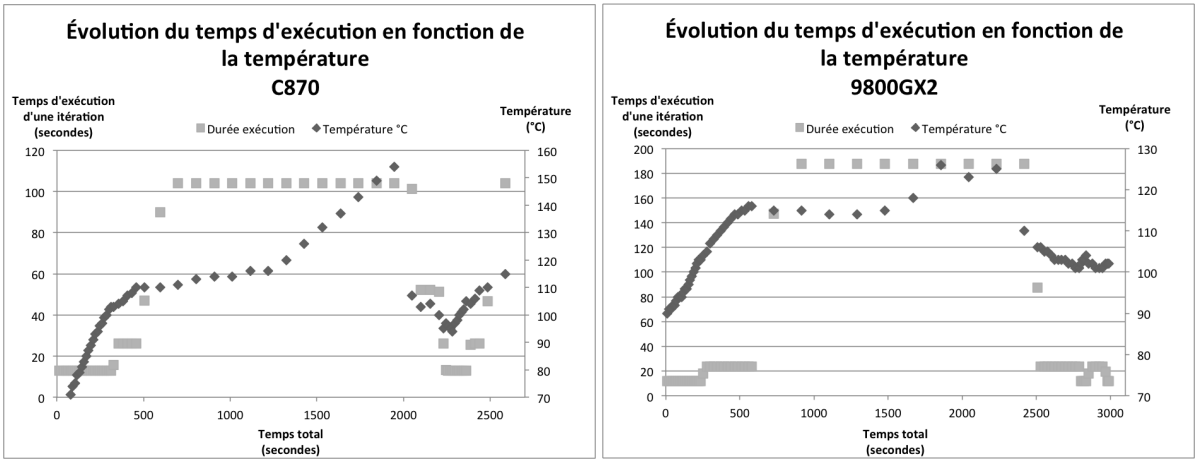
Les graphiques de la figure 3 représentent l'évolution de la température en fonction du temps pour l'exécution du kernel MAD en boucle infinie pour respectivement les cartes C870 et 9800GX2. Nous observons sur ces graphiques, qu'il existe différents paliers de fonctionnement des coeurs de calcul. En particulier nous avons constaté que les cartes C870 avaient 3 paliers de fonctionnements visant chacun à réduire à fréquence de fonctionnement par deux. A la vue de ces graphiques nous constatons que contrairement au TSP qui désactive la carte et nécessite un redémarrage physique de la machine, l'adaptation de la fréquence est réversible. En effet, lorsque la température redescend en dessous du seuil d'activation, la fréquence d'origine est rétablie sans avoir à redémarrer la machine.

Nous n'avons pas été en mesure d'observer plus d'un pallier sur les autres cartes, la différence entre la température d'abaissement de la fréquence et celle d'activation du mécanisme de protection thermique n'étant pas suffisante (différence de 5 ˚C, voire de l'ordre de 1 ˚C pour la GTX560). Dans ces conditions, on peut s'interroger sur l'intérêt d'intégrer un mécanisme d'abaissement de la fréquence sur les cartes

TABLE 1 – Caractéristiques des mécanismes de protection thermique observées sur différentes générations de cartes graphiques.

GPU	Fabriquant	Fab. Process (nm)	Trottling (˚C : Mhz)	Thermal Shutdown Protection (TSP) (˚C)
G80	NVIDIA C870	90	*105/110* : 600/300/150/75	-
G92	NVIDIA GeForce 9800GX2	65	*105/115* : 600/300/37	130
GT200	NVIDIA T10P	65	*105* : 900/475	115
GF100	NVIDIA GeForce GTX480	40	*100* : 700/350	110
GF114	ASUS GeForce GTX560	40	*100* : 810/405	105
GK104	PNY GeForce GTX670	28	*100* : 915/457	100

actuelles.



(a) C870 (b) 9800GX2

FIGURE 3 – Évolution du temps d'exécution d'une itération par rapport à la température au cours du temps

5.2. Test de fiabilité

Nous avons constaté que seules les cartes 9800GX2 et C870 pouvaient monter à 130 ˚C sans modifications matérielles. Nous avons donc réalisé une première série de test à cette température. Pour cela nous avons fait fonctionner en boucle l'ensemble des trois tests décrits en section 4.2.2 pendant trois semaines sur deux exemplaires simultanément de chaque carte. A l'issue de cette période, aucune erreur n'est apparue.

Nous avons ensuite fait fonctionner deux exemplaires de carte C870 à des températures de l'ordre de 160 ˚C. Nous avons constaté qu'il était difficile de stabiliser la température précisément, cette dernière variant de $+/-5$ ˚C . Les tests ont fonctionné pendant une période de 15 jours. Une carte s'est révélée défaillante après une période de 11 jours sans qu'aucune erreur n'apparaisse. La défaillance s'est révélée permanente même après redémarrage de la machine.

Nous avons ensuite légèrement augmenté la température de fonctionnement de deux cartes C870 pour les porter à une température moyenne de 170 ˚C. Nous avons fait fonctionner ces cartes pendant 70 minutes et reporté le type d'erreur et leur fréquences d'apparition dans les graphiques 4 et 5.

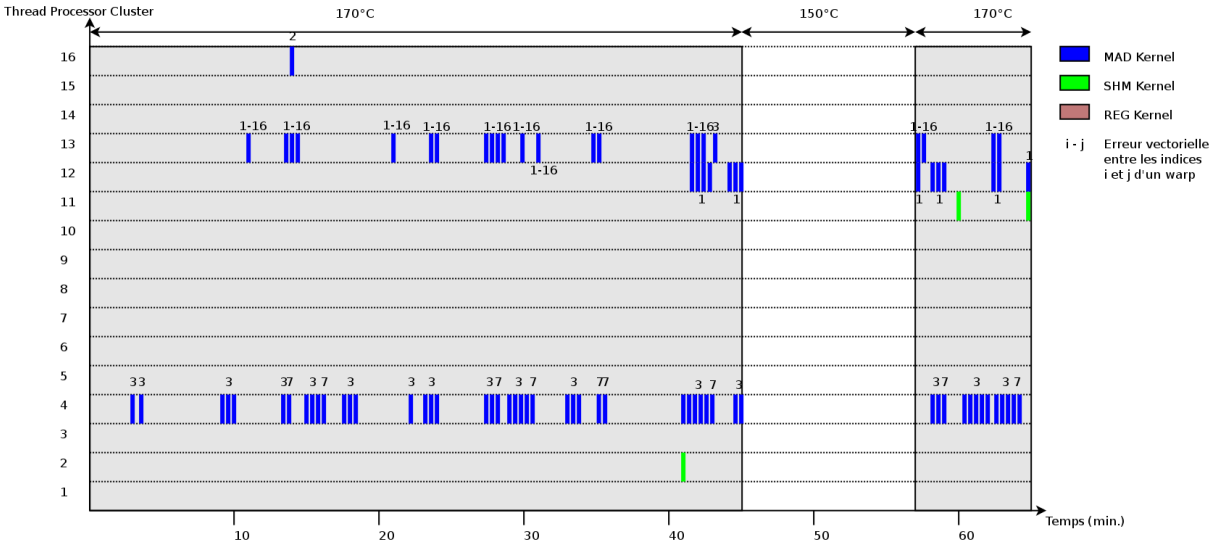


FIGURE 4 – Localisation des erreurs en fonction du temps et de la température sur le GPU C870 N °1.

Sur aucune des deux cartes, nous n'avons constaté de dysfonctionnement au niveau du banc de registres. Presque toutes les erreurs sont apparues lors de l'exécution du kernel MAD. Nous avons observé deux types d'erreurs : des erreurs vectorielles et des erreurs scalaires. Les erreurs vectorielles concernaient toujours les résultats produits par le premier demi-warp (résultats 1 à 16). Ce type d'erreur suppose une erreur au niveau de l'exécution de l'instruction, ou du banc de registres. Cependant le test du banc de registres ne montrant pas d'erreur, ces erreurs résultent plus probablement de l'exécution de l'instruction. Les erreurs scalaires se caractérisent par une erreur ponctuelle impliquant toujours le même PE d'un TPC. Ces erreurs survenaient dans la majorité des cas en mode *burst*, c'est-à-dire sur des résultats rapprochés dans le temps et produits par le même PE.
Nous constatons également, qu'à 170 °C les premières erreurs sont apparues après un certains temps et ont immédiatement cessé lorsque la température est redescendue à 150 °C. En revanche, elles sont immédiatement réapparues lorsque la température à été rétablie à 170 °C. Ce qui implique que la dégénérescence du circuit, une fois opérée, est irréversible. Nous avons constaté que, d'autre part, aussi légères soient-elles, les variations de températures semblaient avoir un impact sur le nombre et le type d'erreurs.

6. Conclusion et travaux futurs

Les processeurs actuels sont de plus en plus sujet aux erreurs matérielles et en particulier aux erreurs intermittentes. Un des facteurs clés dans l'apparition de ces erreurs est la température de fonctionnement du processeur. Les GPU n'échappent pas à la règle. Nous avons montré à travers nos tests qu'en l'espace de 6 ans (entre l'architecture G80 et le GK104), les constructeurs conscients de ce problème ont, au fil des générations, abaissé la température d'activation du TSP pour la ramener à la température d'activation de réduction de la fréquence de fonctionnement.
Nous avons proposé une plateforme de test générique, écrite en OpenCL, dont l'objectif est de tester si un GPU donné est sujet aux erreurs intermittentes. Au delà de la simple présence ou absence d'erreurs sur le périphérique, cette plateforme permet aussi d'isoler la partie du processeur incriminée. Cette information est essentielle si l'on souhaite isoler la partie d'un processeur sujet à ce type d'erreurs. En effet, nous avons constaté que l'on retrouve une hiérarchie dans l'apparition des erreurs intermittentes, caractéristique de l'organisation hiérarchique des calculs au sein des GPU. Ainsi, au lieu de considérer l'intégralité de la puce comme défectueuse, avec ce type de test, il sera possible d'isoler la partie du processeur qui s'avère défectueuse simplement en ne considérant pas les calculs qu'elle produit. Mais

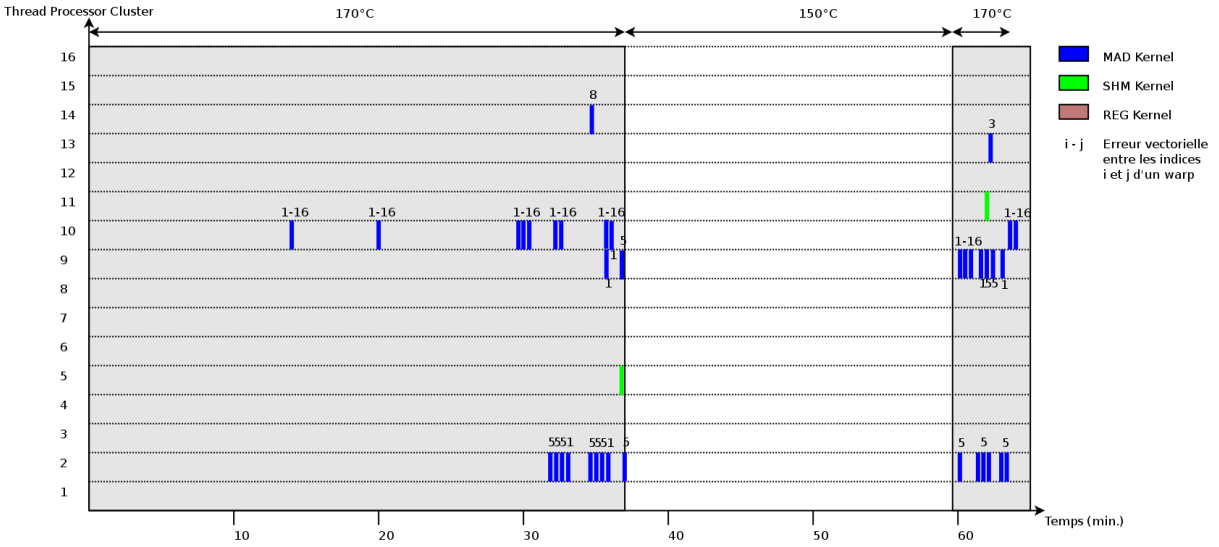


FIGURE 5 – Localisation des erreurs en fonction du temps et de la température sur le GPU C870 N °2.

avant de pouvoir atteindre ce niveau de détails, il nous faut compléter la bibliothèque de microkernels de la plateforme proposé, afin de tester d'autres parties du processeur et nous adapter aux autres architectures.

Bibliographie

1. Aharon (A.), Goodman (D.), Levinger (M.), Lichtenstein (Y.), Malka (Y.), Metzger (C.), Molcho (M.), Shurek (G.) et Metzger (Y. M. C.). – Test program generation for functional verification of powerpc processors in ibm, 1995.
2. Che (S.), Boyer (M.), Meng (J.), Tarjan (D.), Sheaffer (J. W.), Lee (S.-H.) et Skadron (K.). – Rodinia : A benchmark suite for heterogeneous computing. *In : Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*. pp. 44–54. – Washington, DC, USA, 2009.
3. Collange (S.), Daumas (M.), Defour (D.) et Parello (D.). – Barra : A parallel functional simulator for gpgpu. *In : Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. pp. 351–360. – Washington, DC, USA, 2010.
4. Danalis (A.), Marin (G.), McCurdy (C.), Meredith (J. S.), Roth (P. C.), Spafford (K.), Tipparaju (V.) et Vetter (J. S.). – The scalable heterogeneous computing (shoc) benchmark suite. *In : Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. pp. 63–74. – New York, NY, USA, 2010.
5. Davies (C. W.) et Kelleher (B. M.). – Apparatus, system, and method for managing aging of an integrated circuit . *Patent*, no10/882,140, June 29 2004.
6. Greskamp (B.), Sarangi (S. R.) et Torrellas (J.). – Threshold voltage variation effects on aging-related hard failure rates. *In : ISCAS*. pp. 1261–1264. – IEEE.
7. Guilhemsang (J.). – *Test en ligne pour la détection des fautes intermittentes dans les architectures multiprocesseurs embarquées*. – These, Université de Nice Sophia-Antipolis, avril 2011.
8. Guilhemsang (J.), Héron (O.), Ventroux (N.), Goncalves (O.) et Giulieri (A.). – Impact of the application activity on intermittent faults in embedded systems. *In : VTS*. pp. 191–196. – IEEE Computer Society.
9. Haque (I. S.) et Pande (V. S.). – Hard data on soft errors : A large-scale assessment of real-world error rates in gpgpu. *In : Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. pp. 691–696. – Washington, DC, USA, 2010.
10. Héron (O.), Guilhemsang (J.), Ventroux (N.) et Giulieri (A.). – Analysis of on-line self-testing policies

for real-time embedded multiprocessors in dsm technologies. *In : IOLTS*. pp. 49–55. – IEEE.
11. JEDEC. – Foundry process qualification guidelines. *JEDEC technical report*, 2004.
12. JEDEC. – Accelerated moisture resistance-unbiased autoclave. *JEDEC technical report*, 2008.
13. JEDEC. – Stress-test-driven qualification of integrated circuits. *JEDEC technical report*, 2011.
14. KHRONOS. – Opencl official website. *http ://www.khronos.org/opencl/*, 2012.
15. Koncaliev (D.). – Intel fdiv bug. *http ://www.cs.earlham.edu/ dusko/cs63/fdiv.html*, 1995.
16. Koncaliev (D.). – Intel fpu bug. *http ://www.cs.earlham.edu/ dusko/cs63/fpu.html*, 1997.
17. Kraus (J.) et Förster (M.). – Facing the multicore-challenge ii. chap. Efficient AMG on heterogeneous systems, pp. 133–146. – Berlin, Heidelberg, Springer-Verlag, 2012.
18. Lobeiras (J.), Amor (M.) et Doallo (R.). – Influence of memory access patterns to small-scale fft performance. *The Journal of Supercomputing*, pp. 1–12. – 10.1007/s11227-012-0807-5.
19. Mimberg (L.), Wagner (B.) et Lao (M.). – Method and system for dynamic power supply voltage adjustment for a semiconductor integrated circuit device . *Patent*, no10/078,292, February 15 2002.
20. Nightingale (E. B.), Douceur (J. R.) et Orgovan (V.). – Cycles, cells and platters : an empirical analysisof hardware failures on a million consumer pcs. *In : Proceedings of the sixth conference on Computer systems*. pp. 343–356. – New York, NY, USA, 2011.
21. ONSEMI. – *dBCOOL Remote Thermal Monitor and Fan Control*. – Rapport technique, http ://www.onsemi.com/pub_link/Collateral/ADT7473-D.PDF, ONSEMI, 2010.
22. Schroeder (B.), Pinheiro (E.) et Weber (W.-D.). – Dram errors in the wild : a large-scale field study. *Commun. ACM*, vol. 54, n2, 2011, pp. 100–107.
23. Sheaffer (J. W.), Luebke (D. P.) et Skadron (K.). – A hardware redundancy and recovery mechanism for reliable scientific computation on graphics processors. *In : Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*. pp. 55–64. – Aire-la-Ville, Switzerland, Switzerland, 2007.
24. Stratton (J.), Rodrigues (C.), Sung (I.), Obeid (N.), Chang (L.), Anssari (N.), Liu (G.) et Hwu (W.). – Parboil : A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing*, 2012.
25. Wagner (I.) et Bertacco (V.). – *Post-Silicon and Runtime Verification for Modern Processors*. – New York, USA, Springer, 2011.
26. Wong (H.), Papadopoulou (M.-M.), Sadooghi-Alvandi (M.) et Moshovos (A.). – Demystifying gpu microarchitecture through microbenchmarking. *In : ISPASS*. pp. 235–246. – IEEE Computer Society.
27. XILINX. – Device reliability report (ug116). *http ://www.xilinx.com/support/documentation/user_guides/ug116.pdf*, 2010.
28. Yang (X.), Liao (X.), Xu (W.), Song (J.), Hu (Q.), Su (J.), Xiao (L.), Lu (K.), Dou (Q.), Jiang (J.) et Yang (C.). – Th-1 : China's first petaflop supercomputer. *Frontiers of Computer Science in China*, vol. 4, 2010, pp. 445–455. – 10.1007/s11704-010-0383-x.
29. Yim (K. S.), Pham (C.), Saleheen (M.), Kalbarczyk (Z.) et Iyer (R.). – Hauberk : Lightweight silent data corruption error detector for gpgpu. *In : Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*. pp. 287–300. – Washington, DC, USA, 2011.