Impacting predictability of GPU's

David Defour

▶ **To cite this version:**

David Defour. Impacting predictability of GPU's. 2014. hal-00951920v2

HAL Id: hal-00951920
https://hal.science/hal-00951920v2

Submitted on 3 Mar 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Impacting predictability of GPU's

David Defour
Laboratoire DALI-LIRMM
52 avenue Paul Alduy
66860 Perpignan Cerdex - France
Email: david.defour@univ-perp.fr

***Abstract*—GPU's are massively multicore architectures managing several thousands of concurrent threads. This concurrence, maintained through several schedulers, is necessary to keep high performance but negatively impact predictability. In this work, we first propose measures of predictability as well as CUDA tests to estimate this measure regarding warp and block scheduler for architectures from G80 to GK104. Finally, we evaluate the impact of hardware reset, underclocking and heavy synchronization on predictability.**

I. Introduction

Efficient exploitation of modern multicore architectures relies on a hierarchical structuration of computation as well as execution concurrency. This impacts determinism and reproducibility making software development more tedious. For example, GPUs are managing several thousands of concurrent threads thanks to hardware resources such as warp and block schedulers. The design complexity of those processors is such that their behavior is mostly unknown and can be considered unpredictable. A workaround is to enforce interaction between tasks using hypothesis on execution order or synchronization mechanism such as lock, atomic or barrier.

Albeit processors behavior can be considered unpredictable, the concept of predictability is exploited at hardware level with branch, address or value predictors. However it is often manipulated under it's binary form and is useful only if the prediction is correct. However, there exist cases for which probabilistic information of predictability is valuable. Such information are essential to characterize processors behaviors to build processor's simulators [1], estimate WCET[2], or helpful to measure the pertinence of random number generator's based on indeterminism in the access of a shared resource [3].

Predictability of manycore and especially GPUs is well pictured by a known problem in numerical algorithm: the summation problem. This consists in computing the sum of several floating-point numbers as it appears for example in BLAS operations. As floating-point addition is not associative, the result is impacted by the order in which numbers are accumulated. To emphasis the impact of scheduling we can consider the following CUDA kernel which computes the sum *Res* located in global memory of N floating-point numbers stored in table *Input*.

```
__global__ void GlobalSum(float *input, int N,
                          float *res) {
  int gid =  blockDim.x*blockIdx.x+threadIdx.x;

  for(uint i=0; i<N; i+=GridDim.x*blockDim.x)
     res = atomicAdd(&res, input[i+gid]);
}
```

Listing 1. Global summation of floating-point numbers

The problem with this simple code is that we do not have any information on the order in which threads will acquire the access to the data *Res*. For example, on a set of $N = 2^{13}$ value with a condition number of 10^8, we measured that out of over 1000 runs with 1 blocks of 1024 threads on a GTX680 we have 1000 different results!

This article brings three contributions regarding predictability of GPUs. Firstly, we propose a measure of predictability of two important components of Nvidia GPUs involved in indeterminism: the scheduler of warp and block. Secondly, we use this measure to determine the impact of hardware generation and therefore scheduling technics over predictability. Thirdly, we estimate the impact of three technics available to the developer over the measure of predictability: Resetting, underclocking and synchronization. It is organized as follow. Section II presents works related to determinism and predictability. Then Section III describes characteristics of the considered GPU that may impact predictability. Section IV describes the measure of predictability we have used and some results on common Nvidia architectures. Then, in Section V, we describes how predictability on GPU can be improved. Finally, in Section VI we apply results regarding predictability to the summation problem.

II. Related Works

Many studies related to determinism have accompanied the development of multicore architectures. The

fundamental problem to ensure determinism of parallel programs is to check interactions between different instances of threads in shared memory. Solutions involve at least one of the 4 elements necessary for the execution of a parallel program: language, compiler, software or hardware.

Some work [4] focused on programming languages that let programs to be deterministic by construction. For example, the language SHIM [5] allows programs developed in message passing to have a deterministic behavior. This same property is made available for the class of functional languages like NESL [6] or Data Parallel Haskell [7].

Some approaches are based on the compiler to ensure determinism, transforming a sequential program (with or without annotation) in a parallel version where determinism is preserved [8], [9]. These solutions are similar to software solutions that test, at the execution, violation of determinism [10], [11].

Hardware based solutions provide deterministic execution for a small additional cost in terms of execution time, but at the cost of an increased hardware complexity. These solutions reduce the possibility of scheduling such that the observed scheduling remains identical [12], [13], [14]. It is commonly believed that this type of solution simplifies testing, debugging, replication and record-replay of multithreaded programs.

Indeterminacy on GPU has also been explored from both approaches: software and hardware. GPU's natively integrate a number of synchronization mechanisms (inter or intra-multiprocessor). These mechanisms limit the indeterminacy in the access to a shared resource, but they can be expensive [15], [16] and do not enforce an execution order. Various extensions have been proposed to improve the determinism. For example, TIMEGRAPH [17] considers the problem of tasks scheduling on GPU in a multitasking environment. KiloTM [18] uses transactional memory, while Boyer and Skadron [19] propose a GPU emulator to detect "race condition" and bank conflicts in CUDA programs. PUG [20] and GPUVerify [21] focus on the detection of " race condition" by a static analysis. GPUDet [22] proposes to use the specific GPU characteristics such as the z-buffer and SIMT execution to remove the indeterminacy. [23] proposes a method for the evaluation of the determinism by the mean of a signature calculation based on a hash function.

While these works have focused on improving the determinism of GPU, some have attempted to a better understanding of GPU architectures using microbenchmarking [24]. But to our knowledge none of them have addressed the issue of measuring and improving the predictability related to GPU schedulers.

III. GPU's

In this article we consider Nvidia GPUs of different generations whose characteristics are described in Table I. These coprocessors are used to support CPU for the execution of tasks exhibiting data parallelism. These tasks are divided into thread operating in SIMT mode and executed by specific hardware. We must distinguish the software and hardware organization of threads.

In CUDA terminology, GPUs consist of CUDA cores also called Streaming Processors (SP) organized hierarchically. These CUDA cores are grouped in multiprocessor (MP). The number of CUDA multiprocessor varies depending on the version of CUDA capability and the type of GPU. Similarly multiprocessors are grouped to form the *Graphics Processing Cluster* (GPC). An additional and transparent level of grouping is introduced at the hardware level: the warp. These warps, corresponding to 32 threads, are created, managed, launched and executed by SIMT units. Multiprocessors maintain a scoreboard for each warp to launch warps depending on their type, instructions and a "fair" scheduling policy.

1) Sources of nondeterminism: The fact that GPUs are coprocessors isolate them from sources of indeterminism such as interrupts or context switches found in the case of general-purpose processors. However, future generations may be more sensitive to these sources, if those architectures will become similar to CPU.

GPUs have multiple clock domains, each operating at an optimal frequency. Synchronization circuits between these areas can introduce indeterminism resulting from delays caused by phase changes when messages are exchanged [25]. The weight of this source is expected to increase in future generations as dynamic voltage and frequency scaling (DVFS) should be generalized to the granularity of the multiprocessor.

Another source of indeterminism is related to the time to access off-chip memory as it depends on the physical location (memory partition) where the data is placed [26]. On modern GPUs, the probability that the memory allocation and placement are the same between two consecutive calls is very low. In addition, variations in DRAM cells prompt the use of techniques such as dynamic refreshing introducing variable access time to DRAM cell [27].

With transistor's miniaturization, the occurrence of hardware's failure becomes more common. As these faults are corrected in some cases by mechanisms introducing additional delays in memory access, this is another source of non-determinism [28].

Finally, the use of uninitialized scheduler introduced indeterminism by changing the order of execution and potentially assignments between software and hardware. This corresponds to warp schedulers in each computing

TABLE I. DESCRIPTION OF NVIDIA GPU'S OF DIFFERENT GENERATIONS.

GPU	Arch.	CUDA Cap.	#MP/ GPC	#MP	CUDA Core / MP	Warp Scheduler / MP	GPU Clock (Mhz)	Memory Clock (Mhz)
C870	G80	1.0	2	16	8	1	1350	800
9800 GX	G92	1.1	2	16	8	1	1500	1000
GTX 480	GF100	2.0	4	15	32	2	1401	1848
GTX 560	GF114	2.1	4	7	48	2	1620	2004
GTX 680	K10	3.0	3	8	192	4	1059	3004

unit, workgroup schedulers, and bus arbiters in the interconnection networks. Although these elements are initialized when processors are placed under tension, they are not between each kernel launch. The states of these units are dependent on previous launched kernels, and can therefore be considered unpredictable.

2) Schedulers: At software level, a given task is hierarchically divided into a grid of thread blocks. These elements are then handled by the hardware to be launched virtually in parallel. This is done by schedulers whose characteristics depend on the generation of the GPU considered, according to their version of *compute capability* [29]. This classification varies from 1.x to 3.x, for the latest architectures available in 2013.

Architectures with compute capability 1.x, have a single warp scheduler per multiprocessor that can launch an integer or floating-point instruction every 4 clock cycles.

Architectures with compute capability 2.x, have two warp schedulers per multiprocessor. More specifically, for architectures with compute capability 2.0, each of the two schedulers may initiate an instruction for a ready warp, while for compute capability 2.1, each scheduler can initiate up to 2 independent instructions. In this case, the first scheduler is responsible for warps with an odd ID, and the second warps with a even ID, except when it is a double precision floating-point instruction. A warp scheduler launches an instruction only on half of CUDA cores. If a non-atomic instruction executed by a warp attempt to write at the same location in global memory, only one thread performs a write and which thread does it is undefined.

Architectures with compute capability 3.x, have four warp schedulers per multiprocessor. When a warp is ready to execute, it is first distributed between one of four warp schedulers. Then, at every instruction issue time, each scheduler launches two independent instructions for its assigned warp.

IV. MEASURE OF PREDICTABILITY

Predictability in a parallel execution model is the ability to predict the behavior or the results of an execution. However these notions are not quantitative but qualitative. Indeed, it is difficult to give a measure of how far a result or behavior is from a prediction, it is either right or wrong. However, if we make observations over several runs, it is possible to collect statistical information's on the space of possible results associated with probabilities of occurrences.

Predictability may be associated with various kinds of observations. It can be an execution time, intermediate or final results. If observations done on the execution time are the easiest to achieve, there are the less accurate as it only characterizes a global behavior and doesn't take into account execution artifacts. The same remark holds with observations of the final result. It is therefore pertinent to analyze intermediate states associated with execution.

We saw in section III-1 that there are several possible sources of indeterminism that may affect results or observable behavior for the execution of a program on a GPU. In this work, we focus on the scheduling of block and warp to quantify their impact on the space of possible results. We therefore propose to use two class of observation to compare execution among them.

A first measure is to read the clock register for each executed warp. The relative order of execution is obtained by comparing the execution time between them. In the remainder of this work we will call this method *clock*. It gives an accurate information on the scheduling of warp within a MP as the clock register will be accessed in read only and will be increased at each clock cycle by the hardware. This measure can be subject to criticisms to analyze block scheduling as each MP has its own clock register. However, even though these registers are independent there are incremented at the same pace corresponding to the clock frequency.

In order to complete the first measure, we proposed a second one based on the access to a global value shared by all blocks and enforce unicity of accesses with atomic instructions. We will refer to this solution by *atomic*. It should be noticed that observations made with this technic might includes artifacts from bus arbiter as well.

A. Tests

We have implemented in CUDA these two tests. They generate execution vectors that we will compare among them. These vectors consist of as many values as threads/warps/blocks executed. Values define an execution order corresponding to either the date to which threads had access to the clock register for the *clock* method (Listings 2), or the order in which they had access to the shared variable via the atomic instruction for the *atomic* method (Listings 3).

For each run, we get N execution vector with $N = 1$ for block scheduling and N equal to the number of block per grid launched for warp scheduling. The code is executed C times such that we have $N.C$ execution vectors to compare. We will use the statistical mode corresponding to the frequency of occurrence of the most frequent vector. For example, if over ten execution vectors, we get the following set of observations $\{x, y, z, w, x, x, y, x, z, t\}$, then predictability corresponds to 40% as vector x appears four times over ten.

Another interesting measure is the cardinality of the set. Each vector consists of the access order to a shared element (clock or global counter). If each vector consists of p elements, we potentially have $p!$ possible arrangements. Then, we can compare the number of different vectors over the $N.C$ observations to the $p!$ possible arrangements.

```
__global__ void TestClock(int* dMem){
  int gid =  blockDim.x*blockIdx.x+threadIdx.x;

  dMem[gid] = clock();
}
```

Listing 2. Test of execution order using clock register

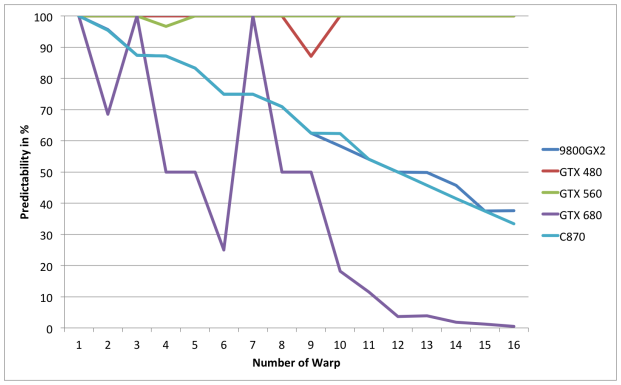
```
__global__ void TestAtomic(int* dMem, int* v){
  int gid =  blockDim.x*blockIdx.x+threadIdx.x;
  int order;

  order = atomicAdd(v, 1);
  dMem[gid] = order;
}
```

Listing 3. Test of execution order using a global variable

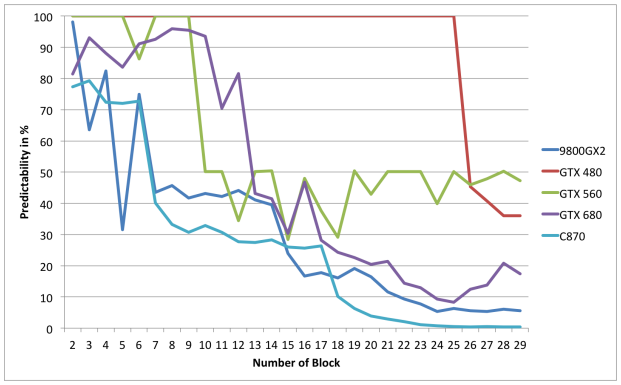
B. Results

We have tested *clock* and *atomic* benchmark on Nvidia graphic cards on a Windows 7 System with Driver 311.06. *atomic* test were not launched on C970 as this feature is not supported on this device.

1) Statistical mode: Figure 1 et 2 describe predictability using statistical mode for respectively *clock* and *atomic* test. Scheduling of blocks was tested with 1 to 32 blocks with a single warp in order to focus to



(a) warp

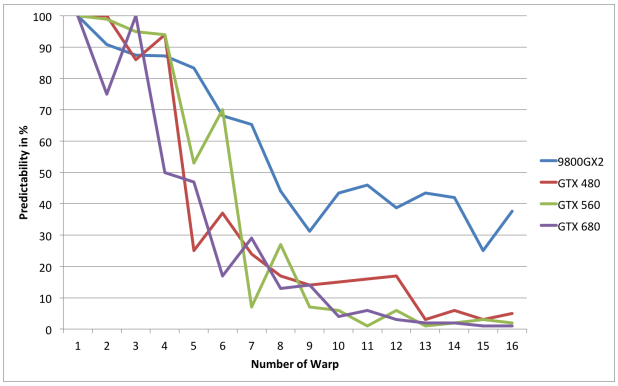


(b) block

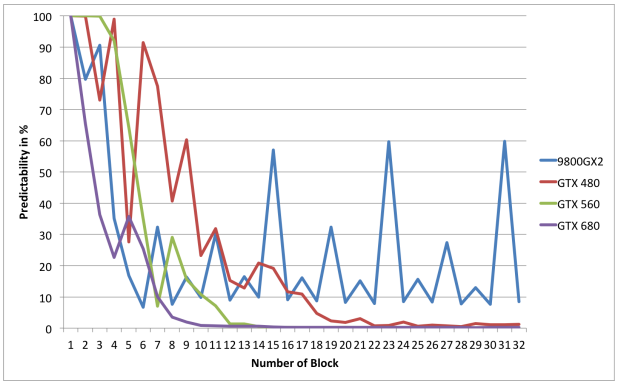
Fig. 1. Statistical mode based on *clock* test

block scheduling. Scheduling of warps were tested using a number of block equal to the number of available multiprocessor on the tested architecture such that each multiprocessor has only warps of one block to schedule with 32 to 512 or 1024 threads in each block according to the capability of the architecture. Every test was launched 1000 consecutive times without rebooting the computer or reinitializing devices. We should mention that tests were also conduced with different drivers and system (Linux) providing similar results. More data are available in the annexe of this report.

One can observed on Figure 1(a) that warp scheduler is very predictable for the GTX480 and GTX560 as the statistical mode is close to 100%. Warp scheduler of C870 and 9800GTX exhibit similar behavior that decreases linearly with the number of launched warp. This is due to the similar configuration of MP between both architectures (Table I). GTX680 exhibits a more chaotic predictability, which could be due to its dynamic adaptability of voltage and clock frequency. Therefore we can observe that warp predictability is strongly linked with compute capability and that architecture which exhibit the highest predictability regarding *clock*



(a) warp



(b) block

Fig. 2. Statistical mode based on *atomic* test

test are those of compute capability 2.x.

This behavior is significantly different when warps try to access a shared resource in global memory (Figure 2(a)). We can notice that for 32 threads, every tested architecture exhibits a predictability of 100%. In addition, the GTX480, GTX560 and the GTX680 have similar behavior which is less predictable regarding atomic access than accessing the clock counter. The predictability is falling bellow 10% from 13 warps launched in these cases. Conversely, 9800GX2 continues to exhibit a predictable behavior greater than 40% and up to the maximum of 16 warps per multiprocessor launched.

Figure 1(b) informs us on the order in which block had access to the clock register. We can observe that the behavior of the block scheduler is different for the two architectures with compute capability 1.x. The same behavior was previously observed in [30] using power consumption measure. GTX480 and GTX560 remain the two architectures with the highest predictability, which is equal to 100% up to 25 blocks launched for the GTX480 and close to 50% for the GTX560.

Figure 2(b) presents *atomic* tests for block scheduling. We can notice that for every tested architectures, predictability falls bellow 20% from 12 blocks launched. Again block scheduling of the 9800GX2 exhibits distinctive z-chart shape predictability. Thus on this architecture, a number of multiple block of eight plus the number of multiprocessor minus one is the configuration that exhibits the highest predictability (60 %).

2) Cardinality: In order to evaluate the space of possible executions, we collected 10^7 execution vectors whatsoever for block or warp scheduling.

Figure 3(a) shows on a logarithmic scale the cardinality of the set of observed execution vectors for a number of block between 2 and 32. Vectors correspond to a measurement based on *atomic* test with a single warp per block in order to focus on block scheduling. We can observe that the cardinality for the 9800GX2 is smaller by 3 orders of magnitude than the one of newer cards as well as the same z-chart shape as in Figure 2(b). Then comes the GTX480 which is an order of magnitude less than the GTX560 and GTX680 's . It should be noticed that the number of possible configuration for b block launched is $b!$, which is 2.10^{35} for $b = 32$, 2.10^{18} for $b = 20$, and 2.10^{6} for $b = 10$. Therefore, eventhough cardinality is high, observed scheduling configuration remains bellow these bounds. This means that only a small subset of possible scheduling are actually possible.

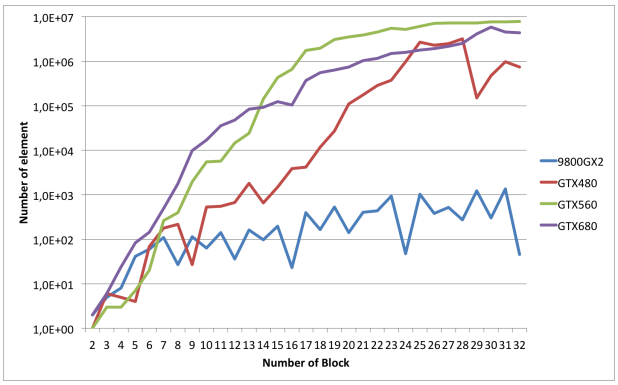
Figure 3(b) shows on a logarithmic scale the cardinality of the set of observed execution vectors for a number of warp between 1 and 16. Vectors correspond to a measurement based on *clock* test. We can notice that the number of observed combination is very low for the scheduling of warp and is consistent with the measurements made for the statistical mode (Figure 1(a)).

V. Impacting predictability

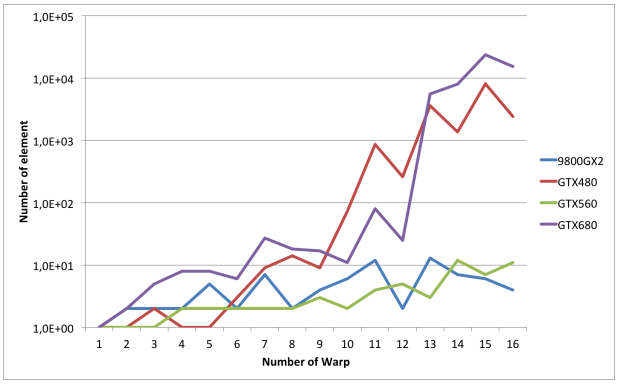
We have measured the impact of 3 technics over warp and block scheduling with detailed results given in the annexe.

The first technic consists in resetting the coprocessor by explicitly destroying and cleaning up resources associated with the device (*cudaDeviceReset*) before any call to a kernel. The expected effect is to reset internal states before starting work and making them identical before the execution.

The second technic consists in adding synchronization barrier at the beginning of the kernel to force warp and instruction scheduler to be in a stable state before dispatching work. This way, we make sure that every instructions and data are available for every scheduler before they really schedule work.



(a) block



(b) warp

Fig. 3. Number of different observed execution vector over 10^7 execution.

The third technics consists in reducing the speed of the GPU as well as GPU memory in order to minimize the effect of synchronization between clock domains. We used an overclocking tool (*NVIDIAInspector*) to modify all the clock frequency. We observed that:

a) Adding synchronization barrier: before starting the execution by inserting the assembly instruction *bar.sync* only brings benefits for GPU of cuda capability 1.x.

b) cudaDeviceReset: is a technics that work for any architectures for both *clock* and *atomics* test, and for warp and block scheduling. For example, this technics has improved by a factor 33 the measure of warp scheduling based on *clock* and 12.4 for the one based on *atomic* for the GTX680. However it should be noticed that the cost of this function represent between 2 and 17 extra seconds depending on the hardware.

c) Underclocking: has brought improvment mainly regarding scheduling of blocks on the GTX680 for the *atomics* test, when combined with cudaDeviceReset.

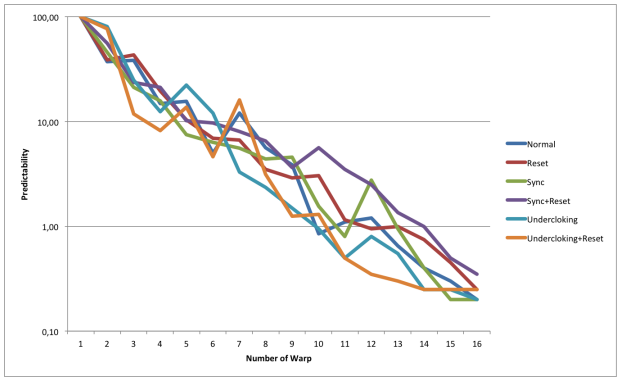


Fig. 4. Predictability for the summation problem on a GTX480 with 1024 elements.

However, it should be noted that there is no clear trend shared by each tested architecture of mixing those three technics.

VI. Application: Summation problem

As we mentioned in the introduction, warp and block schedulers impact the order in which threads are executed. To enforce atomics access to a shared ressources, atomic operations are available on devices starting from compute capability 1.1. On devices above 2.x, atomic floating-point addition in single precision has been added. This instruction can be used when threads are producing floating-point results which have to be accumulated. This problem identified as the summation problem, is known to be a difficult problem due to the non associativity of floating-point addition.

To illustrate the impact of predictibility on this problem we have executed the program in 1 on the GTX680 on a set of 1024 floating-point numbers having a condition number of 10^{11}. We executed this program 1000 times with one block and a number of threads ranging from 32 to 512. Figure 4 gives results regarding predictability with various optimization (reseting the device, undercloking, synchronization). We can observe that predictability for the summation problem is quickly decreasing as the number of warp is increasing. On this problem, none of the previously mentioned solution brings a clear advantage over the normal execution.

We noticed in figure 2(a) that the scheduling of atomic instructions is predictable at 100% up to 32 threads for any tested architectures. This means that atomic instructions within a warp are always handled in the same order. Thanks to this property, it is possible to design a summation kernel based on atomic predictable at 100% on any architecture. It consists in ensuring that only threads of a warp can access a shared resource at a given time. In our case, this is achieved by using a multilevel atomic reduction scheme at the granularity of

the warp avoiding inter-warp or inter-block exchange as described in Listings 4.

```
__global__ void PredSum(float *input, int N,
                        float *res) {
  int gid =  blockDim.x*blockIdx.x+threadIdx.x;
  __shared__ float sacc[32];

  sacc[threadIdx.x%32]=0;
  __syncthreads();

  for(uint i=0; i<N; i+=GridDim.x*blockDim.x)
    atomicAdd(&sacc[threadIdx.x/32],
              input[i+gid]);

  __syncthreads();
  if (threadIdx.x<32)
     atomicAdd(&res[blockIdx.x],
               &sacc[threadIdx.x/32]);
}
```

Listing 4. Predictable global summation of floating-point numbers

We have tested this solution on each architecture and confirmed that it gives reproducible result in every case. However, this workaround is indeed not a good solution, as we are using more atomics variables and inter-block synchronization. We believe that providing single precision atomic floating-point addition is bad idea as it make reproducibility more difficult to ensure, which could potentially leads to dangerous deadlock [31]. We advocate that such operation should be removed from the ISA and encourage users to use fixed reduction scheme or bit-accurate solution such as [32] instead. Another solution, is to invite manufacturer to offer a predictible execution mode in which initial state will be reset and schedulers will have a fixed scheduling strategy.

VII. Conclusion

In this article we looked at a probabilistic measure of predictability regarding warp and block schedulers. The measure is based on a signature embedding the order of thread's access to a shared variable. This variable can be in read only mode such as the clock counter, or a shared variable in read/write access with a global visibility and accessed with atomic instructions. The major advantage of collecting signature under this format is that in future work we should be able to take into account temporal information in our model that could leads to a Hidden Markov Model that will let us estimate possible executions with probability of occurrence.

We have provided measures of predictability for several Nvidia GPUs and compared their respective behavior. We have shown for example, that the warp scheduler of the GTX480 and GTX560 is predictable at almost 100% when accessing the clock register. In addition we have shown that predictability can be marginally impacted using resetting, underclocking or synchronization. The impact is dependent of the considered test and architecture.

We have applied those tests on the summation problem based on floating-point atomic addition. The high level of concurrency of accesses to a shared variable make the results of this instruction highly unpredictable and potentially dangerous if not manipulated cautiously. Finally, we showed how to exploit intra-warp predictability for atomic accesses to achieve a reproducible summation on any NVidia architecture,

References

[1] S. Collange, M. Daumas, D. Defour, and D. Parello, "Barra: A parallel functional simulator for gpgpu," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, 2010, pp. 351–360.

[2] B. Lisper, "Towards parallel programming models for predictability," in *12th International Workshop on Worst-Case Execution Time Analysis, WCET 2012, July 10, 2012, Pisa, Italy*, ser. OASICS, T. Vardanega, Ed., vol. 23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012, pp. 48–58. [Online]. Available: http://dx.doi.org/10.4230/OASIcs.WCET.2012.48

[3] J. Chan, B. Sharma, J. Lv, G. Thomas, R. Thulasiram, and P. Thulasiraman, "True random number generator using gpus and histogram equalization techniques," in *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, 2011, pp. 161–170.

[4] R. Bocchino, V. Adve, S. Adve, and M. Snir, "Parallel programming must be deterministic by default," in *Proc. HotPar '09 (1st USENIX Workship on Hot Topics in Parallelism), USB Data Stick*. Berkeley, CA: Usenix Assoc., Mar. 2009, uIUC.

[5] S. A. Edwards and O. Tardieu, "SHIM: a deterministic model for heterogeneous embedded systems," *IEEE Trans. VLSI Syst*, vol. 14, no. 8, pp. 854–867, 2006. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TVLSI.2006.878473

[6] G. E. Blelloch, "NESL," in *Encyclopedia of Parallel Computing*, D. A. Padua, Ed. Springer, 2011, pp. 1278–1283. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-09766-4

[7] J. M. D. Hill, "Data parallel haskell: Mixing old and new glue," QMW CS, Tech. Rep. 611, Dec. 1992. [Online]. Available: ftp://ftp.dcs.qmw.ac.uk/pub/cpc/jon_hill/dpGlue.ps.Z

[8] M. J. Bridges, N. Vachharajani, Y. Zhang, T. B. Jablin, and D. I. August, "Revisiting the sequential programming model for the multicore era," *IEEE Micro*, vol. 28, no. 1, pp. 12–20, 2008. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/MM.2008.13

[9] R. Allen and K. Kennedy, *Optimizing Compilers for Modern Architectures: A Dependence-Based Approach*. Morgan Kaufmann, 2002.

[10] M. D. Allen, S. Sridharan, and G. S. Sohi, "Serialization sets: a dynamic dependence-based parallel execution model," in *Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, (14th PPOPP'09)*. Raleigh, NC, USA: ACM, Feb. 2009, pp. 85–96.

[11] A. Welc, S. Jagannathan, and A. Hosking, "Safe futures for Java," *ACM SIGPLAN Notices*, vol. 40, no. 10, pp. 439–453, Oct. 2005.

[12] A. Aviram, S.-C. Weng, S. Hu, and B. Ford, "Efficient system-enforced deterministic parallelism," *Commun. ACM*, vol. 55, no. 5, pp. 111–119, 2012. [Online]. Available: http://doi.acm.org/10.1145/2160718.2160742

[13] E. D. Berger, T. Yang, T. Liu, and G. Novark, "Grace: safe multithreaded programming for C/C++," *ACM SIGPLAN Notices*, vol. 44, no. 10, pp. 81–96, Oct. 2009.

[14] T. Liu, C. Curtsinger, and E. D. Berger, "Dthreads: Efficient deterministic multithreading," in *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (23rd SOSP'11)*. Cascais, Portugal: ACM Press, Oct. 2011, pp. 327–336.

[15] W. chun Feng and S. Xiao, "To gpu synchronize or not gpu synchronize?" in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 2010, pp. 3801–3804.

[16] S. Xiao and W. chun Feng, "Inter-block GPU communication via fast barrier synchronization," in *IPDPS*. IEEE, 2010, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2010.5470477

[17] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "Timegraph: Gpu scheduling for real-time multi-tasking environments," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002181.2002183

[18] W. Fung, I. Singh, A. Brownsword, and T. Aamodt, "Kilo tm: Hardware transactional memory for gpu architectures," *Micro, IEEE*, vol. 32, no. 3, pp. 7–16, 2012.

[19] M. Boyer, K. Skadron, and W. Weimer, "Automated Dynamic Analysis of CUDA Programs," 2008. [Online]. Available: http://www.cs.virginia.edu/~{}skadron/Papers/stmcs08.pdf

[20] G. Li and G. Gopalakrishnan, "Scalable SMT-based verification of GPU kernel functions," in *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*, G.-C. Roman and K. J. Sullivan, Eds. ACM, 2010, pp. 187–196. [Online]. Available: http://doi.acm.org/10.1145/1882291.1882320

[21] A. Betts, N. Chong, A. Donaldson, S. Qadeer, and P. Thomson, "GPUVerify: a verifier for GPU kernels," *ACM SIGPLAN Notices*, vol. 47, no. 10, pp. 113–132, Oct. 2012.

[22] M. Bond, "GPUDet: a deterministic GPU architecture," *ACM SIGPLAN Notices*, vol. 48, no. 4, pp. 1–12, Apr. 2013.

[23] M. D. Hill and M. Xu, "Racey: A stress test for deterministic execution," 2009. [Online]. Available: http://pages.cs.wisc.edu/~markhill/racey.html

[24] M. Papadopoulou, M. Sadooghi-Alvandi, and H. Wong, "Micro-benchmarking the gt200 gpu," Computer Group, ECE, University of Toronto, Tech. Rep., 2009.

[25] S. R. Sarangi, B. Greskamp, and J. Torrellas, "CADRE: Cycle-accurate deterministic replay for hardware debugging," in *DSN*. IEEE Computer Society, 2006, pp. 301–312. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/DSN.2006.19

[26] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying gpu microarchitecture through microbenchmarking," in *ISPASS*, 2010.

[27] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-aware intelligent DRAM refresh," in *ISCA*. IEEE, 2012, pp. 1–12. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6230820

[28] C. M. Wittenbrink, E. Kilgariff, and A. Prabhu, "Fermi GF100 GPU architecture," *IEEE Micro*, vol. 31, no. 2, pp. 50–59, Mar./Apr. 2011.

[29] NVIDIA Corporation, *NVIDIA CUDA C Programming Guide*, June 2011.

[30] S. Collange, D. Defour, and A. Tisserand, "Power consumption of GPUs from a software perspective," in *Computational Science - ICCS 2009, 9th International Conference, Baton Rouge, LA, USA, May 25-27, 2009, Proceedings, Part I*, ser. Lecture Notes in Computer Science, G. Allen, J. Nabrzyski, E. Seidel, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, Eds., vol. 5544. Springer, 2009, pp. 914–923. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01970-8

[31] K. Doertel, "Best known method: Avoid heterogeneous precision in control flow calculations," Intel, Tech. Rep., 2013.

[32] S. Collange, D. Defour, S. Graillat, and R. Iakymchuk, "Full-speed deterministic bit-accurate parallel floating-point summation on multi- and many-core architectures," HAL-CCSD, Tech. Rep. hal-00949355, 2014. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00949355

ANNEXE

Table II and III correspond to the measured predictability for the scheduling of warp according to the *clock* and *atomic* test respectively. Table IV and V correspond to the measured predictability for the scheduling of block according to the *clock* and *atomic* test respectively. The reduced frequency execution mode consists in settting the set (GPU clock rate; Memory clock rate) to (1059MHZ; 324MHZ) for the GTX680, and (101MHZ; 101MHZ) for the GTX480 and GTX560.

TABLE II. PREDICTABILITY REGARDING *clock* TEST FOR SCHEDULING OF WARP ACCORDING TO THE NUMBER OF WARP LAUNCHED PER BLOCK. NUMBER OF OBSERVED EXECUTION VECTORS AND STATISTICAL MODE IS GIVEN FOR STANDARD FREQUENCY, REDUCED FREQUENCY, WITH AND WITHOUT SYNCHRONIZATION, WITH AND WITHOUT INITIALIZATION OVER 1000 EXECUTIONS.

		Standard Frequency								Reduced Frequency							
		sync0				sync1				sync0				sync1			
		Init 0		Init 1		Init 0		Init 1		Init 0		Init 1		Init 0		Init 1	
	nb warp	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode
GTX680	2	2	75	1	100	2	75	1	100	2	75	1	100	2	75	1	100
	3	1	100	1	100	1	100	1	100	1	100	1	100	1	100	1	100
	4	2	50	1	100	2	50	1	100	2	50	1	100	2	50	1	100
	5	3	50	2	96,9	3	50	2	91,6	2	50	2	99,4	2	50	1	100
	6	6	25	2	98,3	6	25	2	93,7	5	25	2	99,9	4	25	1	100
	7	1	100	2	99,5	1	100	2	93,2	1	100	2	99,7	1	100	1	100
	8	3	50	2	98,4	2	50	3	93,9	2	50	2	99,9	2	50	1	100
	9	5	49,9	1	100	3	50	2	94	4	49,9	1	100	3	50	1	100
	10	12	17,2	1	100	12	16,8	2	93,2	12	17,7	1	100	9	14,4	1	100
	11	17	11	1	100	16	9,2	2	99,7	16	9,6	1	100	8	13,7	1	100
	12	66	4,4	2	99,9	64	3	2	99,3	64	3,7	1	100	32	4,4	1	100
	13	110	2,4	2	65,2	113	3,1	4	65,3	111	3	2	67,2	66	4,7	7	34,4
	14	324	1,6	3	34,4	390	1,1	4	34,1	325	1,7	3	35,2	207	1,8	8	23,4
	15	402	0,9	14	17,9	462	0,9	13	18,3	366	1	12	17,9	549	0,7	16	16,3
	16	781	0,6	18	10,1	831	0,4	21	9,4	820	0,5	18	9,4	892	0,4	24	8,1
	17	910	0,4	30	15,7	915	0,3	32	14,5	889	0,5	30	14,1	928	0,4	24	9,8
	18	975	0,4	45	7,3	996	0,2	51	8,2	991	0,2	45	6,8	986	0,3	36	4,3
	19	975	0,5	52	12,3	1000	0,1	56	10,9	995	0,2	52	10,9	999	0,2	32	6
	20	1000	0,1	67	7,7	999	0,2	71	7,6	1000	0,1	68	6,6	999	0,2	45	4,2
	21	999	0,2	68	8,3	1000	0,1	70	6,5	1000	0,1	67	6,4	999	0,2	47	3,6
	22	1000	0,1	65	7,8	1000	0,1	72	7,2	1000	0,1	68	7	1000	0,1	45	3,8
	23	1000	0,1	65	7,4	1000	0,1	68	6,3	1000	0,1	66	6,6	1000	0,1	45	3,6
	24	1000	0,1	64	7,5	1000	0,1	67	6,5	1000	0,1	66	6,7	999	0,2	45	3,9
	25	1000	0,1	65	6,2	1000	0,1	66	6	1000	0,1	65	6,4	1000	0,1	45	3,9
	26	1000	0,1	57	7,5	1000	0,1	60	7,4	1000	0,1	58	8,1	1000	0,1	33	7,1
	27	1000	0,1	52	7,4	1000	0,1	56	7,8	1000	0,1	52	8,7	999	0,2	33	6,2
	28	1000	0,1	48	7,4	1000	0,1	50	7,1	1000	0,1	49	9,3	1000	0,1	27	7,7
	29	1000	0,1	72	5,5	1000	0,1	75	6,3	1000	0,1	74	6,3	1000	0,1	27	6,7
	30	1000	0,1	70	5,8	1000	0,1	71	5,7	1000	0,1	70	5,7	1000	0,1	27	7
	31	1000	0,1	65	5,3	1000	0,1	67	4,8	1000	0,1	63	5,6	998	0,2	27	6,4
	32	1000	0,1	54	4,8	1000	0,1	57	5,1	1000	0,1	53	5,6	1000	0,1	27	6,9
GTX560	2	1	100	1	100	2	75	1	100	1	100	1	100	1	100	1	100
	3	1	100	1	100	1	100	1	100	1	100	1	100	1	100	1	100
	4	1	100	1	100	2	50	1	100	1	100	1	100	1	100	1	100
	5	1	100	1	100	3	50	2	93,3	1	100	1	100	1	100	1	100
	6	1	100	1	100	5	25	2	92,5	1	100	1	100	1	100	1	100
	7	1	100	1	100	1	100	2	94,6	1	100	1	100	1	100	1	100
	8	1	100	1	100	2	50	2	93,8	1	100	1	100	1	100	1	100
	9	1	100	1	100	3	50	2	93,1	1	100	1	100	1	100	1	100
	10	1	100	1	100	13	18,8	2	93,6	1	100	1	100	1	100	1	100
	11	1	100	1	100	16	9,1	2	99,8	1	100	1	100	1	100	1	100
	12	1	100	1	100	64	2,7	2	99,9	1	100	1	100	1	100	1	100
	13	1	100	1	100	112	2,8	4	64,9	1	100	1	100	1	100	1	100
	14	1	100	1	100	376	1,2	4	34,7	1	100	1	100	1	100	1	100
	15	1	100	1	100	460	0,7	13	18,5	1	100	1	100	1	100	1	100
	16	1	100	1	100	820	0,4	24	9,4	1	100	1	100	1	100	1	100
	17	3	50	1	100	935	0,3	32	13,2	3	50	1	100	3	50	1	100
	18	1	100	1	100	998	0,2	45	7,8	2	83,9	1	100	2	99,3	1	100
	19	2	50	1	100	1000	0,1	53	12,2	2	50,1	1	100	2	50,2	1	100
	20	1	100	1	100	1000	0,1	70	6,4	1	100	1	100	1	100	1	100
	21	2	50	1	100	1000	0,1	72	6,2	2	50	1	100	2	50,2	2	99
	22	1	100	1	100	1000	0,1	70	7,2	1	100	1	100	1	100	1	100
	23	2	50	1	100	1000	0,1	68	6,5	2	50,2	1	100	2	50	2	98,3
	24	1	100	1	100	1000	0,1	72	6,5	1	100	1	100	1	100	1	100
	25	2	50	1	100	1000	0,1	68	5,6	2	50	1	100	2	50,1	1	100
	26	1	100	1	100	1000	0,1	59	6,5	1	100	1	100	1	100	1	100
	27	2	50	1	100	1000	0,1	53	8,1	2	50,2	1	100	2	50	2	97,3
	28	1	100	1	100	1000	0,1	50	9,9	2	89,9	1	100	1	100	1	100
	29	2	50	1	100	1000	0,1	77	5,5	2	50	1	100	2	50,2	1	100
	30	1	100	1	100	1000	0,1	73	6,5	1	100	1	100	1	100	1	100
	31	2	50	1	100	1000	0,1	65	4,5	2	50,1	1	100	2	50,1	1	100
	32	1	100	1	100	1000	0,1	58	4,6	1	100	1	100	2	96,4	1	100
GTX480	2	1	100	1	100	2	75	1	100	1	100	1	100	1	100	1	100
	3	1	100	1	100	1	100	1	100	1	100	1	100	2	91,9	1	100
	4	1	100	1	100	2	50	1	100	1	100	1	100	1	100	1	100
	5	1	100	1	100	3	50	2	92,4	1	100	1	100	3	90,6	1	100
	6	2	91,9	1	100	5	25	2	91,2	2	91,6	1	100	2	94,3	1	100
	7	3	98,8	1	100	1	100	2	92,7	1	100	1	100	2	98,1	1	100
	8	1	100	1	100	2	50	2	93,2	3	97,6	1	100	1	100	1	100
	9	3	92,5	1	100	4	50	2	94,4	3	91,5	1	100	1	100	1	100
	10	1	100	1	100	12	17,3	2	94,3	3	93,1	1	100	3	92	1	100
	11	3	84	1	100	16	10	2	99,6	3	84,1	1	100	3	84,2	1	100
	12	1	100	1	100	64	3,6	2	99,7	1	100	1	100	1	100	1	100
	13	1	100	1	100	111	2,6	3	65,3	1	100	1	100	3	90	1	100
	14	1	100	1	100	391	1,2	4	34,4	1	100	1	100	3	89,2	1	100
	15	7	92,3	1	100	465	0,8	16	17,1	7	92,5	1	100	3	92	1	100
	16	1	100	1	100	824	0,4	22	9,5	1	100	1	100	3	86,4	1	100
	17	2	50	1	100	898	0,4	34	14,2	2	50	1	100	1	100	1	100
	18	1	100	1	100	996	0,2	49	7,3	1	100	1	100	1	100	1	100
	19	2	50	1	100	998	0,2	53	12,6	2	50	1	100	2	50	1	100
	20	1	100	1	100	1000	0,1	70	7,2	1	100	1	100	1	100	1	100
	21	2	50	2	99,9	1000	0,1	69	5,8	2	50	1	100	1	100	1	100
	22	1	100	1	100	1000	0,1	71	7,4	1	100	1	100	1	100	1	100
	23	2	50	1	100	1000	0,1	66	6,8	2	50	1	100	1	100	1	100
	24	1	100	1	100	1000	0,1	68	6,9	1	100	1	100	1	100	1	100
	25	2	50	1	100	1000	0,1	67	6,8	2	50	1	100	1	100	1	100
	26	1	100	1	100	1000	0,1	59	6,5	1	100	1	100	1	100	1	100
	27	2	50	1	100	1000	0,1	54	8	2	50	1	100	1	100	1	100
	28	1	100	1	100	1000	0,1	51	7,5	1	100	1	100	1	100	1	100
	29	2	50	1	100	1000	0,1	74	5,2	2	50	1	100	1	100	1	100
	30	1	100	1	100	1000	0,1	70	5,8	1	100	1	100	1	100	1	100
	31	2	50	1	100	1000	0,1	65	4,8	3	50	1	100	1	100	1	100
	32	1	100	1	100	1000	0,1	55	4,9	1	100	1	100	1	100	1	100

TABLE III. PREDICTABILITY REGARDING *atomic* TEST FOR SCHEDULING OF WARP ACCORDING TO THE NUMBER OF WARP LAUNCHED PER BLOCK. NUMBER OF OBSERVED EXECUTION VECTORS AND STATISTICAL MODE IS GIVEN FOR STANDARD FREQUENCY, REDUCED FREQUENCY, WITH AND WITHOUT SYNCHRONIZATION, WITH AND WITHOUT INITIALIZATION OVER 1000 EXECUTIONS.

		Standard Frequency								Reduced Frequency							
		sync0				sync1				sync0				sync1			
		Init 0		Init 1		Init 0		Init 1		Init 0		Init 1		Init 0		Init 1	
	nb warp	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode
GTX680	2	2	75	1	100	2	75	2	98,7	2	75	1	100	2	82,5	1	100
	3	1	100	1	100	3	99,4	4	99,1	1	100	1	100	4	74,1	3	42,4
	4	2	50	1	100	3	50	1	100	2	50	1	100	6	39,2	3	48,5
	5	6	34,1	2	74,3	4	48,7	2	51,3	5	49,5	2	75,2	6	30,4	4	58,1
	6	17	19,2	2	74,7	10	24,7	2	51,6	18	24,6	2	75,5	6	52,7	2	78,6
	7	6	43	3	50,2	4	50,8	4	49,6	7	43,3	3	51,6	2	52,2	2	59,9
	8	17	20,5	4	26,4	10	30,6	5	47,7	15	25,2	4	25,9	4	40	3	60,4
	9	23	21,5	6	34,2	13	32,1	5	67,2	26	14,7	6	33,3	8	24,1	8	19,8
	10	114	7,1	6	19,1	38	8,8	4	50,2	101	7	6	18,8	57	7,1	16	18,4
	11	116	3,6	24	6,7	53	9,7	6	45,7	124	6,7	24	6,2	282	1,3	53	8,6
	12	364	0,9	36	3,4	147	2,5	7	54,5	372	1,4	36	4	602	0,7	89	8,8
	13	666	0,6	42	8,5	374	1,9	32	27,4	536	1	42	7,5	688	0,6	144	5,7
	14	901	0,4	27	8,4	907	0,3	65	15,9	866	0,5	27	8,7	925	0,3	121	3,6
	15	986	0,2	71	8,3	979	0,3	169	4,1	973	0,2	71	5,3	965	0,3	141	2,7
	16	990	0,2	37	6,8	999	0,2	221	2,4	994	0,2	38	6,7	989	0,2	86	3,7
	17	996	0,2	415	1,1	998	0,2	240	3,4	997	0,2	177	2,9	998	0,2	150	2,9
	18	1000	0,1	356	1,8	1000	0,1	245	2,6	1000	0,1	143	3,1	998	0,2	124	2,9
	19	1000	0,1	339	1,1	1000	0,1	204	2,4	1000	0,1	199	2,5	1000	0,1	141	2,7
	20	1000	0,1	135	2,2	1000	0,1	205	2,9	1000	0,1	144	1,6	1000	0,1	157	2,6
	21	1000	0,1	145	2	1000	0,1	213	2,7	1000	0,1	207	1,9	1000	0,1	181	2,7
	22	1000	0,1	201	2,4	1000	0,1	213	2,2	1000	0,1	250	1,6	999	0,2	188	2,7
	23	1000	0,1	220	2,3	1000	0,1	207	2,5	1000	0,1	198	2	1000	0,1	202	2,6
	24	1000	0,1	211	1,9	1000	0,1	183	3,1	1000	0,1	151	2,1	1000	0,1	228	2,4
	25	1000	0,1	186	2	1000	0,1	220	2,5	1000	0,1	188	1,7	1000	0,1	248	2,1
	26	1000	0,1	169	2,4	1000	0,1	183	2	1000	0,1	213	2,3	1000	0,1	214	2,5
	27	1000	0,1	192	1,7	1000	0,1	211	2,2	1000	0,1	215	2	1000	0,1	190	2,8
	28	1000	0,1	127	2,1	1000	0,1	220	2,5	1000	0,1	108	2,2	1000	0,1	165	2,7
	29	1000	0,1	199	1,9	1000	0,1	207	2,8	1000	0,1	108	2,1	1000	0,1	173	3,3
	30	1000	0,1	455	0,8	1000	0,1	336	1,8	1000	0,1	181	1,6	1000	0,1	236	2,3
	31	1000	0,1	703	0,7	1000	0,1	520	2,1	1000	0,1	266	2	1000	0,1	432	1,4
	32	1000	0,1	898	0,4	1000	0,1	681	0,8	1000	0,1	349	1,3	1000	0,1	665	1,2
GTX560	2	1	100	1	100	2	74,4	2	99,5	1	100	1	100	2	90,8	1	100
	3	1	100	1	100	4	99,5	3	99,7	1	100	1	100	1	100	1	100
	4	1	100	1	100	3	50	1	100	1	100	1	100	1	100	1	100
	5	10	68,8	17	23,2	4	49,3	2	52,4	21	20,3	19	20,8	1	100	3	56,1
	6	6	49,8	20	12,7	10	24,6	2	51,8	19	28,1	19	18,4	6	71,2	6	26,2
	7	66	8,1	58	13	4	50,5	4	50,1	14	33,5	70	13,9	15	22,3	23	27,6
	8	49	18,6	79	12,3	10	33,8	5	47,9	26	16,6	78	12,3	9	47,3	69	16,2
	9	270	8,8	127	5	12	33,2	5	65,7	70	9,1	182	4,3	58	11,2	236	4,7
	10	357	3,2	381	4,1	33	8,8	4	51,7	463	2	341	4,3	354	6,9	447	2,4
	11	497	4,2	469	2,9	49	9,5	8	47,7	625	1,2	511	4,4	755	1,9	615	2,3
	12	629	2,8	633	1,2	137	2,8	8	55,6	408	4	639	1,9	276	14,3	577	3,1
	13	632	3,4	744	1,3	388	1,6	35	27,4	618	1,8	765	1	700	3,9	766	1,3
	14	618	2,4	724	1,2	893	0,4	62	16,7	483	2,1	751	1,6	675	3,2	816	1,2
	15	794	1,6	811	0,9	979	0,3	173	4,2	710	1,4	793	1,1	849	1,6	955	0,5
	16	650	3,3	801	1,2	997	0,2	216	2,8	915	0,8	931	0,6	947	0,4	853	2,1
	17	820	0,8	446	2,6	997	0,2	223	2,5	806	0,9	747	1,6	923	0,4	726	1,1
	18	819	1,4	670	2,8	1000	0,1	237	2	566	2	807	1	824	0,9	736	1,7
	19	643	3,1	610	2,1	1000	0,1	212	2,4	664	1,7	744	0,9	776	1	794	1
	20	620	3,7	510	4,7	1000	0,1	204	2,3	592	2,7	647	1,7	781	1,7	705	2
	21	709	2,3	501	2,7	1000	0,1	201	2,2	647	1,6	729	1,5	935	0,5	865	0,8
	22	752	2,2	705	1,2	1000	0,1	201	2,1	545	2,4	758	1,7	849	0,7	776	1,3
	23	767	1,5	650	1,9	1000	0,1	184	2,7	801	2,2	829	0,8	944	0,6	855	1,8
	24	703	1,5	794	1,4	1000	0,1	192	3,2	690	1,8	883	0,9	861	1	877	1
	25	704	1,9	714	1,4	1000	0,1	228	2,1	787	1,7	840	0,8	964	0,6	910	0,6
	26	629	4,2	718	1,1	1000	0,1	181	2,4	914	0,9	774	2	880	0,8	872	0,8
	27	939	0,5	783	1,4	1000	0,1	210	2,2	945	0,9	872	1	979	0,4	881	1,1
	28	930	0,6	828	0,7	1000	0,1	241	2,3	992	0,2	923	0,4	961	0,4	904	1,8
	29	769	1	788	2,4	1000	0,1	218	2,8	932	0,6	876	0,8	997	0,2	948	0,5
	30	819	2	862	0,9	1000	0,1	334	1,7	935	0,6	934	0,4	987	0,3	957	0,4
	31	955	0,7	806	1,1	1000	0,1	525	1	992	0,2	939	0,4	999	0,2	976	0,4
	32	863	0,7	822	1,2	1000	0,1	681	1,1	884	0,6	939	0,8	987	0,2	948	0,3
GTX480	2	1	100	1	100	2	75,9	2	99,6	2	93,4	2	91,5	1	100	1	100
	3	1	100	1	100	4	99,3	4	99,3	1	100	1	100	2	60,2	3	42,2
	4	4	75,5	1	100	2	50	1	100	4	43,6	1	100	12	49,6	2	61,1
	5	11	50,1	4	38,5	4	48,9	2	54	51	14,4	4	41,7	16	24,5	4	35,8
	6	62	23,3	12	18,4	10	24,6	2	51,2	114	10,9	12	19,7	108	5,2	12	13,8
	7	36	22	22	21,3	4	52,8	4	51,2	42	10,1	24	12,7	355	3,1	48	6,4
	8	119	12,9	58	22	10	31,9	5	47,2	398	1,9	115	4,3	493	1,6	196	3,8
	9	118	10,8	155	9,3	13	33,1	4	65,7	326	3,8	332	2,9	631	1,2	438	1,9
	10	112	14,1	194	7,9	38	10,7	4	51,5	210	15,3	561	1,3	540	2,1	531	1,7
	11	105	12,8	175	6,9	51	9,8	7	46,4	538	1,1	681	0,9	810	0,6	663	1
	12	222	14,3	328	3,5	139	3,1	7	55,4	771	0,9	733	1,9	869	0,5	758	1,5
	13	237	12,7	533	2,4	380	1,7	31	27,3	903	0,6	647	2,1	788	0,9	727	1,1
	14	287	4,9	591	3,3	900	0,4	63	17,8	945	0,3	846	0,8	826	0,8	867	1,2
	15	648	1,5	680	1,2	981	0,2	164	4	952	0,7	858	0,7	926	0,4	811	1
	16	566	1,4	621	2,7	998	0,2	219	2,3	969	0,6	939	0,5	982	0,3	927	0,8
	17	530	1,7	636	1,3	1000	0,1	232	2,4	979	0,3	968	0,3	974	0,3	968	0,4
	18	469	2,4	591	3,2	999	0,2	244	2,4	946	0,4	970	0,6	939	0,4	975	0,3
	19	511	3,6	571	2,3	1000	0,1	206	2,6	980	0,3	982	0,3	971	0,4	979	0,3
	20	379	7,8	573	1,9	1000	0,1	217	2,6	967	0,3	969	0,4	969	0,4	992	0,2
	21	696	1,1	721	1,7	1000	0,1	203	2,7	997	0,2	981	0,2	988	0,2	984	0,3
	22	605	1,8	780	1,2	1000	0,1	192	2,9	976	0,4	988	0,3	979	0,3	989	0,2
	23	807	2,2	832	0,6	1000	0,1	188	2,8	1000	0,1	993	0,2	994	0,2	985	0,2
	24	749	2,2	789	1,5	1000	0,1	196	2,7	997	0,3	999	0,2	995	0,2	991	0,2
	25	807	1,5	955	0,5	1000	0,1	234	2,1	997	0,2	996	0,2	995	0,2	998	0,2
	26	714	2,2	838	0,8	1000	0,1	203	2,7	990	0,2	990	0,2	997	0,2	996	0,2
	27	861	0,9	918	0,6	1000	0,1	214	2,5	997	0,2	994	0,2	999	0,2	998	0,2
	28	901	0,8	989	0,3	1000	0,1	235	2,2	1000	0,1	997	0,2	1000	0,1	1000	0,1
	29	922	0,6	948	0,5	1000	0,1	215	2,4	999	0,2	998	0,2	1000	0,1	999	0,2
	30	899	1	993	0,2	1000	0,1	337	2,1	1000	0,1	999	0,2	999	0,2	1000	0,1
	31	944	0,4	985	0,3	1000	0,1	502	1,2	998	0,3	1000	0,1	1000	0,1	1000	0,1
	32	995	0,2	999	0,2	1000	0,1	669	1,4	1000	0,1	1000	0,1	1000	0,1	1000	0,1

TABLE IV. PREDICTABILITY REGARDING *clock* TEST FOR SCHEDULING OF BLOCK ACCORDING TO THE NUMBER OF BLOCK LAUNCHED. NUMBER OF OBSERVED EXECUTION VECTORS AND STATISTICAL MODE IS GIVEN FOR STANDARD FREQUENCY, REDUCED FREQUENCY, WITH AND WITHOUT SYNCHRONIZATION, WITH AND WITHOUT INITIALIZATION OVER 1000 EXECUTIONS.

		Standard Frequency								Reduced Frequency							
		sync0				sync1				sync0				sync1			
		Init 0		Init 1		Init 0		Init 1		Init 0		Init 1		Init 0		Init 1	
	nb block	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode
GTX680	2	1	100	2	97,9	2	96,4	2	97,8	2	65,2	1	100	3	84,1	1	100
	3	4	89,8	4	68,3	7	70,2	4	73,1	1	100	3	66,3	7	91	3	90,8
	4	1	100	15	50,4	10	84,6	15	47,8	1	100	13	24,4	1	100	13	53,9
	5	1	100	33	35,9	1	100	30	34,8	1	100	13	21,2	1	100	12	52,8
	6	1	100	35	36	1	100	30	39,4	1	100	29	23,8	1	100	29	40,8
	7	1	100	68	17,3	1	100	65	19,1	1	100	47	16,4	1	100	44	50
	8	1	100	113	23,6	1	100	106	24,6	1	100	83	7,5	1	100	89	23,1
	9	2	74,3	141	26,4	2	74,4	150	24,6	2	67,1	81	6,3	2	75	102	19,7
	10	3	74,7	146	24,5	2	74,1	142	28,2	2	67,4	150	4,1	2	75	125	19,6
	11	2	74,6	208	19,3	2	74,8	200	19,7	15	64,4	226	5	2	75	190	13,5
	12	2	74,5	302	16,1	2	74,7	270	19,7	2	67	315	2,4	2	75	225	11,8
	13	4	56,5	291	17,5	4	57,1	282	20	22	52,9	318	2,5	4	50	237	12,1
	14	4	56	304	19,7	4	57	274	19,7	4	52,7	396	2,2	4	50	262	12,1
	15	4	58,4	346	15,3	4	56,9	334	18,2	4	54	481	2	4	50	309	10,3
	16	4	57	393	17,7	4	54,9	372	16,3	4	52,4	548	1,4	4	50	349	7,6
	17	2	74,4	406	14,9	2	72,2	367	19,7	2	69,1	568	2	2	50	548	4,8
	18	7	30,1	412	15,4	7	29,7	373	18,2	32	29,4	647	1,4	4	25	368	8,3
	19	11	26,1	489	13,4	11	25,9	466	14,4	11	26	753	1,4	5	25	412	5,6
	20	19	25,3	539	10,1	19	25,3	521	11,7	68	24,8	787	0,6	4	25	472	3,6
	21	24	25	555	8,1	23	25	518	11,2	81	23,3	794	0,8	4	25	468	4,7
	22	29	19,7	587	7,4	28	21,1	540	8,8	54	22	836	0,5	4	25	502	4,5
	23	39	14,6	677	3,7	39	15,2	657	4,4	41	16,1	877	0,3	4	25	548	4,1
	24	57	11,1	733	2,8	56	11,1	713	3,2	93	11,7	861	0,5	4	25	596	2,4
	25	53	9,7	810	1,7	54	8,9	758	2,3	84	10,3	934	0,3	4	25	623	2,7
	26	52	15,5	788	1,9	49	11,3	786	2,9	50	12,4	943	0,3	4	25	627	2,6
	27	50	18,1	820	1,9	47	16,9	813	1,5	47	18,4	958	0,3	4	25	703	1,3
	28	48	25	866	1	45	25	850	1,4	69	24,4	979	0,3	5	25	753	1,2
	29	49	19,2	877	0,9	48	20,6	866	1,5	108	16,4	969	0,2	4	25	746	1,1
	30	52	14,6	891	1	50	14,8	856	1,5	168	13,4	970	0,3	4	25	761	1,2
	31	55	11,2	898	1,2	51	11,2	890	0,8	79	11,6	947	0,3	4	25	799	1,4
	32	60	9,2	902	0,8	61	8,2	901	1	57	10,8	938	0,4	4	25	833	0,6
GTX560	2	1	100	1	100	3	98,7	1	100	1	100	1	100	1	100	1	100
	3	1	100	1	100	1	100	4	70,1	1	100	1	100	1	100	1	100
	4	1	100	1	100	1	100	15	46,4	1	100	1	100	1	100	1	100
	5	1	100	1	100	1	100	32	37,7	1	100	1	100	1	100	1	100
	6	1	100	1	100	1	100	34	34,6	1	100	1	100	1	100	2	99,9
	7	1	100	1	100	1	100	71	19,7	1	100	1	100	1	100	2	99,9
	8	1	100	1	100	1	100	128	23	1	100	1	100	1	100	2	99,7
	9	2	99,9	1	100	2	74	158	25,2	2	99,9	2	99,9	1	100	2	99,9
	10	2	50	2	99,9	2	74,2	156	25,9	2	50	1	100	2	50	1	100
	11	2	50	2	75,4	2	74,6	209	21,5	2	50	1	100	2	50	2	99,8
	12	2	50	2	95,3	2	74,6	260	18,2	2	50	1	100	2	50	3	99,5
	13	3	50	2	65,5	4	57,2	298	20	2	50	1	100	2	50	4	49,9
	14	2	50	1	100	4	57,2	279	18,3	2	50	1	100	2	50	4	50,4
	15	3	50	1	100	4	54,7	363	15,9	2	50	1	100	3	50	8	26,1
	16	2	50	1	100	4	55,5	377	16,3	2	50	1	100	2	50	5	53,6
	17	2	50	1	100	2	73,8	381	18,4	2	50	1	100	2	50	3	80,2
	18	2	50	1	100	7	29,4	381	18,8	2	50	1	100	2	50	2	82,4
	19	3	50	1	100	11	25,9	432	16,6	2	50	1	100	2	50	2	78,4
	20	2	50	1	100	19	25,4	499	13	2	50	2	72,4	2	50	2	75,7
	21	2	50	1	100	27	25	517	12,8	2	50	14	87,6	2	50	3	45,4
	22	2	50	2	99,9	28	19,6	555	12	2	50	2	83	2	50	3	46
	23	3	50	1	100	38	15,3	635	5,6	2	50	1	100	2	50	3	59,8
	24	2	50	1	100	58	10,1	721	2,9	2	50	1	100	2	50	3	55,7
	25	2	50	1	100	53	8,3	763	2,4	2	50	2	99,9	2	50	4	57,3
	26	2	50	1	100	52	13,2	763	1,9	2	50	1	100	2	50	6	58,3
	27	3	50	1	100	47	19	806	1,5	2	50	1	100	2	50	6	50,4
	28	2	50	1	100	47	25	861	1,3	2	50	1	100	2	50	8	27,3
	29	3	50	1	100	49	20,2	870	1,6	2	50	1	100	6	34,3	10	25,3
	30	2	50	1	100	50	13,8	869	1,1	2	50	1	100	6	41	16	13,5
	31	2	50	1	100	57	11,3	861	1,4	2	50	1	100	4	37,7	14	15,2
	32	2	50	2	57,9	61	8,5	874	0,9	2	50	1	100	53	30	17	13,8
GTX480	2	1	100	1	100	2	99,4	1	100	1	100	1	100	1	100	1	100
	3	1	100	1	100	1	100	4	70,1	1	100	1	100	1	100	1	100
	4	1	100	1	100	1	100	15	48,4	1	100	1	100	1	100	1	100
	5	1	100	1	100	1	100	31	32,7	1	100	1	100	1	100	1	100
	6	1	100	1	100	1	100	35	33,7	1	100	1	100	1	100	1	100
	7	1	100	1	100	1	100	63	18,9	1	100	1	100	1	100	1	100
	8	1	100	1	100	1	100	115	23,6	1	100	1	100	1	100	1	100
	9	1	100	1	100	2	74,6	152	25,8	2	99,9	1	100	1	100	1	100
	10	1	100	1	100	2	74,7	160	24,5	1	100	1	100	1	100	1	100
	11	1	100	1	100	2	75	200	17,8	1	100	1	100	1	100	1	100
	12	1	100	1	100	3	74,1	268	19,1	1	100	1	100	1	100	1	100
	13	1	100	1	100	4	54,9	290	16,8	1	100	1	100	1	100	1	100
	14	1	100	1	100	4	56,5	306	19,9	1	100	1	100	1	100	1	100
	15	1	100	1	100	4	56,2	319	15,9	1	100	1	100	1	100	1	100
	16	1	100	1	100	4	56,4	374	13,8	1	100	1	100	1	100	1	100
	17	1	100	1	100	2	73,6	376	19	1	100	1	100	1	100	1	100
	18	1	100	1	100	7	31,3	412	14,9	1	100	1	100	1	100	1	100
	19	1	100	1	100	11	27	455	14,8	1	100	1	100	1	100	1	100
	20	1	100	1	100	19	25,2	486	10,4	1	100	1	100	1	100	1	100
	21	1	100	1	100	30	25	530	8,6	1	100	1	100	1	100	1	100
	22	1	100	1	100	29	20	598	9,1	1	100	1	100	1	100	1	100
	23	1	100	1	100	38	14,2	637	5,2	1	100	1	100	1	100	1	100
	24	1	100	1	100	56	12,1	730	4,6	1	100	1	100	1	100	1	100
	25	1	100	1	100	52	9,8	794	2,5	1	100	1	100	1	100	2	99,9
	26	4	44,9	1	100	49	14,1	780	1,6	4	44,9	1	100	4	44,9	1	100
	27	9	40,6	1	100	48	19,4	800	2,3	8	40,6	2	99,9	8	40,6	1	100
	28	12	36,5	1	100	48	25	854	0,9	12	36,5	1	100	12	36,5	2	99,9
	29	12	36,4	1	100	48	20,8	853	1,5	12	36,4	2	99,9	12	36,4	1	100
	30	16	32,1	2	99,9	51	14,8	861	1,2	16	32,1	1	100	16	32,1	1	100
	31	21	27,8	1	100	55	12	877	0,9	20	27,8	2	99,9	20	27,8	1	100
	32	24	23,8	2	99,9	62	9,4	901	0,9	24	23,8	1	100	24	23,8	2	99,9

TABLE V. PREDICTABILITY REGARDING *atomic* TEST FOR SCHEDULING OF BLOCK ACCORDING TO THE NUMBER OF BLOCK LAUNCHED. NUMBER OF OBSERVED EXECUTION VECTORS AND STATISTICAL MODE IS GIVEN FOR STANDARD FREQUENCY, REDUCED FREQUENCY, WITH AND WITHOUT SYNCHRONIZATION, WITH AND WITHOUT INITIALIZATION OVER 1000 EXECUTIONS.

		Standard Frequency								Reduced Frequency							
		sync0				sync1				sync0				sync1			
		Init 0		Init 1		Init 0		Init 1		Init 0		Init 1		Init 0		Init 1	
	nb block	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode	Card	Mode
GTX680	2	1	100	2	59,5	1	100	1	100	2	99,6	2	52,4	1	100	1	100
	3	2	55,3	6	28,1	2	58,2	2	60,1	3	62,4	6	28,6	2	62,9	2	57,5
	4	6	32,5	15	21,3	6	33,1	6	33	7	37,4	11	41	6	27,8	6	53,4
	5	11	33,8	11	33,8	12	34,2	6	30,2	13	37,9	16	88	14	23,5	43	32,5
	6	18	53,1	32	18,5	18	45,9	10	46	14	69,3	20	57,4	26	35,7	20	69
	7	29	25,9	74	15,4	27	25,4	24	40,2	17	42,2	12	34,8	69	20	18	74,1
	8	107	9,8	192	8,5	111	10,1	80	8,8	46	23,4	31	17,5	242	6,8	16	42,2
	9	160	9,4	258	7,7	156	9,7	96	9,3	69	20,1	7	27,5	345	4	12	49
	10	270	5,1	316	6	270	4,4	143	9,4	119	10,5	18	20,7	469	3,1	9	39,4
	11	381	3,9	408	4,4	402	5,5	214	6,3	173	7,7	23	16,7	574	3,5	6	44,5
	12	425	4,3	514	2,8	433	4	257	4,9	216	7,2	32	28,5	605	2,2	13	51
	13	463	5,2	630	2,4	456	4,8	274	5,2	238	6,6	18	15,9	584	2,7	19	43,2
	14	465	3,9	692	1,3	475	2,9	306	6,4	242	5,3	10	37,8	545	2,5	13	43,6
	15	465	2,8	771	1,1	513	2,6	382	3,3	295	5,1	24	19	561	2,1	14	31,4
	16	452	4,3	802	0,7	502	3,8	420	3,7	293	5,9	24	15,2	541	2,7	31	41,3
	17	423	5	829	0,9	441	5	427	3,5	244	6,5	36	11,4	473	4,1	40	21,2
	18	704	1,5	893	0,9	740	1	492	1,9	637	1,4	12	53,6	737	1,1	37	22
	19	758	0,8	904	0,7	811	0,8	537	2,3	768	0,9	30	15,1	818	1,2	58	14,9
	20	771	1	944	0,4	819	1	584	2,1	782	1,4	154	4,9	810	0,7	87	14
	21	802	0,7	954	0,4	855	0,7	617	1,4	877	0,6	43	9,1	841	0,8	50	16,5
	22	852	0,7	953	0,5	905	0,4	710	1,4	825	0,8	90	22,8	855	0,6	21	22,5
	23	902	0,4	968	0,4	928	0,4	778	0,9	824	0,5	132	3,4	892	0,4	65	12,6
	24	915	0,4	953	0,4	940	0,4	845	0,9	853	0,5	349	1,9	909	0,4	307	2,8
	25	941	0,4	694	2,4	969	0,3	867	0,7	835	0,5	425	1,9	924	0,5	361	1,8
	26	950	0,3	266	6	973	0,2	944	0,4	863	0,5	176	7,4	937	0,7	164	8,7
	27	977	0,3	104	4,3	977	0,3	959	0,3	879	0,5	94	9,8	954	0,6	133	9,5
	28	1000	0,1	436	1,2	1000	0,1	700	0,6	1000	0,1	122	6,1	1000	0,1	206	3,3
	29	1000	0,1	445	1,2	1000	0,1	839	0,6	996	0,2	156	6,3	1000	0,1	368	2,1
	30	980	0,3	189	2,9	990	0,2	583	1	912	0,9	36	17,3	966	0,3	104	9,1
	31	984	0,3	242	1,7	992	0,3	642	1	953	0,5	54	7,4	975	0,3	156	4,3
	32	1000	0,1	494	1,3	1000	0,1	847	0,4	998	0,2	280	2	999	0,2	471	1,1
GTX560	2	1	100	1	100	1	100	1	100	1	100	2	99,4	1	100	2	87,1
	3	2	98,4	2	59,3	2	57,9	2	60,8	2	99,6	3	92,8	2	97	3	97,3
	4	2	95,1	6	35	6	32,9	6	32,1	3	98,3	3	99,7	1	100	4	78,8
	5	6	56,7	6	31,7	12	31	6	29,2	6	69,3	4	90,9	4	81,8	3	95,8
	6	8	52,1	10	45,2	16	46,5	10	41,6	11	62,8	8	71,3	3	84,4	9	74,6
	7	28	25,7	24	37,2	32	27	21	38,3	56	14,3	74	8,1	45	20,7	54	25,6
	8	93	9,7	79	12,1	105	8,5	78	8,4	46	17	143	5,7	76	24,6	106	24,7
	9	178	12,1	99	8,9	159	7,8	103	8,5	188	8,6	213	8,4	214	14	194	6,8
	10	242	9	145	12,5	271	6,2	140	8,5	227	5,6	215	5,1	118	14,6	139	18,2
	11	321	5	210	9,7	364	6,1	209	7	439	2,2	394	1,8	338	5,9	393	3,5
	12	488	4,7	265	5,1	455	5,6	263	5,2	692	1,3	606	0,9	582	2,3	713	1,2
	13	597	1,8	282	6,4	460	5,1	275	5,9	705	1,6	574	1,3	567	2,1	725	0,7
	14	910	0,7	291	7,5	493	2,3	310	6,2	926	0,5	857	0,6	821	1,4	845	1,1
	15	906	0,9	340	8	488	2,2	379	4,6	968	0,4	962	0,3	918	0,7	924	0,6
	16	920	0,5	414	3,4	488	3,5	417	4,1	959	0,4	937	0,4	888	0,6	909	0,5
	17	946	0,4	416	4,3	470	3,8	436	3,8	975	0,3	962	0,5	972	0,4	884	0,7
	18	952	0,7	490	3,6	717	1	487	2,5	986	0,2	981	0,3	975	0,3	914	0,9
	19	958	0,6	528	2,3	822	0,8	527	2,3	979	0,3	984	0,2	979	0,5	915	0,8
	20	991	0,3	530	1,8	794	0,8	557	1,6	985	0,2	993	0,2	985	0,2	921	0,6
	21	981	0,4	594	1,7	870	0,5	614	1,6	996	0,2	986	0,3	983	0,3	963	0,7
	22	986	0,4	660	2,7	886	0,6	732	1,3	996	0,2	996	0,2	980	0,5	982	0,6
	23	995	0,2	760	1,5	932	0,6	799	0,9	996	0,2	988	0,3	992	0,2	986	0,3
	24	998	0,2	799	1,5	952	0,3	851	0,6	998	0,2	996	0,2	995	0,2	987	0,3
	25	1000	0,1	845	0,8	966	0,3	883	0,6	997	0,2	990	0,2	995	0,2	990	0,4
	26	998	0,2	886	0,4	974	0,2	935	0,4	1000	0,1	996	0,2	992	0,3	997	0,2
	27	994	0,2	941	0,4	982	0,3	954	0,3	997	0,3	990	0,2	998	0,2	999	0,2
	28	994	0,2	657	0,7	1000	0,1	661	0,6	999	0,2	997	0,2	1000	0,1	999	0,2
	29	994	0,2	830	0,8	1000	0,1	813	0,6	999	0,2	999	0,2	999	0,2	1000	0,1
	30	995	0,3	602	1,2	985	0,3	573	0,9	1000	0,1	999	0,2	1000	0,1	999	0,2
	31	985	0,4	688	0,7	986	0,3	657	0,7	999	0,2	1000	0,1	1000	0,1	999	0,2
	32	997	0,2	842	0,4	1000	0,1	839	0,5	1000	0,1	1000	0,1	1000	0,1	1000	0,1
GTX480	2	1	100	2	78,9	1	100	1	100	1	100	1	100	1	100	1	100
	3	2	80,8	3	77,1	2	60,2	2	56,8	3	72,7	4	75,1	3	57	2	75,1
	4	2	78,1	2	99,9	6	36,6	6	28,7	3	70,4	2	86,5	2	87,4	3	80,5
	5	6	71,2	1	100	12	31,8	6	31	9	67,7	6	73	14	45,5	8	52,4
	6	11	70,6	4	52	18	46	11	43,3	13	59,8	12	41,7	11	64,7	11	49,4
	7	20	40,2	9	68	28	27,4	25	37,6	17	27	13	69	19	46,2	14	40,1
	8	44	33,6	14	66,8	107	10,5	81	8,5	33	38,4	27	55,2	25	38,9	20	51,1
	9	16	61,5	7	62	161	6,6	102	11	42	22,3	45	21,4	39	22,5	40	26,8
	10	28	26,4	16	39,3	272	5,2	145	10,6	59	13,9	60	20,2	56	26,9	58	12,3
	11	40	17,6	44	26,6	382	4,2	199	8,3	74	23,9	61	24,7	51	27,9	69	13,4
	12	26	23,5	22	26	429	5,4	268	5,2	73	16,1	98	16,9	58	18,6	91	11,6
	13	28	16,7	26	18,4	456	4	277	5,7	107	7,3	97	9,1	78	12,1	114	7,6
	14	42	49,7	98	20,6	486	2,6	309	7,6	159	8,6	165	8,9	156	6,3	193	5,7
	15	86	14,2	64	20,2	495	2,6	373	4,2	167	5,5	163	5,8	144	6	193	5,4
	16	106	8,9	167	6,1	473	3,9	429	4,3	318	3,3	323	2,6	251	4,6	309	4,2
	17	271	6,4	296	5,2	414	4,8	426	4,5	660	0,9	654	0,9	549	1,5	635	1,9
	18	377	2,4	274	4,2	713	1,2	495	1,8	625	1,7	645	0,8	501	1,8	633	1,5
	19	551	2	382	5,5	805	0,8	544	2,2	810	0,6	892	0,4	808	0,6	853	1,1
	20	652	2,8	697	1,3	830	1,1	549	2,3	944	0,5	968	0,3	878	0,5	937	0,5
	21	661	2,2	663	1,4	869	0,8	611	1,8	967	0,4	976	0,5	939	0,5	948	0,5
	22	842	1,1	735	1,2	883	0,8	746	1,8	974	0,3	987	0,2	934	0,5	962	0,4
	23	862	0,7	885	0,7	926	0,4	807	0,7	978	0,3	990	0,3	937	0,5	970	0,4
	24	819	0,9	790	1,5	945	0,3	846	0,9	973	0,3	984	0,3	965	0,4	966	0,4
	25	910	0,6	916	0,7	971	0,3	871	0,7	982	0,2	985	0,3	982	0,2	969	0,4
	26	945	0,4	857	0,8	973	0,3	925	0,3	996	0,2	991	0,3	992	0,2	987	0,2
	27	946	0,8	912	0,8	990	0,2	968	0,2	996	0,2	989	0,3	990	0,3	990	0,3
	28	968	0,5	922	0,8	1000	0,1	674	0,8	995	0,2	990	0,2	999	0,2	989	0,3
	29	643	2,4	611	2	1000	0,1	823	0,4	998	0,2	995	0,2	993	0,2	997	0,2
	30	737	1,4	725	1,4	987	0,3	598	1	996	0,2	997	0,2	998	0,2	997	0,2
	31	840	2	784	0,9	988	0,2	655	0,8	996	0,2	998	0,2	999	0,2	998	0,2
	32	863	1,1	840	0,6	1000	0,1	855	0,5	998	0,2	998	0,2	1000	0,1	999	0,2