Using Scheduling entropy amplification in CUDA/OpenMP code to exhibit non-reproducibility issues.

David DEFOUR, Université de Perpignan, FRANCE

Acknowledgment:

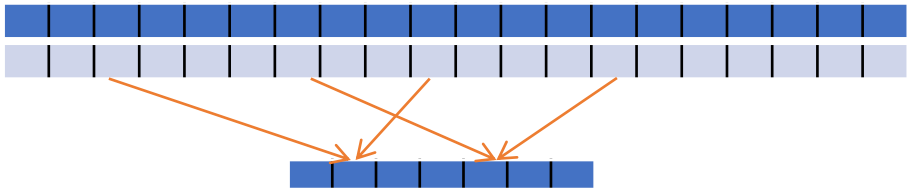
Funding: This work was supported by the ANR-20-CE46-0009 InterFLOP project

Motivations

- Usage of parallel primitive is increasing
- Locks, Atomics, Barrier:
  - Solve memory consistency issues
  - Does not address numerical issues such as reproducibility
  - Can leads to validation, debugging issues or deadlock
- Example: Floating-Point Histogram

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

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



$N=2^{16}$ values with $C=10^{8}$,
1000 runs of 1 block/1024 threads
=> 1000 different results

Motivations

- Goal:
  - “Offer developers solutions to measure the numerical impact of scheduling even those which are not possible at a given time”.
  - Ability to estimate sensitivity of a code to such phenomena especially when considering legacy property of a code (aptitude to delivers numerically satisfying results on future hypothetical architecture and/or language/compiler

Outline

- Numerical non reproducibility in parallel execution environment
- Execution model
- Proposed solution
  - Accumulator identification
  - Interposer
  - Scheduling amplification
- Results

Outline

- **Numerical non reproducibility in parallel execution environment**
- Execution model
- Proposed solution
  - Accumulator identification
  - Interposer
  - Scheduling amplification
- Results

Numerical non reproducibility in parallel execution environment

- IEEE754 issues
  - Rounding error $1 + 2^{100} = 2^{100}$
  - Non associativity of operations $(1 + 2^{100}) - 2^{100} \neq 1 + (2^{100} - 2^{100})$
  - Issues exacerbate in parallel environments
- Workarounds
  - Reduce numerical error
    - use of large accumulator for the summation problem,
    - compensated algorithm, …
  - Use deterministic execution order either at software/execution level
    - Enforce an order using loop iteration => does not solve all issues (ex: reduction clause in OpenMP)
    - Set an execution order at the execution stage => challenging when no assumption can be made on the execution order (ex: CUDA/OpenCL)

Outline

- Numerical non reproducibility in parallel execution environment
- **Execution model**
- Proposed solution
  - Accumulator identification
  - Interposer
  - Scheduling amplification
- Results

Hardware Execution Model

- Where parallelism is encountered at hardware level
- Level 0: Sub-Word parallelism
  - Register width fixed at architecture level with computational unit supporting multiple operand precision (ex: FP16, FP32)
- Level 1: SWAR (SIMD Within A Register)
  - Pack several data within a single register (ex: ARM SVE range from 128 to 2048 bits)
  - Cross lane instruction with swizzling ability
- Level 2: Warp
  - Instruction stream are scheduled simultaneously across processing unit of 1+ SIMD blocks to form a subgroup called wave/warp (ex: 32 for Nvidia, 64 for AMD) Level 2: Warp
- Level 3: SIMT
  - On some architectures (ex: GPU) parallelism is exploited using array processing (vs vector processing)
  - Spacio-Temporal SIMT corresponds to the issuing of the same instruction multiple time but for different set of invocations
- Level 4: SMT
  - Ability to schedule instructions from another wave in order to hide long-latency inst.
- Level 5: Core
  - Processor embed several copy of the same hardware (SM in CUDA terminology)

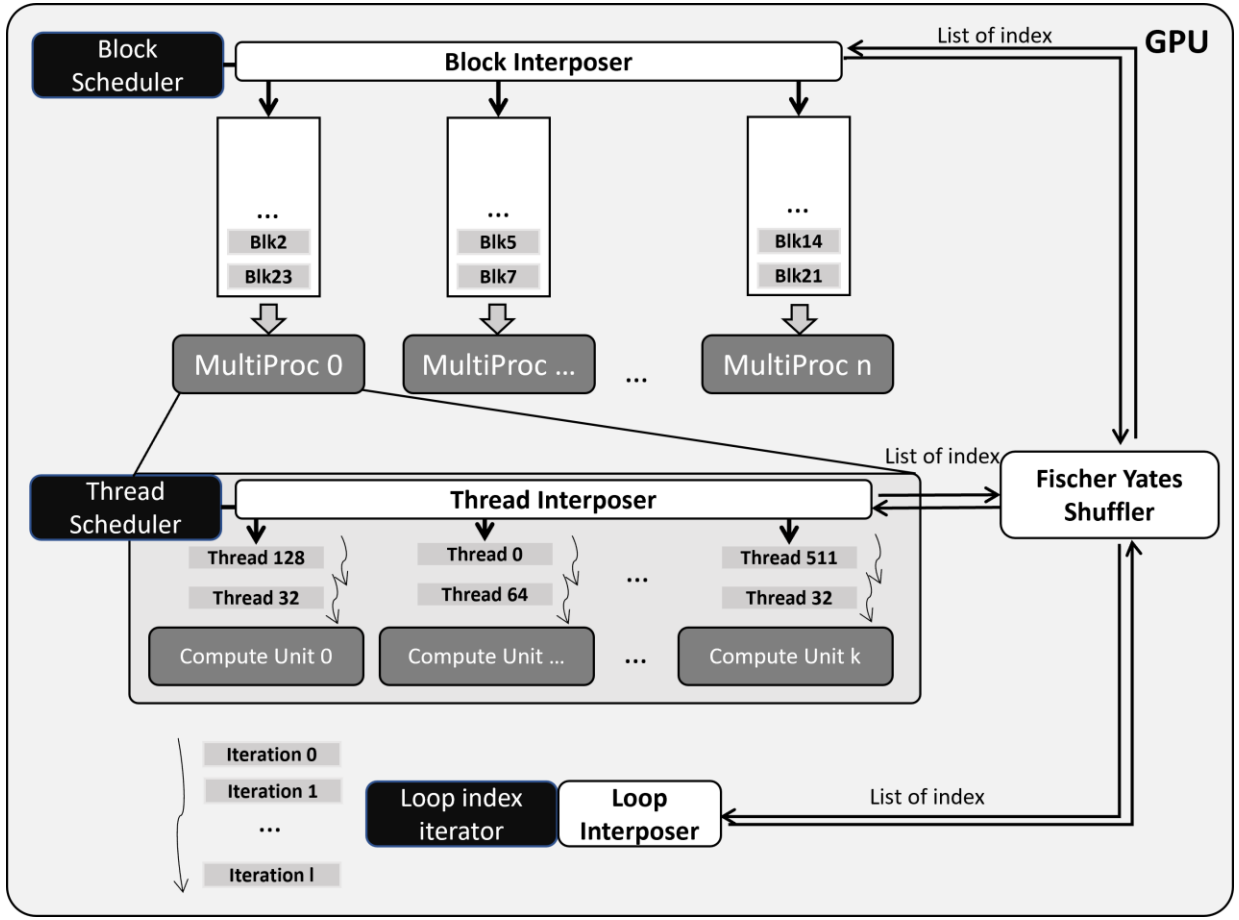
Software Execution Model

- How parallelism is exposed and exploited at software level
  - OpenMP/Threads:
    - Parallelization based on threads (parallel regions), tasks(implicit/explicit), work sharing (parallel construct), accelerator offloading, SIMD lanes
    - Synchronization : barriers, reductions, task dependencies, task wait, lock, critical sections
    - Number of threads, usage of middleware, usage of FP specific reduction clause
    - State of the art: tools to help detecting data race (dynamic detection, hybrid, static analysis…)
  - CUDA/OpenCL
    - Work distribution limited to thread and block identifier scheduling
    - At software level, grid of block of thread
    - Limited support of synchronization primitive

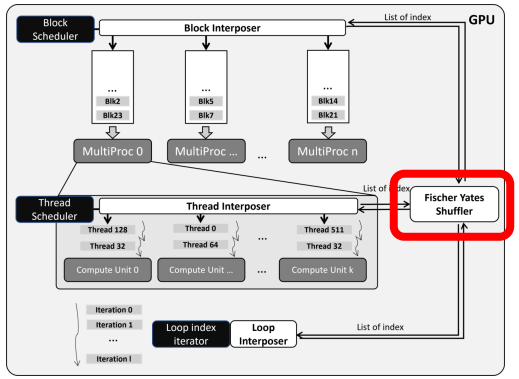
Outline

- Numerical non reproducibility in parallel execution environment
- Execution model
- **Proposed solution**
  - Accumulator identification
  - Interposer
  - Scheduling amplification
- Results

Proposed architecture

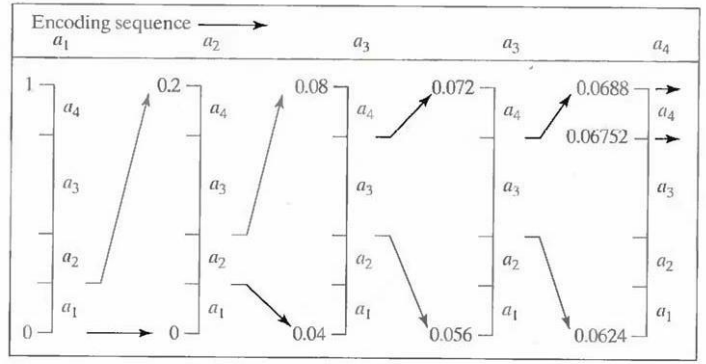


Impact of the entropy amplifier



- Generation of random permutation
  - Requires efficient parallel random generation number
  - Based on Fischer-Yates shuffler (Sattolo's variant => unbiaised permutation)

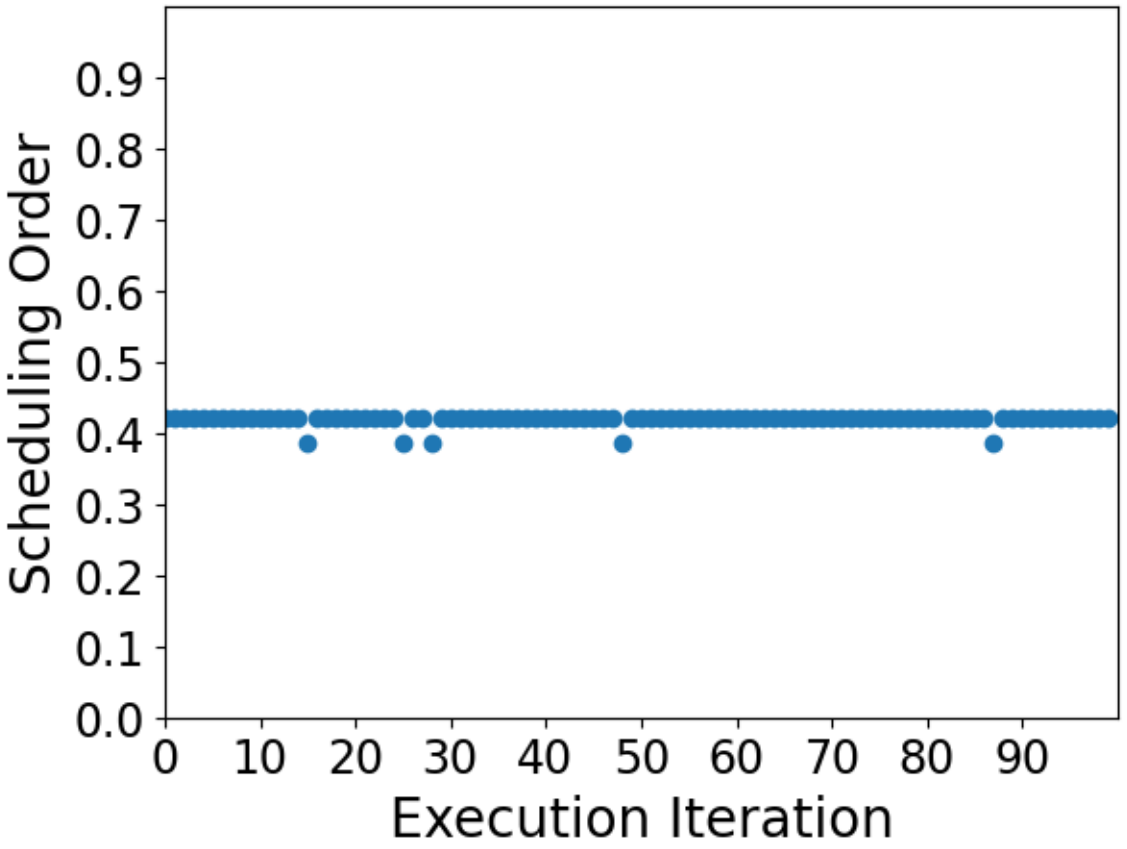
- Measure of the entropy
  - Plot scheduling as a FP number $\in [0,1]$
  - Use arithmetic encoding of the sequence of index identifier according to their execution order

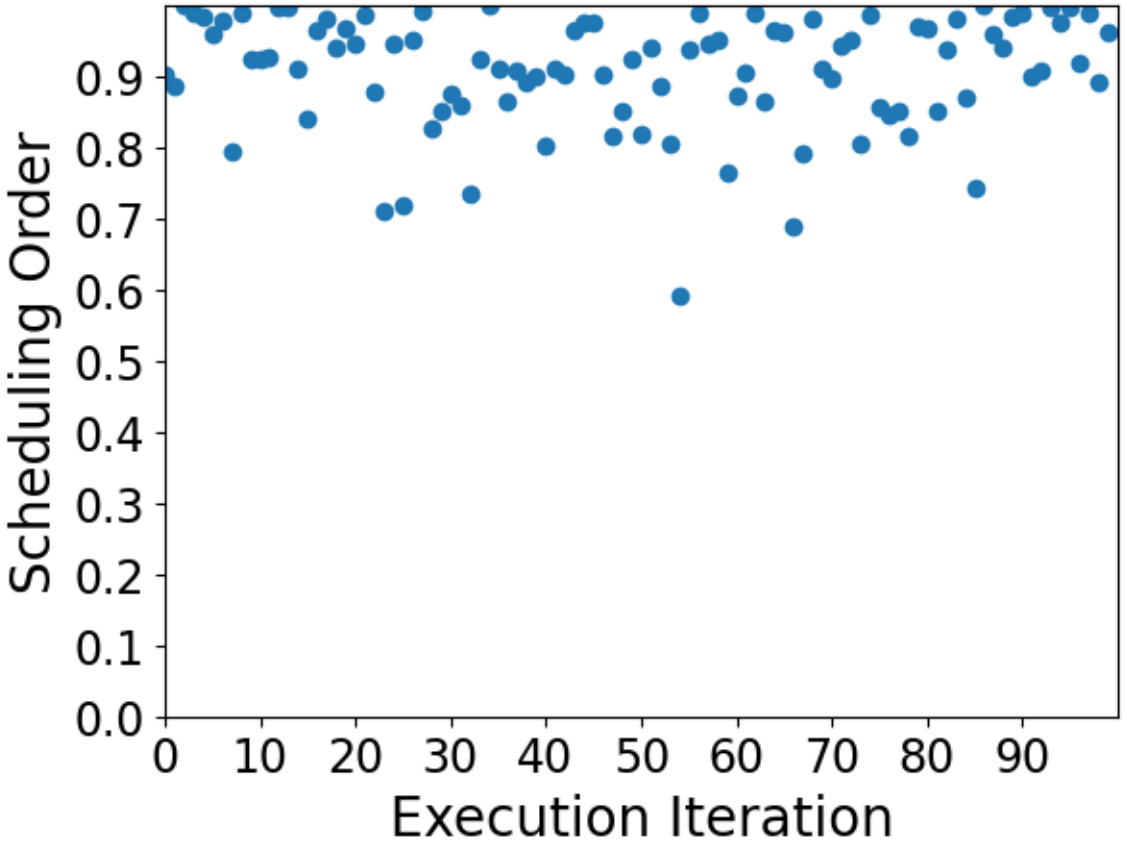


Outline

- Numerical non reproducibility in parallel execution environment
- Execution model
- Proposed solution
  - Accumulator identification
  - Interposer
  - Scheduling amplification
- **Results**

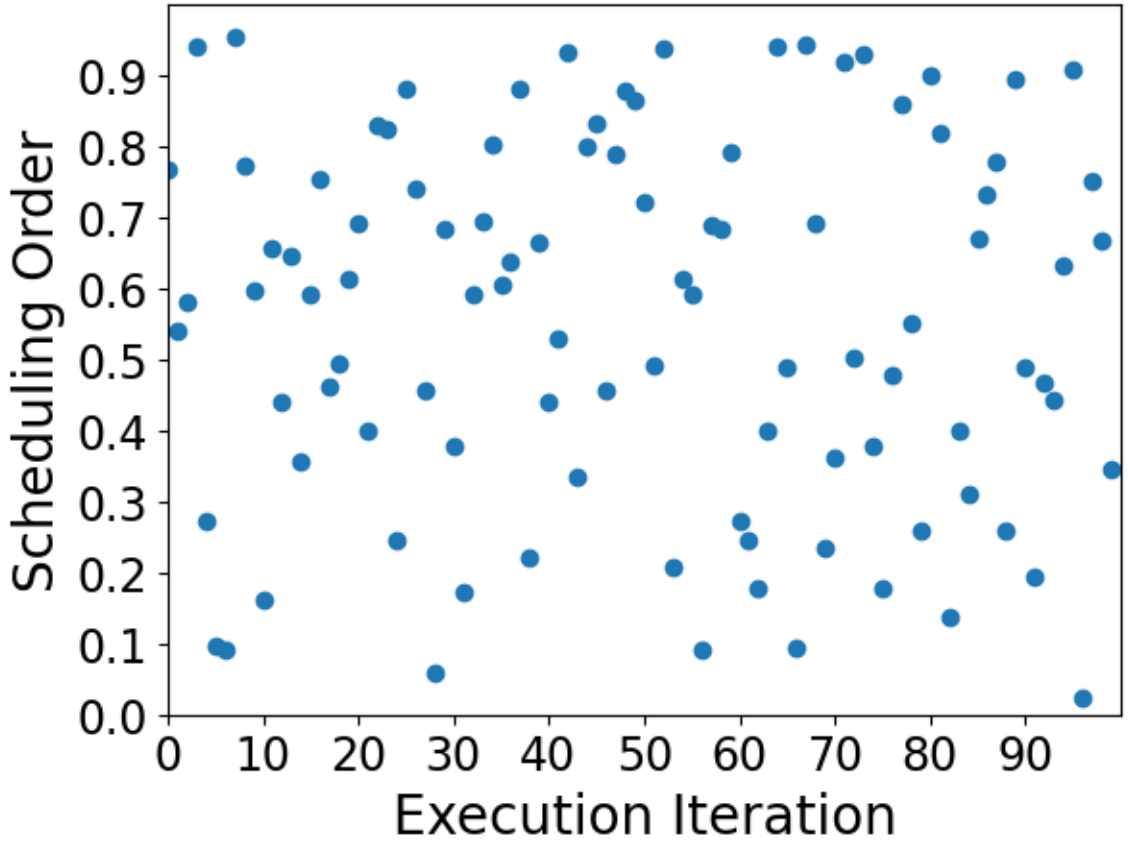
Results: scheduling comparison

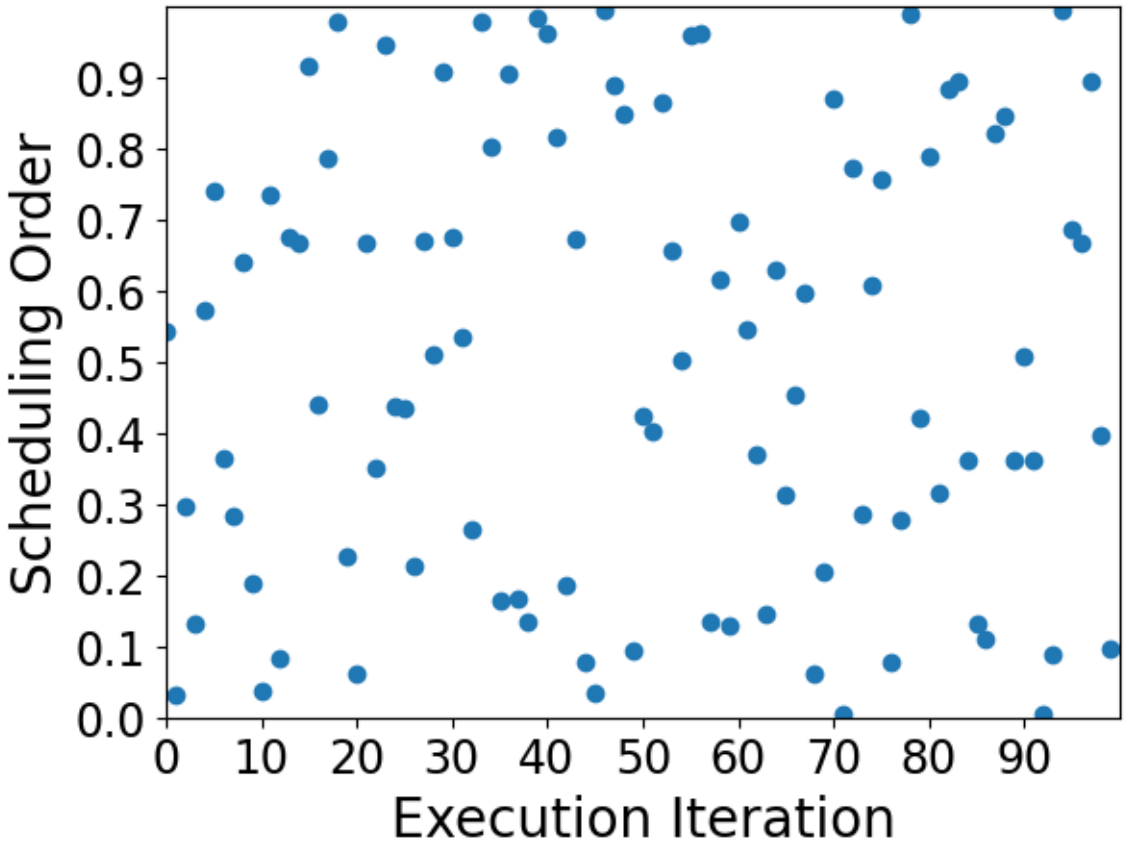




Measure of the entropy for block scheduling **with the hardware scheduler solely** for 8 (first column) and 320 (second column) launched block over 100 executions

Results: scheduling comparison





Measure of the entropy for block scheduling **with the entropy amplifier**, for 8 (first column) and 320 (second column) launched block over 100 executions

Evaluation on benchmark

- Selected benchmark from SHOC, PARBOIL
  - Spot at least one accumulator & use the interposer to interface the Block/Thread/Loop Index with the amplifier
- Hardware platform
  - Intel Xeon E5645, 6 cores,
  - Nvidia RTX2060, 30 SM
- 4 execution configurations
  1. Use the same execution configuration for each run
  2. Change the execution configuration (number of threads, block, ...) between each run
  3. Change work scheduling between each run
  4. Change loop indexing between each run

Name	Description
Cutcp	Compute short-range electrostatic potentials induced by point charges in a 3D volume.
LBM	Lid-Driven Cavity fluid dynamics simulation using the Lattice-Boltzmann Method.
MRI-Q	Matrix computation used in a 3D magnetic resonance image reconstruction.
MRI-GRID	Compute for a 3D grid the contribution of every data point onto its neighboring grid points.
SGEMM	A register-tiled matrix-matrix multiplication, with default column-majored layout.
SPMV	Sparse-Matrix Dense-Vector Product.
Stencil	Seven point stencil.
MD	Molecular dynamics.
Reduction	Sum reduction operation using single precision floating-point data.
NeuralNet	Training of neural networks using backpropagation.

Results

Observed numerical variability over 100 runs for various CUDA/OpenMP programs according to 4 different execution configurations.

	Conf. 1		Conf. 2		Conf. 3		Conf. 4	
	error	# diff.	error	# diff.	error	# diff.	error	# diff.
cuda_cutcp	0	0	-	-	0	0	7,52E+00	12482
omp_cutcp	3,02E-01	11970	4,70E-01	12253	7,05E-01	12135	-	-
cuda_MRI-GRID	0	0	0	0	0	0	5,65E-03	59
omp_MRI-GRID	0	0	9,83E-05	23	7,27E-04	32	1,6	87
cuda_MRI-Q	0	0	0	0	0	0	1,23E+00	64427
omp_MRI-Q	0	0	0	0	0	0	1,17E+00	64461
cudasdk_Reduction	0	0	7,30E-03	9	0	0	9,24E-02	100
cudasdk_Jacobi	0	0	4,79E-03	3	6,90E-01	100	8,87	100
cuda_SGEMM	0	0	0	0	0	0	1,75E-03	20482
omp_SGEMM	0	0	0	0	0	0	1,75E-03	20482
cuda_SPMV	0	0	0	0	0	0	3,69E-06	11948
omp_SPMV	0	0	0	0	0	0	1,16E-07	11948
(cuda/omp)_Stencil	0	0	0	0	0	0	0	0
(cuda/omp)_lbm	0	0	0	0	0	0	0	0

4 execution configurations

1. Use the same execution configuration for each run
2. Change the execution configuration (number of threads, block, ...) between each run
3. Change work scheduling between each run
4. Change loop indexing between each run

Conclusion

- POC demonstrating that Amplifying numerical variability
  - Depend on the coding style and is not an easy task
  - Useful to anticipate potential issues of a given code on future system

- Future work
  - Limited to accumulation what about other errors ?
  - What about MPI, Matrix Ops, Simdization