ESCAPE 824064

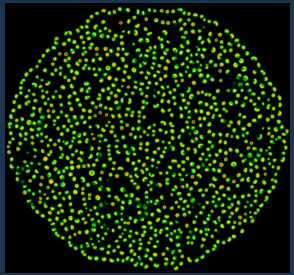
LAPP
Laboratoire d'Annecy de Physique des Particules

Use NVidia HPC SDK on MUST

Pierre Aubert

School computing example

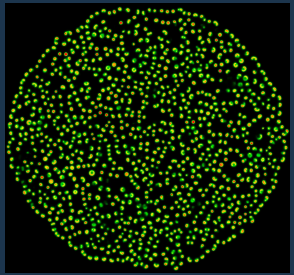
Gray Scott reaction (a chemistry game of life)
(for **CNRS 2023 Computing School**)



School computing example

Gray Scott reaction (a chemistry game of life)
(for **CNRS 2023 Computing School**)

$$U + 2V \longrightarrow 3V$$
$$V \longrightarrow P$$



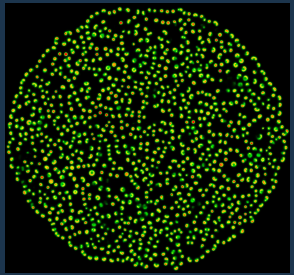
School computing example

Computing :

Gray Scott reaction (a chemistry game of life)
(for **CNRS 2023 Computing School**)

$$\begin{aligned} U + 2V &\longrightarrow 3V \\ V &\longrightarrow P \end{aligned}$$

$$\begin{aligned} \frac{\partial u}{\partial t} &= r_u \nabla^2 u - uv^2 + f_r \times (1 - u) \\ \frac{\partial v}{\partial t} &= r_v \nabla^2 v + uv^2 - (f_r - k_r) \times v \end{aligned}$$



- u and v are concentration of product **U** and **V**
- r_u and r_v diffusion rate of **U** and **V**
- k_r (**Kill Rate**), conversion rate from **V** to **P**
- f_r (**Feed Rate**), speed of process which feed **U** and kills **V** and **P**
- $\nabla^2 u$ and $\nabla^2 v$ are différence of space concentration between current cell and its neighbours

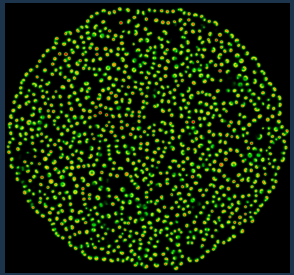
School computing example

Gray Scott reaction (a chemistry game of life)
(for **CNRS 2023 Computing School**)

$$U + 2V \longrightarrow 3V$$
$$V \longrightarrow P$$

Computing :

$$\frac{\partial u}{\partial t} = r_u \nabla^2 u - uv^2 + f_r \times (1 - u)$$
$$\frac{\partial v}{\partial t} = r_v \nabla^2 v + uv^2 - (f_r - k_r) \times v$$



- u and v are concentration of product **U** and **V**
- r_u and r_v diffusion rate of **U** and **V**
- k_r (**Kill Rate**), conversion rate from **V** to **P**
- f_r (**Feed Rate**), speed of process which feed **U** and kills **V** and **P**
- $\nabla^2 u$ and $\nabla^2 v$ are différence of space concentration between current cell and its neighbours

- Easy to understand
- Not so easy for the compiler
- Possibility of high speed up

Computation exercices

- Compute $1000 \times 34 = 34\,000$ images 1920×1080 **float**, store 1000 in **HDF5** file (8.3 GB).
- Evaluate full computation with **time**
- Use any tool for unit performance tests
- Converter from **HDF5** to **png** is provided
- Any optimisation trick can be shared (for example a really efficient auto-vectorized implementation)

Computation exercices

- Compute $1000 \times 34 = 34\,000$ images 1920×1080 **float**, store 1000 in **HDF5** file (8.3 GB).
- Evaluate full computation with **time**
- Use any tool for unit performance tests
- Converter from **HDF5** to **png** is provided
- Any optimisation trick can be shared (for example a really efficient auto-vectorized implementation)

For **C++20 lecture** `https://lappweb.in2p3.fr/~paubert/PERFORMANCE_WITH_STENCIL/index.html`

- **2h 43min 30s** : **-O3** single-thread
- **59min 21s** : naive **intrinsics** single-thread, AVX2
- **9min 49s** : **intrinsics** per block single-thread, AVX2
- **6min 41s** : **intrinsics** per block (AVX2, 8 threads)

For **Cuda Quadro M2200** (1024 cores, 4 GB Mem) :

- **4min 48s** : implementation with temporary)
- **6min 48s** : implementation without temporary)

Computation exercices

- Compute $1000 \times 34 = 34\,000$ images 1920×1080 **float**, store 1000 in **HDF5** file (8.3 GB).
- Evaluate full computation with **time**
- Use any tool for unit performance tests
- Converter from **HDF5** to **png** is provided
- Any optimisation trick can be shared (for example a really efficient auto-vectorized implementation)

For **C++20 lecture** `https://lappweb.in2p3.fr/~paubert/PERFORMANCE_WITH_STENCIL/index.html`

- **2h 43min 30s** : **-O3** single-thread
- **59min 21s** : naive **intrinsics** single-thread, AVX2
- **9min 49s** : **intrinsics** per block single-thread, AVX2
- **6min 41s** : **intrinsics** per block (AVX2, 8 threads)

For **Cuda Quadro M2200** (1024 cores, 4 GB Mem) :

- **4min 48s** : implementation with temporary)
- **6min 48s** : implementation without temporary)

To get same results as **C++20 lecture** :

- **diffusion rate** : $r_u = 0.1$ and $r_v = 0.05$
- **Kill Rate** : $k_r = 0.054$
- **Feed Rate** : $f_r = 0.014$
- $dt = 1$

$$\nabla^2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

GPU MUST

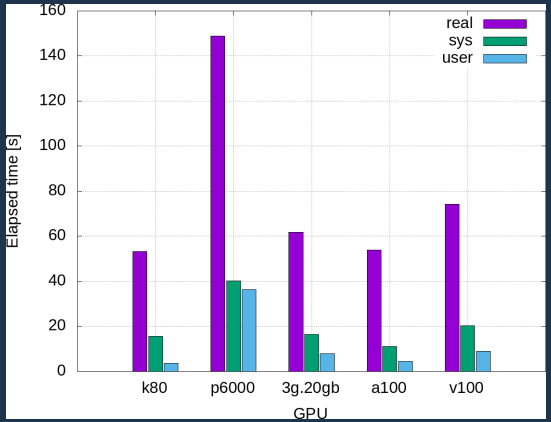
	K80	P6000	T4	V100	A100	3G.20GB
TFlops (float)	8.73 (boost)	12.6	8.1	14	19.5	9.75
Memory (GB)	11.441 (24)	24	15	16	40	20
Nb Cuda Cores	2496 (4992)	3840	2560	5120	6912	2688
Clock rate (GHz)	0.824	1.645	1.590	1.380	1.410	1.410
Generation	3.7	6.1	7.5	7.0	8.0	8.0

Computation $1\,000 \times 34 = 34\,000$ images

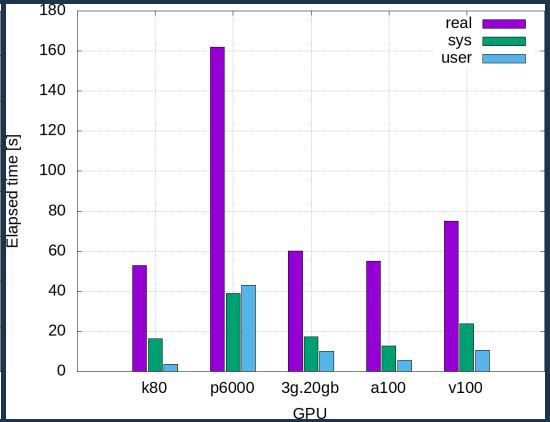
I/O

Result with 100 tests per **GPU**

With temporary

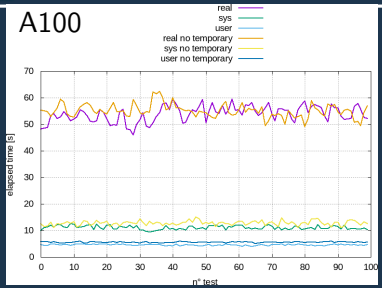


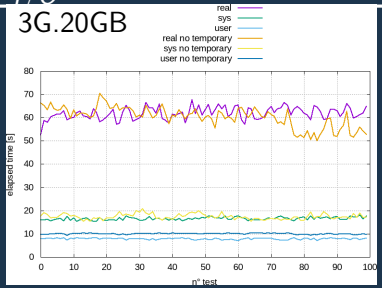
Without temporary

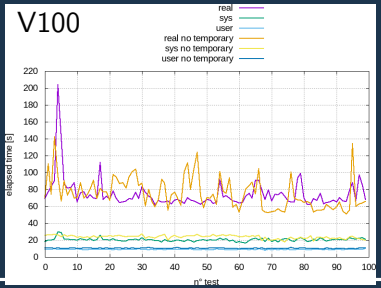


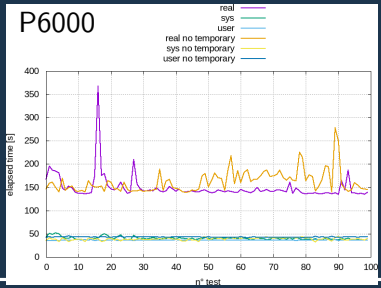
Detail computation $1\,000 \times 34 = 34\,000$ images I/O

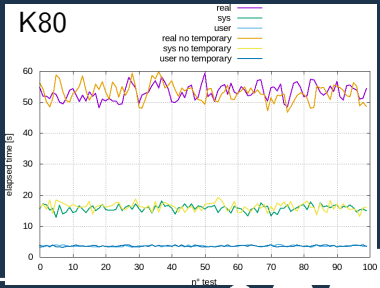
Started : 15/02/2022 11:46:39







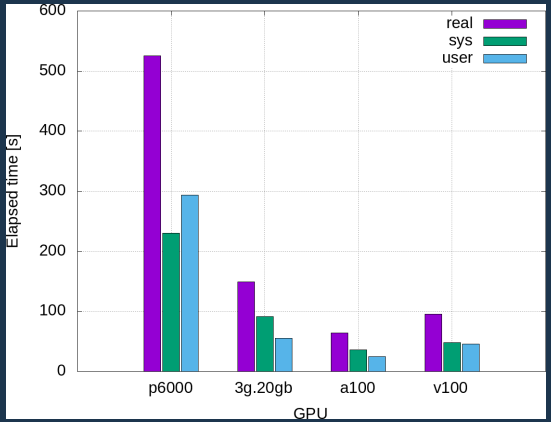




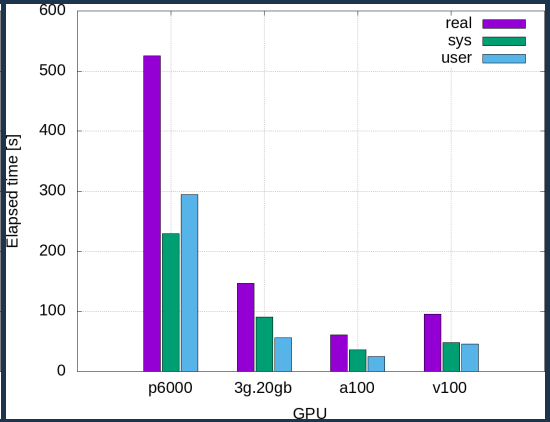
Computation $\underset{I/O}{5} \times 68\,000 = 340\,000$ **images**

Result with 100 tests per **GPU**

With temporary

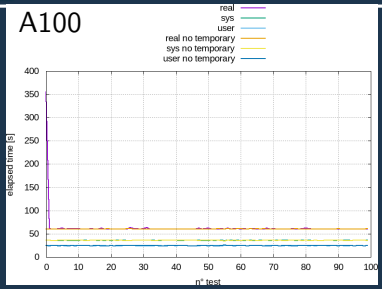


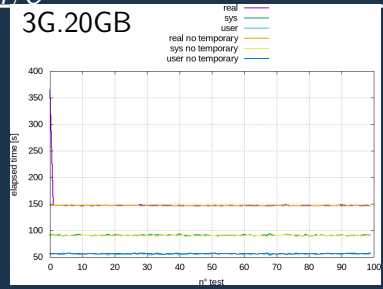
Without temporary

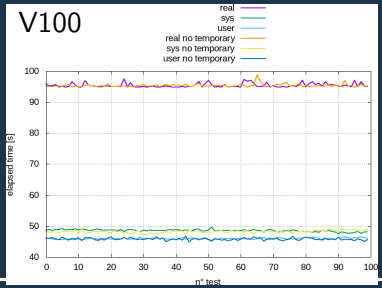


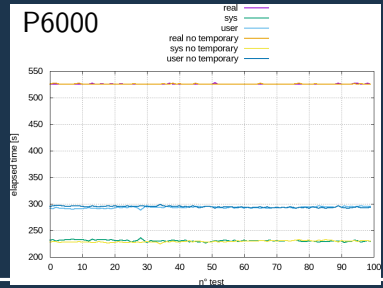
Detail computation $5 \times 68\,000 = 340\,000$ images

Started : 16/02/2022 21:51:57





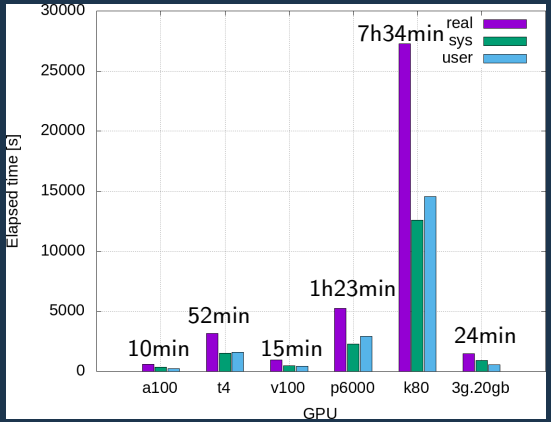




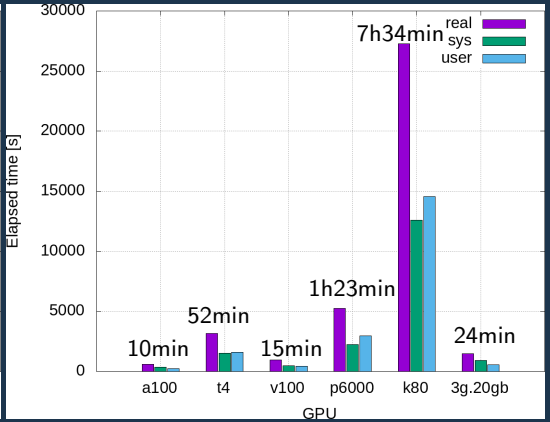
Computation $\frac{5}{I/O} \times 680\,000 = 3\,400\,000$ images

Result with 100 tests per **GPU**

With temporary

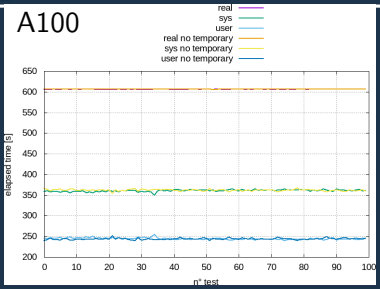


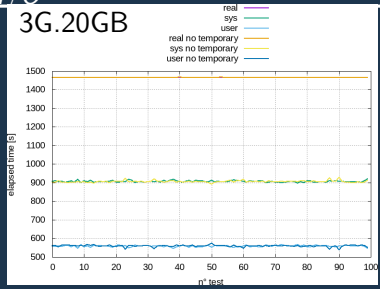
Without temporary

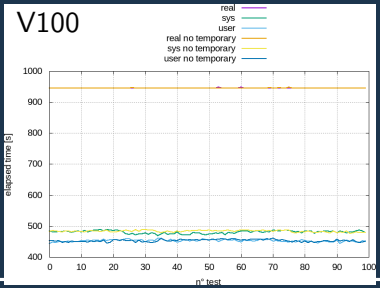


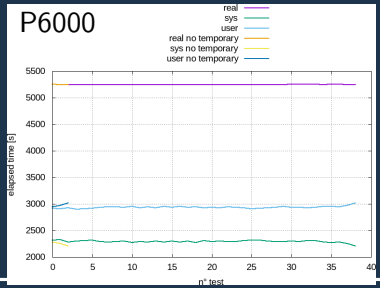
Detail computation $5 \times 680\,000 = 3\,400\,000$ images

Started :
18/02/2022
11:39:39



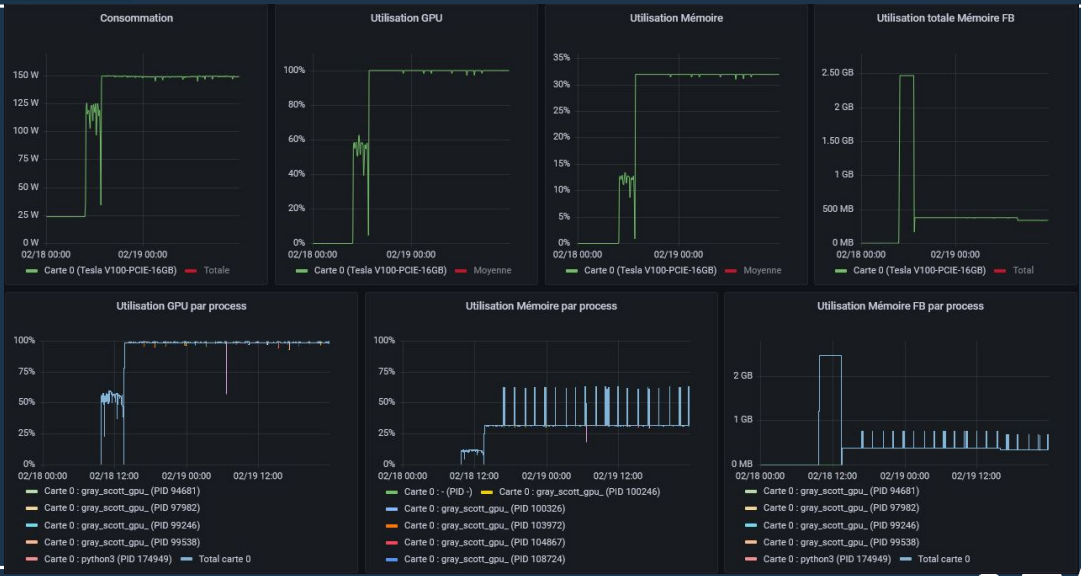






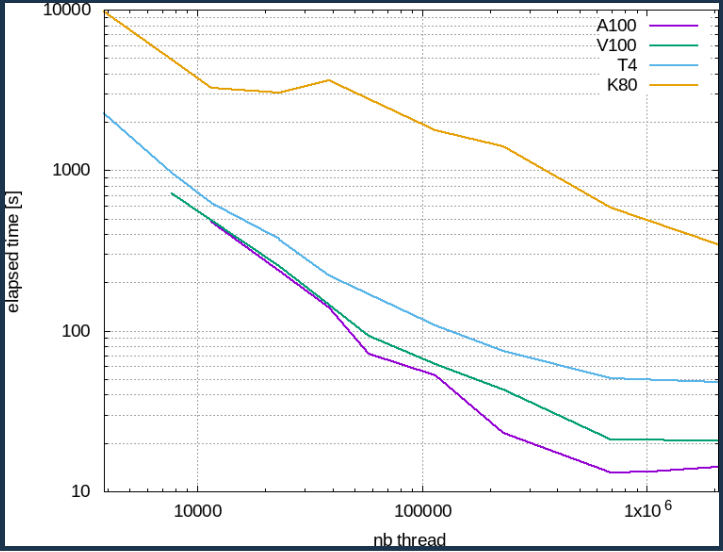
Graphana perf A100 for 3 400 000 images

Graphana perf V100 for 3 400 000 images

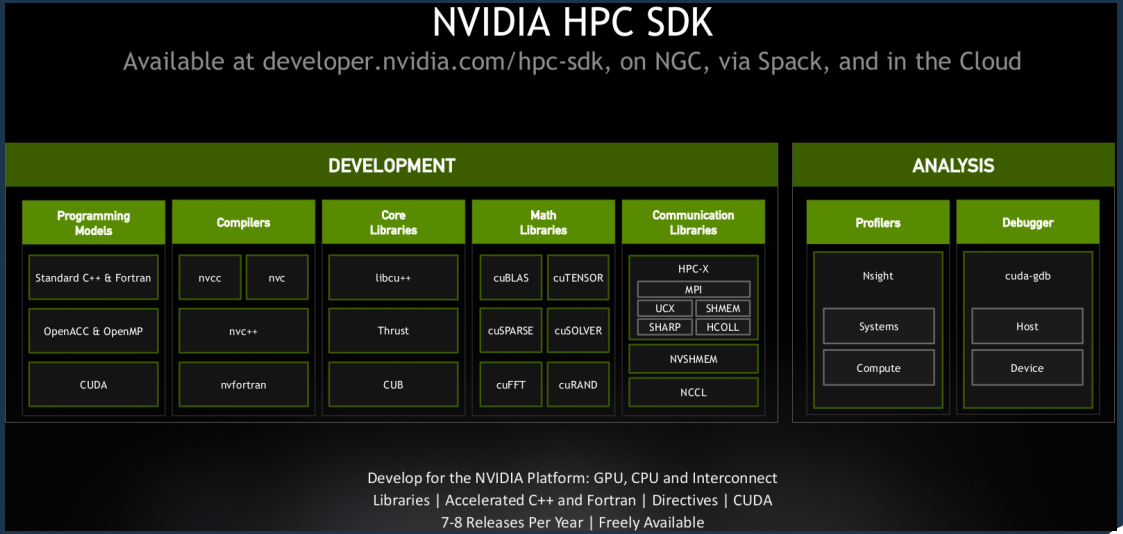


Graphana perf P6000 for 3 400 000 images

Perf VS nb threads for 340 000 images



NVidia HPC SDK introduction



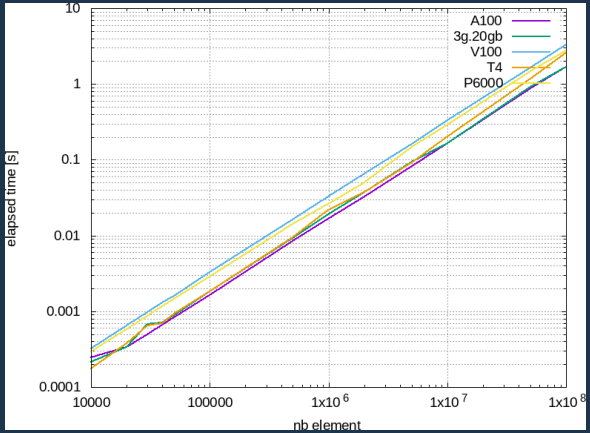
GPU MUST and HPC SDK

Use directly **C++17** to use **GPU** with **NVC++** :

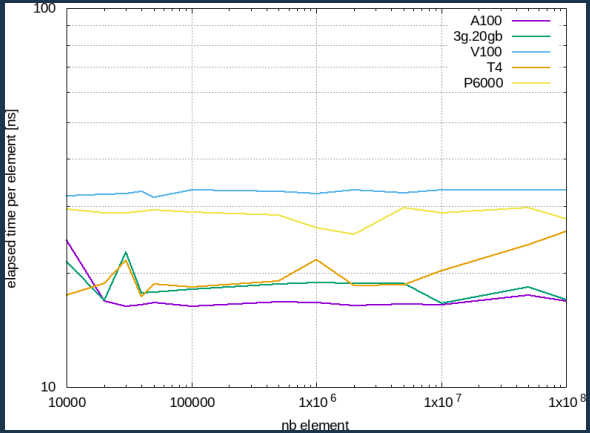
- Only for compute capabilities ≥ 6.0
- Can specify only one compute capability at compilation time
- Only for **C++17** (working with **G++-9 or newer**)
- Parallelism only with **TBB 2018** or newer (not on **CentOS 7**)

Hadamard Product with nvc++ (std::par cc60)

Full elapsed time [s]

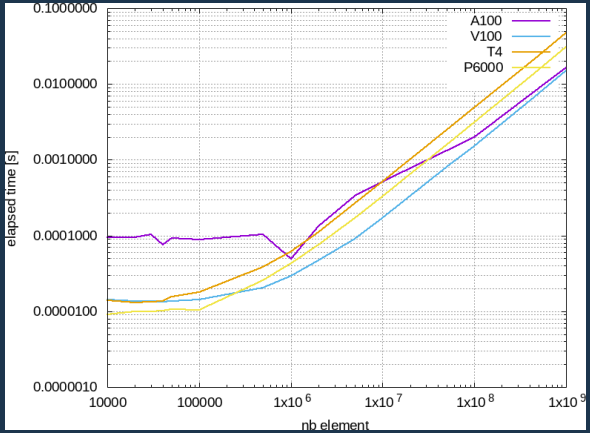


Elapsed time per element [ns]

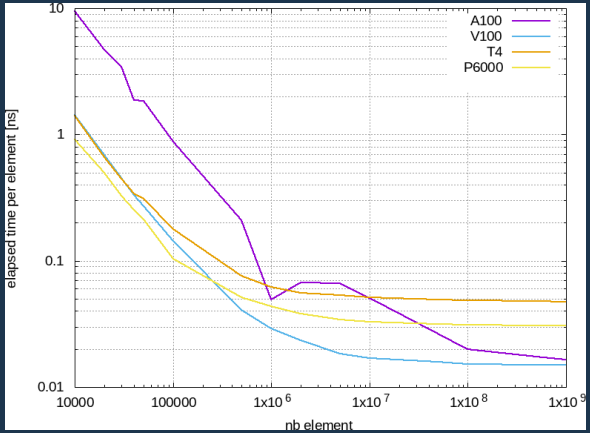


Hadamard Product nvc++ (std::par_unseq cc optimal)

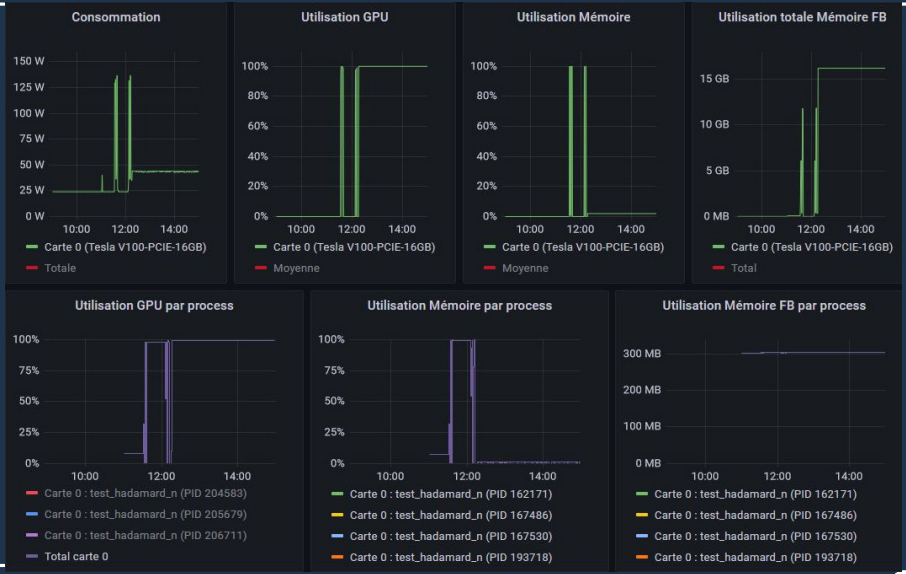
Full elapsed time [s]



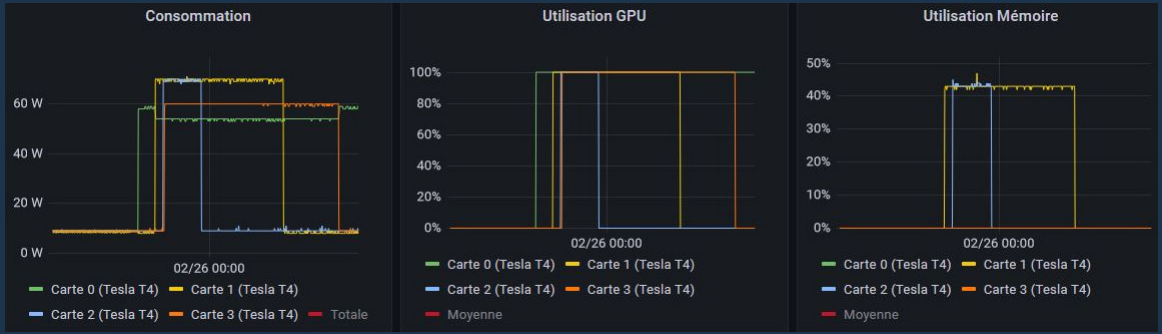
Elapsed time per element [ns]



Graphana V100 for Hadamard nvc++



Graphana perf T4



Conclusion

- Good performances on **GPU**
  - With **nvcc** (**CUDA**)
  - With **nvc++**
- **HPC SDK** installed on **MUST**
- Compiler **nvc++** powerful and easy to use
  - No explicit linking
  - No **GPU** Targeting (or with **CUDA_VISIBLE_DEVICE**)

- Warning about industrial software
  - Will to drive for update
  - Old **GPU** become **obsolete** :
    - **nvc++** : compute capabilities ≥ 6 (no **K80**)
    - **nvcc** : compute capabilities ≥ 3.5
  - Need to save binaries to ensure long usability of **GPU**