

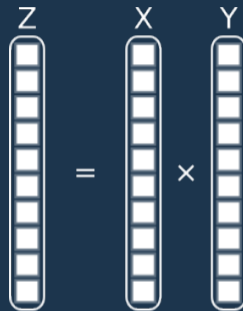
# The Hadamard Product : With NaN and Denorm

Pierre Aubert



# The Hadamard product

$$z_i = x_i \times y_i, \quad \forall i \in 1, N$$



```
for(long unsigned int i(0lu); i < nbElement; ++i){
    >>     tabResult[i] = tabX[i]*tabY[i];
}
```

# Compilation options

---

<https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>

# Compilation options

---

<https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>

▶ **-O0**

- ▶ Try to reduce compilation time, but **-Og** is better for debugging.

# Compilation options

<https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>

- ▶ **-O0**
  - ▶ Try to reduce compilation time, but **-Og** is better for debugging.
- ▶ **-O1**
  - ▶ Constant forwarding, remove dead code (never called code)...

<https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>

- ▶ **-O0**
  - ▶ Try to reduce compilation time, but **-Og** is better for debugging.
- ▶ **-O1**
  - ▶ Constant forwarding, remove dead code (never called code)...
- ▶ **-O2**
  - ▶ Partial function inlining, Assume strict aliasing...

<https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>

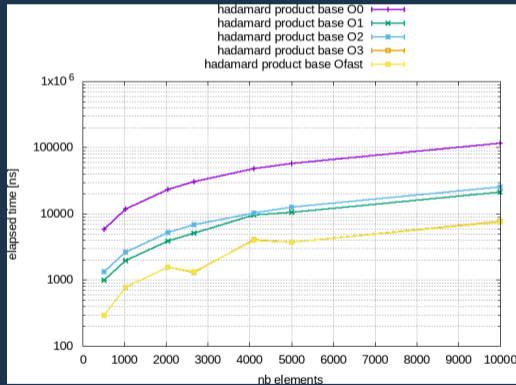
- ▶ **-O0**
  - ▶ Try to reduce compilation time, but **-Og** is better for debugging.
- ▶ **-O1**
  - ▶ Constant forwarding, remove dead code (never called code)...
- ▶ **-O2**
  - ▶ Partial function inlining, Assume strict aliasing...
- ▶ **-O3**
  - ▶ More function inlining, loop unrolling, partial vectorization...

<https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>

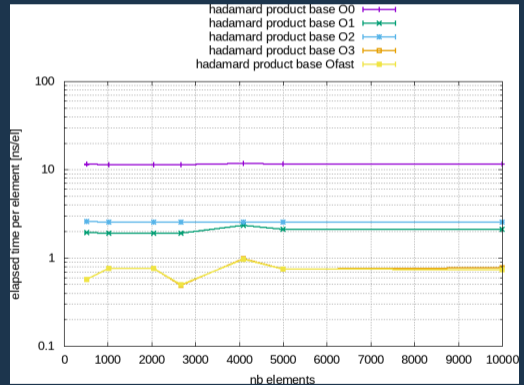
- ▶ **-O0**
  - ▶ Try to reduce compilation time, but **-Og** is better for debugging.
- ▶ **-O1**
  - ▶ Constant forwarding, remove dead code (never called code)...
- ▶ **-O2**
  - ▶ Partial function inlining, Assume strict aliasing...
- ▶ **-O3**
  - ▶ More function inlining, loop unrolling, partial vectorization...
- ▶ **-Ofast**
  - ▶ Disregard strict standards compliance. Enable **-ffast-math**, stack size is hardcoded to 32 768 bytes (borrowed from **gfortran**).  
**Possibly degrades the computation accuracy.**



Total Elapsed Time (cy)



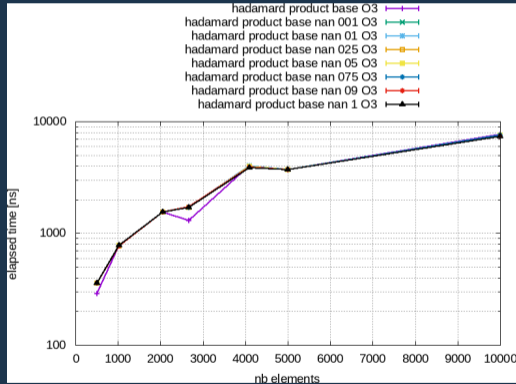
Elapsed Time per element (cy/el)



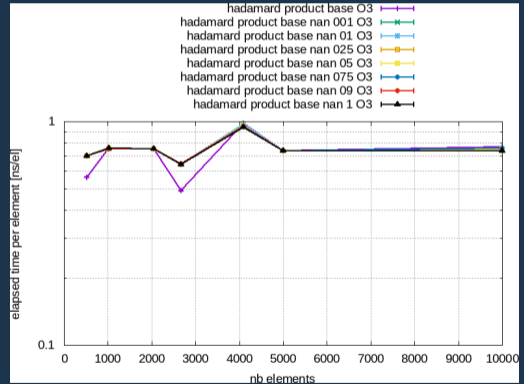
Speed up of 14 between -O0 and -O3 or -Ofast



## Total Elapsed Time (cy)

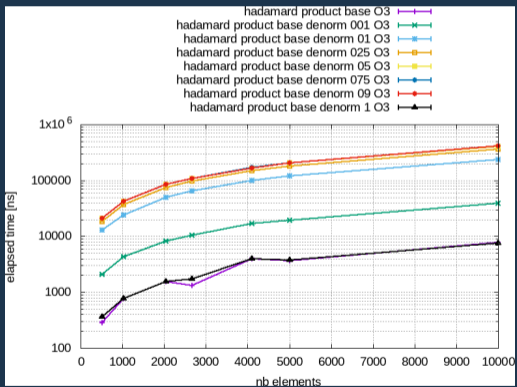


## Elapsed Time per element (cy/el)

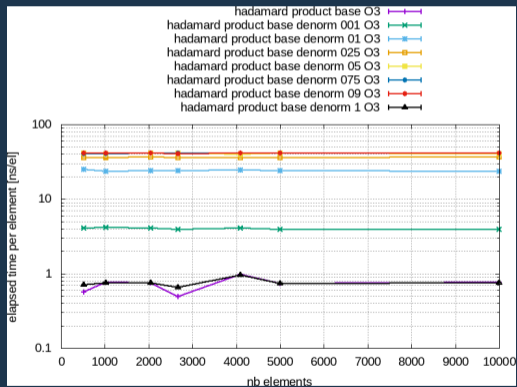


Same performances in -O3

## Total Elapsed Time (cy)

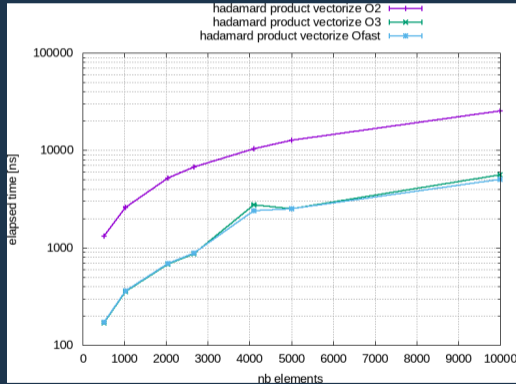


## Elapsed Time per element (cy/el)

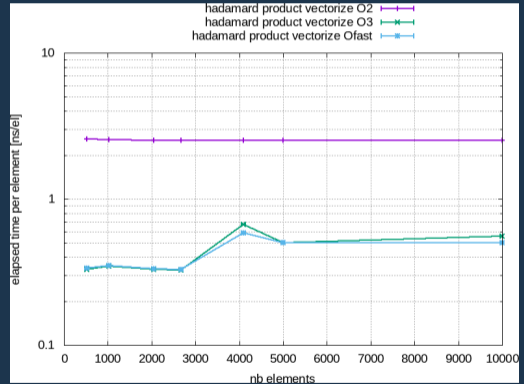


High impact on performances in -O3

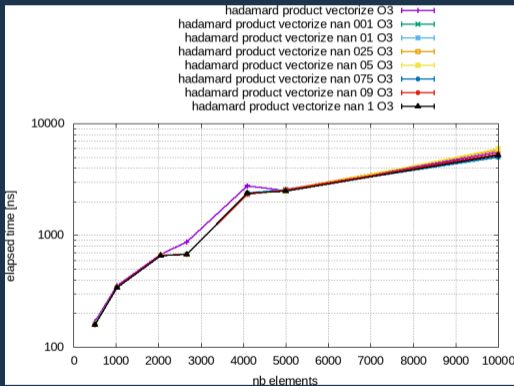
Total Elapsed Time (cy)



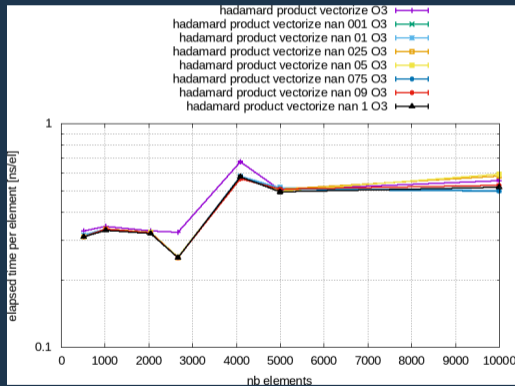
Elapsed Time per element (cy/el)



Total Elapsed Time (cy)

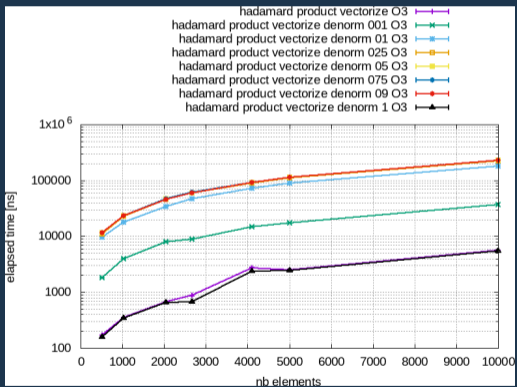


Elapsed Time per element (cy/el)

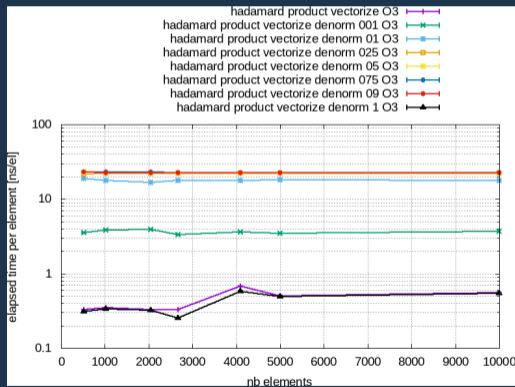


Same performances in -O3

## Total Elapsed Time (cy)

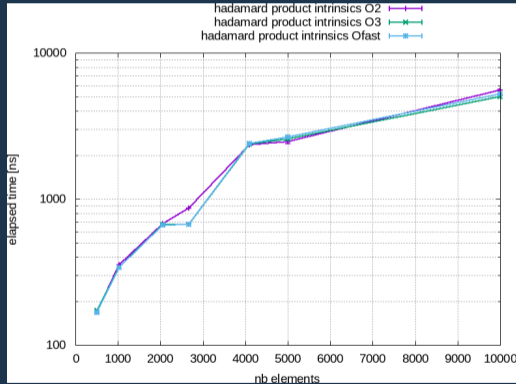


## Elapsed Time per element (cy/el)

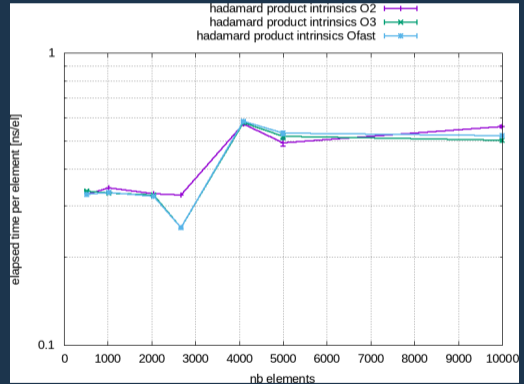


High impact on performances in -O3

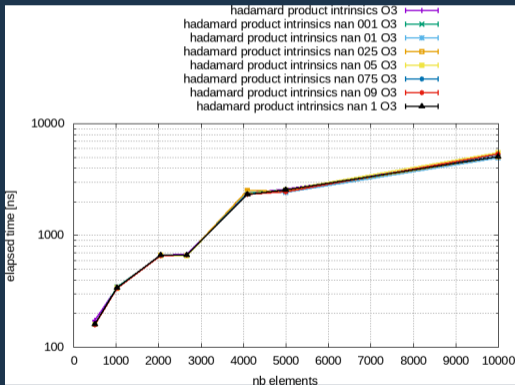
Total Elapsed Time (cy)



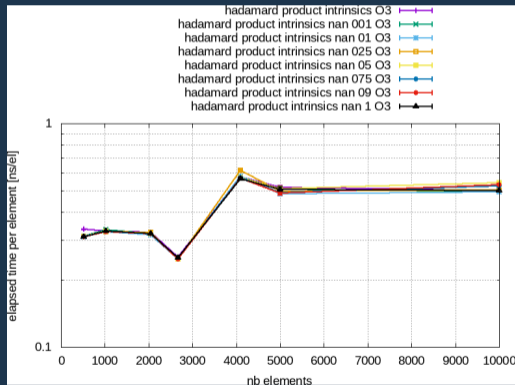
Elapsed Time per element (cy/el)



Total Elapsed Time (cy)



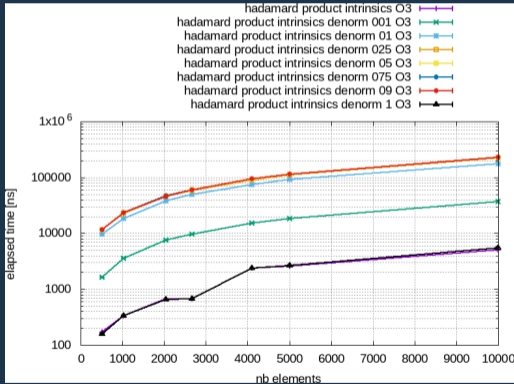
Elapsed Time per element (cy/el)



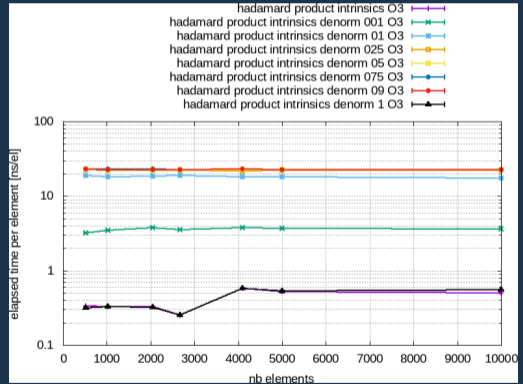
Same performances in -O3



Total Elapsed Time (cy)



Elapsed Time per element (cy/el)



High impact on performances in -O3



NaN values do not slow down the performances

Denormalised values affect a lot the performances :

- ▶ 1% : slow down computation by  $\sim 5.7$
- ▶ 10% : slow down computation by  $\sim 31$
- ▶ 50 – 90% : slow down computation by  $\sim 57$
- ▶ 100% : same performance as 0%

Denormalised values affect a lot the intrinsics performances :

- ▶ 1% : slow down computation by  $\sim 13$
- ▶ 10% : slow down computation by  $\sim 53$
- ▶ 50 – 90% : slow down computation by  $\sim 22$
- ▶ 100% : same performance as 0%

But can we solve this problem ?