LAPP
Laboratoire d'Annecy de Physique des Particules

Optimisation : Reduction

Pierre Aubert

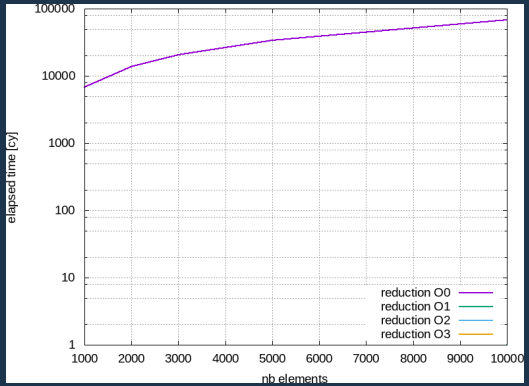
The Reduction (sum)

$$\alpha = \sum_{i=1}^{N} x_i$$

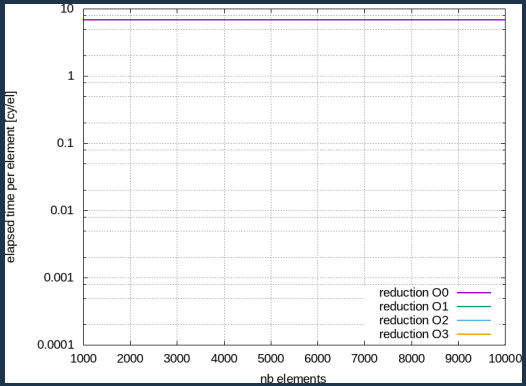

```
float reduction(const float * tabValue, long unsigned int nbElement){
»       float res(0.0f);
»       for(long unsigned int i(0lu); i < nbElement; ++i){
»       »       res += tabValue[i];
»       }
»       return res;
}
```

The reduction : first performances

Total Elapsed Time (cy)

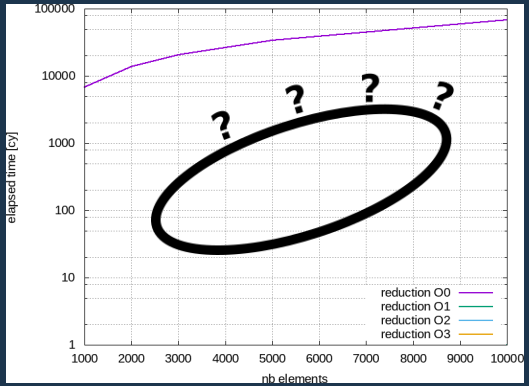


Elapsed Time per element (cy/el)

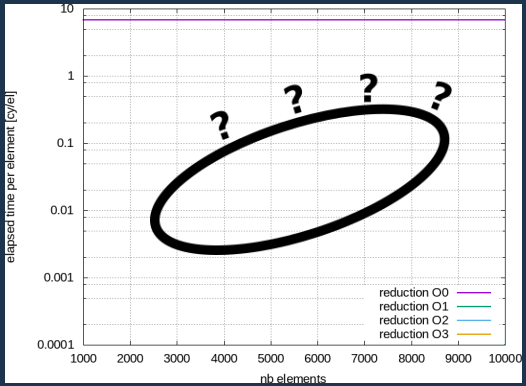


The reduction : first performances

Total Elapsed Time (cy)



Elapsed Time per element (cy/el)



The Performance : what is the issue ?

- Performances **-O0** : slow but reasonable
- Other performances (**-O1**, **-O2**, **-O3**, **-Ofast**) are too fast (non sence)

GCC is smart of guileful depending on the points of view.

- GCC noticed you **do not** use the result of the **reduction** function.
- The call to **reduction** is considered as dead code (or never called code).

To avoid that, you have to compile the **reduction** function in an other file.

Dead code for compiler

```
float reduction(const float * tabValue, long unsigned int nbElement){
	float res(0.0f);
	for(long unsigned int i(0lu); i < nbElement; ++i){
		res += tabValue[i];
	}
	return res;
}

void evaluateReduction(long unsigned int nbElement, long unsigned int nbRepetition){
	float * tabValue = (float*)asterics_malloc(sizeof(float)*nbElement);
	for(long unsigned int i(0lu); i < nbElement; ++i){
		tabValue[i] = (float)(i*32lu%17lu);
	}

	long unsigned int beginTime(rdtsc());
	for(long unsigned int i(0lu); i < nbRepetition; ++i){
		reduction(tabValue, nbElement);
	}
	long unsigned int elapsedTime((double)(rdtsc() - beginTime)/((double)nbRepetition));

	double cyclePerElement(((double)elapsedTime)/((double)nbElement));
	cout << "evaluateReduction : nbElement = "<<nbElement<<", cyclePerElement = " << cyclePerElement << " cy/
	el, elapsedTime = " << elapsedTime << " cy" << endl;
	cerr << nbElement << "\t" << cyclePerElement << "\t" << elapsedTime << endl;

	asterics_free(tabValue);
}
```

Dead code for compiler

```
float reduction(const float * tabValue, long unsigned int nbElement){
	float res(0.0f);
	for(long unsigned int i(0lu); i < nbElement; ++i){
		res += tabValue[i];
	}
	return res;
}

void evaluateReduction(long unsigned int nbElement, long unsigned int nbRepetition){
	float * tabValue = (float*)asterics_malloc(sizeof(float)*nbElement);
	for(long unsigned int i(0lu); i < nbElement; ++i){
		tabValue[i] = (float)(i*32lu%17lu);
	}

	long unsigned int beginTime(rdtsc());
	for(long unsigned int i(0lu); i < nbRepetition; ++i){
		reduction(tabValue, nbElement);
	}
	long unsigned int elapsedTime((double)(rdtsc() - beginTime)/((double)nbRepetition));

	double cyclePerElement(((double)elapsedTime)/((double)nbElement));
	cout << "evaluateReduction : nbElement = "<<nbElement<<", cyclePerElement = " << cyclePerElement << " cy/
	el, elapsedTime = " << elapsedTime << " cy" << endl;
	cerr << nbElement << "\t" << cyclePerElement << "\t" << elapsedTime << endl;

	asterics_free(tabValue);
}
```

Unused result

Dead code for compiler

```
float reduction(const float * tabValue, long unsigned int nbElement){
	float res(0.0f);
	for(long unsigned int i(0lu); i < nbElement; ++i){
		res += tabValue[i];
	}
	return res;
}

void evaluateReduction(long unsigned int nbElement, long unsigned int nbRepetition){
	float * tabValue = (float*)asterics_malloc(sizeof(float)*nbElement);
	for(long unsigned int i(0lu); i < nbElement; ++i){
		tabValue[i] = (float)(i*32lu%17lu);
	}

	long unsigned int beginTime(rdtsc());
	for(long unsigned int i(0lu); i < nbRepetition; ++i){
		reduction(tabValue, nbElement);
	}
	long unsigned int elapsedTime((double)(rdtsc() - beginTime)/((double)nbRepetition));

	double cyclePerElement(((double)elapsedTime)/((double)nbElement));
	cout << "evaluateReduction : nbElement = "<<nbElement<<", cyclePerElement = " << cyclePerElement << " cy/
	el, elapsedTime = " << elapsedTime << " cy" << endl;
	cerr << nbElement << "\t" << cyclePerElement << "\t" << elapsedTime << endl;

	asterics_free(tabValue);
}
```

Unused result

Dead code for compiler

```
float reduction(const float * tabValue, long unsigned int nbElement){
        float res(0.0f);
        for(long unsigned int i(0lu); i < nbElement; ++i){
                res += tabValue[i];
        }
        return res;
}

void evaluateReduction(long unsigned int nbElement, long unsigned int nbRepetition){
        float * tabValue = (float*)asterics_malloc(sizeof(float)*nbElement);
        for(long unsigned int i(0lu); i < nbElement; ++i){
                tabValue[i] = (float)(i*32lu%17lu);
        }

        long unsigned int beginTime(rdtsc());
        for(long unsigned int i(0lu); i < nbRepetition; ++i){
                reduction(tabValue, nbElement);
        }
        long unsigned int elapsedTime((double)(rdtsc() - beginTime)/((double)nbRepetition));

        double cyclePerElement(((double)elapsedTime)/((double)nbElement));
        cout << "evaluateReduction : nbElement = "<<nbElement<<", cyclePerElement = " << cyclePerElement << " cy/
el, elapsedTime = " << elapsedTime << " cy" << endl;
        cerr << nbElement << "\t" << cyclePerElement << "\t" << elapsedTime << endl;

        asterics_free(tabValue);
}
```

Useless function

Unused result

Dead code for compiler

```
float reduction(const float * tabValue, long unsigned int nbElement){
        float res(0.0f);
        for(long unsigned int i(0lu); i < nbElement; ++i){
                res += tabValue[i];
        }
        return res;
}

void evaluateReduction(long unsigned int nbElement, long unsigned int nbRepetition){
        float * tabValue = (float*)asterics_malloc(sizeof(float)*nbElement);
        for(long unsigned int i(0lu); i < nbElement; ++i){
                tabValue[i] = (float)(i*32lu%17lu);
        }

        long unsigned int beginTime(rdtsc());
        for(long unsigned int i(0lu); i < nbRepetition; ++i){
                reduction(tabValue, nbElement);
        }
        long unsigned int elapsedTime((double)(rdtsc() - beginTime)/((double)nbRepetition));

        double cyclePerElement(((double)elapsedTime)/((double)nbElement));
        cout << "evaluateReduction : nbElement = "<<nbElement<<", cyclePerElement = " << cyclePerElement << " cy/
el, elapsedTime = " << elapsedTime << " cy" << endl;
        cerr << nbElement << "\t" << cyclePerElement << "\t" << elapsedTime << endl;

        asterics_free(tabValue);
}
```

Useless function

Useless loop

Unused result

Dead code for compiler

```
float reduction(const float * tabValue, long unsigned int nbElement){
        float res(0.0f);
        for(long unsigned int i(0lu); i < nbElement; ++i){
                res += tabValue[i];
        }
        return res;
}

void evaluateReduction(long unsigned int nbElement, long unsigned int nbRepetition){
        float * tabValue = (float*)asterics_malloc(sizeof(float)*nbElement);
        for(long unsigned int i(0lu); i < nbElement; ++i){
                tabValue[i] = (float)(i*32lu%17lu);
        }

        long unsigned int beginTime(rdtsc());
        for(long unsigned int i(0lu); i < nbRepetition; ++i){
                res += reduction(tabValue, nbElement);
        }
        long unsigned int elapsedTime((double)(rdtsc() - beginTime)/((double)nbRepetition));

        double cyclePerElement(((double)elapsedTime)/((double)nbElement));
        cout << "evaluateReduction : nbElement = "<<nbElement<<", cyclePerElement = " << cyclePerElement << " cy/
el, elapsedTime = " << elapsedTime << " cy" << endl;
        cerr << nbElement << "\t" << cyclePerElement << "\t" << elapsedTime << endl;

        asterics_free(tabValue);
}
```

Useless function

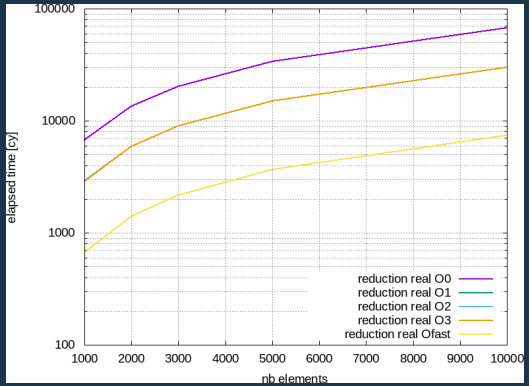
Nothing between timers

Useless loop

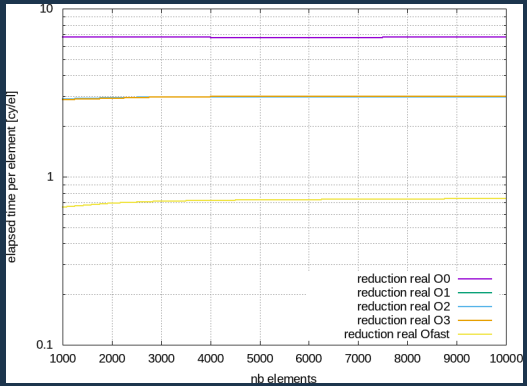
Unused result

The reduction : real performances

Total Elapsed Time (cy)



Elapsed Time per element (cy/el)



Modifications for vectorization

- Data alignement :
  - All the data to be aligned on vectorial registers size.
  - Change **new** or **malloc** to **memalign** or **posix_memalign**

You can use **asterics_malloc** to have LINUX/MAC compatibility (in **evaluateReduction**):

```
(float*)asterics_malloc(sizeof(float)*nbElement);
```

The **__restrict__** keyword (arguments of **reduction** function):

```
const float * __restrict__ ptabValue
```

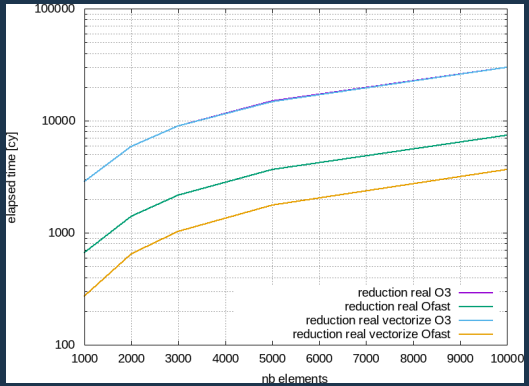
The **__builtin_assume_aligned** function call (in **reduction** function):

```
const float* tabValue = (const float*)__builtin_assume_aligned(ptabValue, VECTOR_ALIGNEMENT);
```

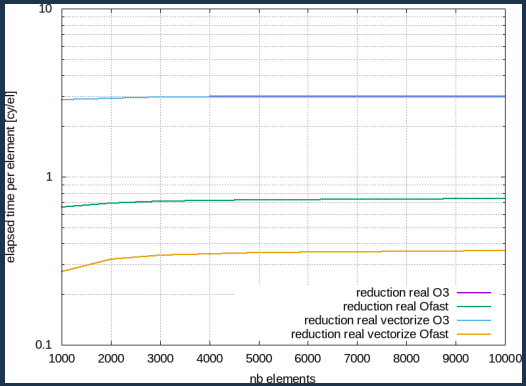
- The Compilation Options become :
  - **-O3 -ftree-vectorize -march=native -mtune=native -mavx2**

The reduction : vectorize performances

Total Elapsed Time (cy)

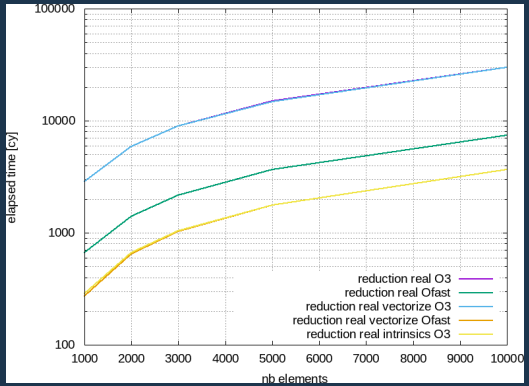


Elapsed Time per element (cy/el)

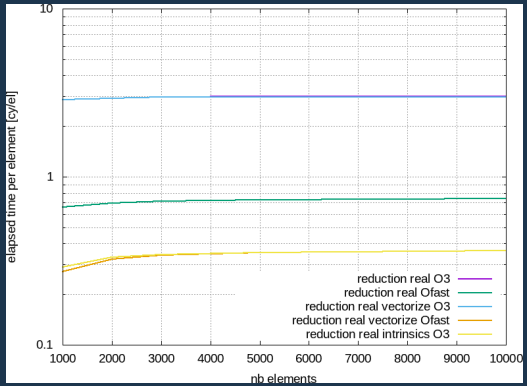


The reduction : intrinsics performances

Total Elapsed Time (cy)

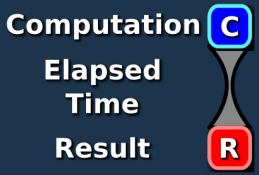


Elapsed Time per element (cy/el)

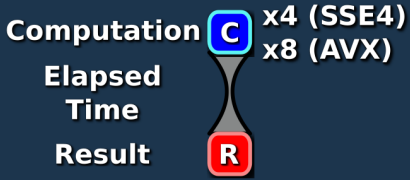


Reduction optimisation

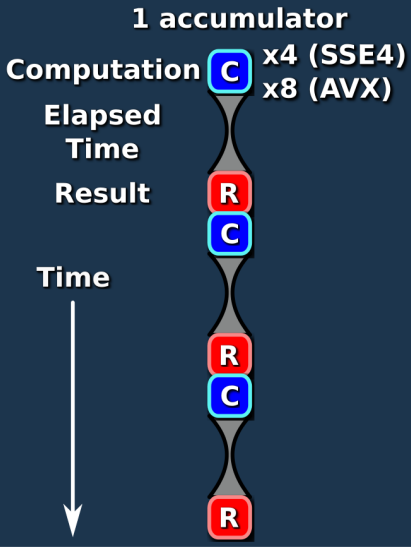
Reduction optimisation



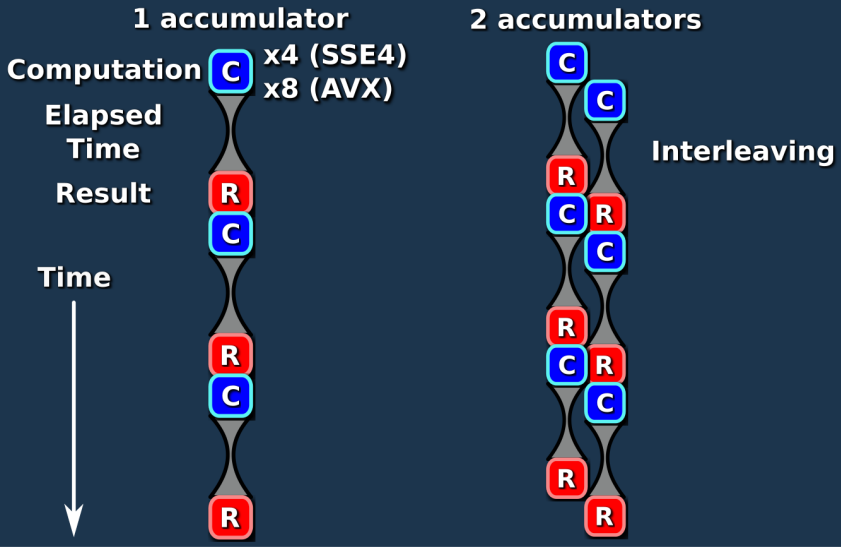
Reduction optimisation



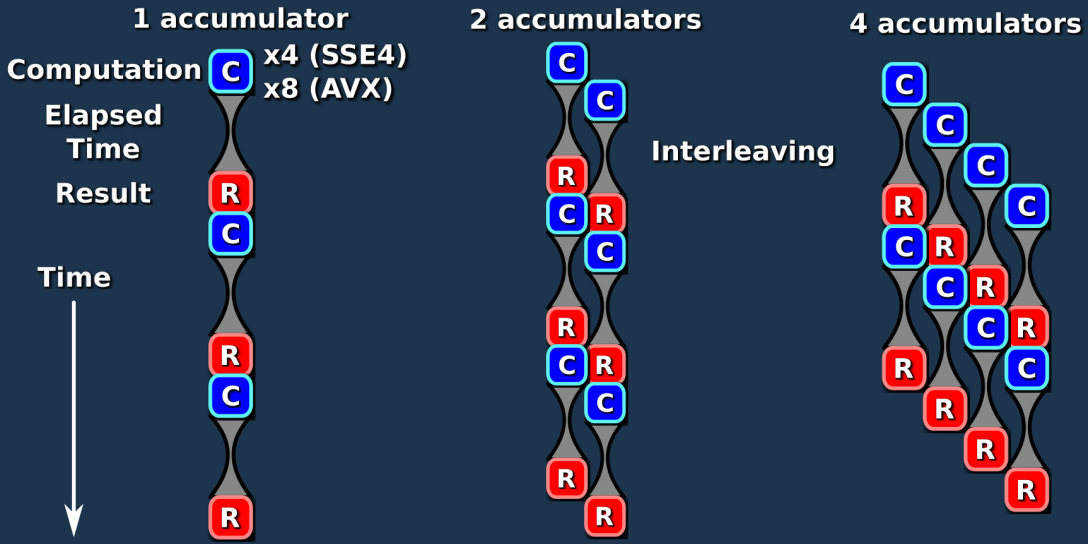
Reduction optimisation



Reduction optimisation

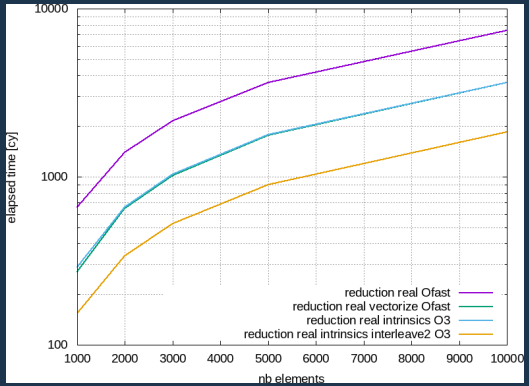


Reduction optimisation

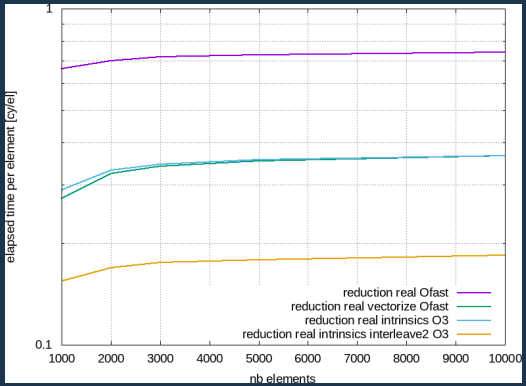


The reduction : intrinsics interleaved 2

Total Elapsed Time (cy)

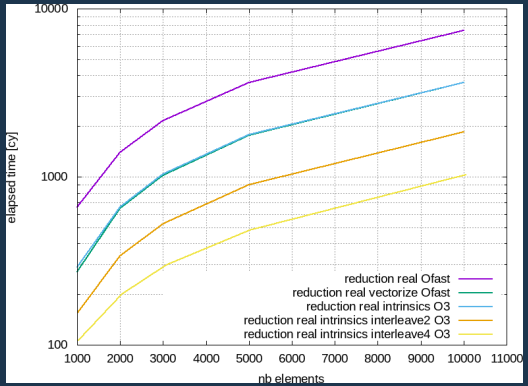


Elapsed Time per element (cy/el)

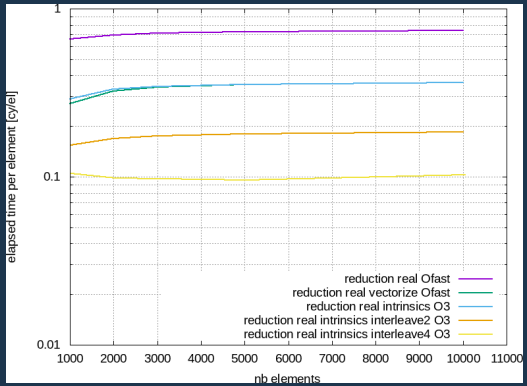


The reduction : intrinsics interleaved 4

Total Elapsed Time (cy)

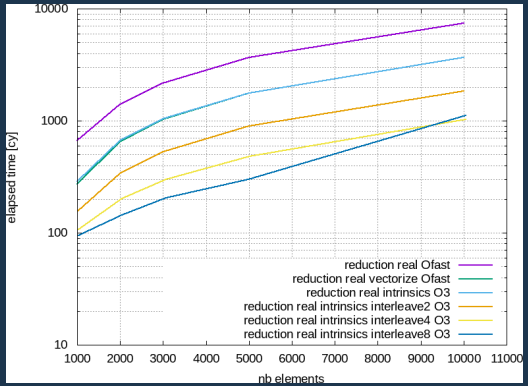


Elapsed Time per element (cy/el)

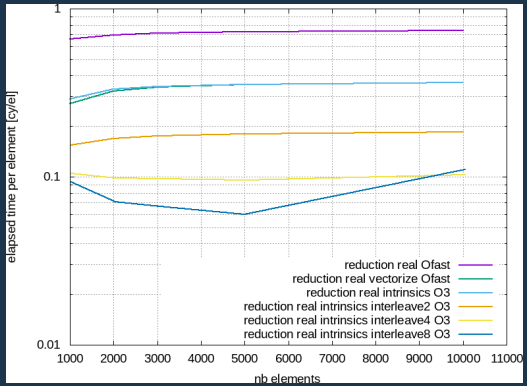


The reduction : intrinsics interleaved 8

Total Elapsed Time (cy)

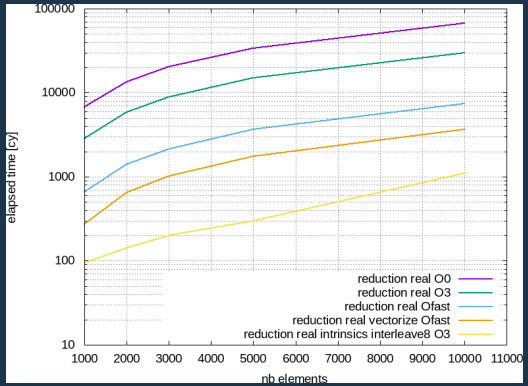


Elapsed Time per element (cy/el)

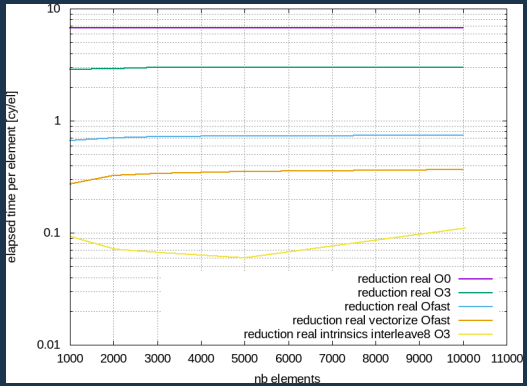


The reduction : summary

Total Elapsed Time (cy)



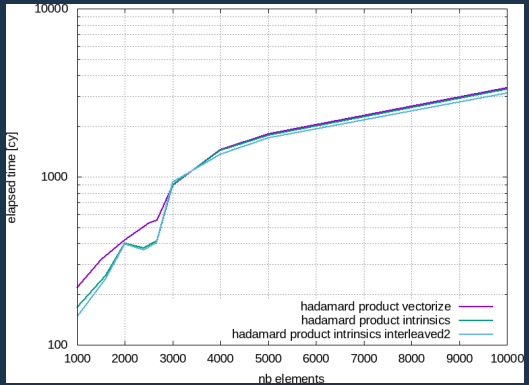
Elapsed Time per element (cy/el)



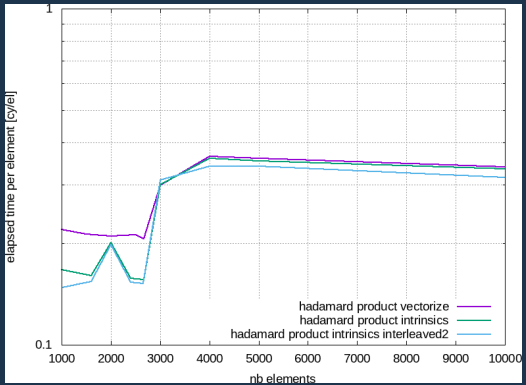
5000 elements, Intrinsics is **166** times faster than -**O0** and 7 times faster than -**Ofast** vectorized

What about the Hadamard product ?

Total Elapsed Time (cy)

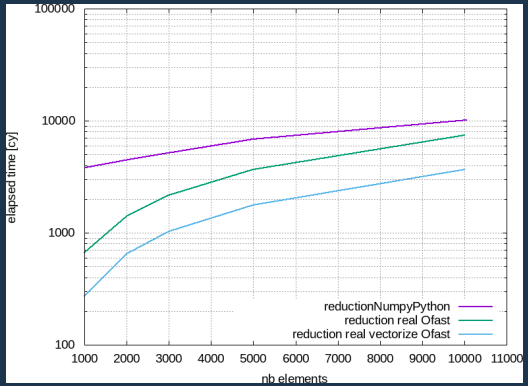


Elapsed Time per element (cy/el)

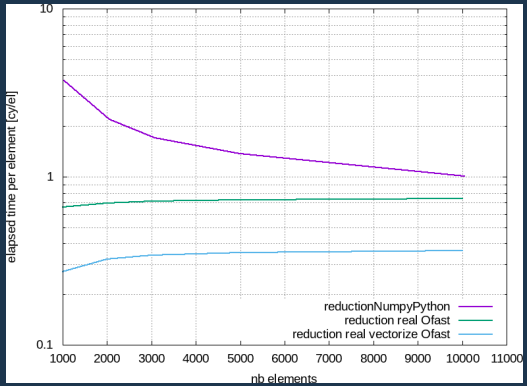


The reduction : Python

Total Elapsed Time (cy)



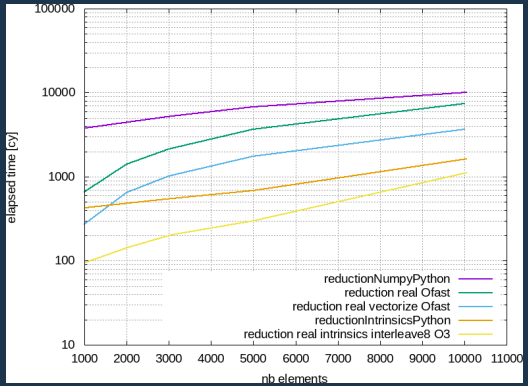
Elapsed Time per element (cy/el)



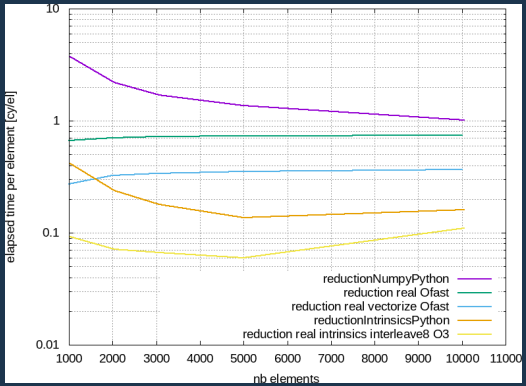
1000 elements, GCC vectorized version is **13** times faster than **numpy** sum

The reduction : Python Summary

Total Elapsed Time (cy)



Elapsed Time per element (cy/el)



1000 elements, our python reduction is **10** times faster than **numpy** sum