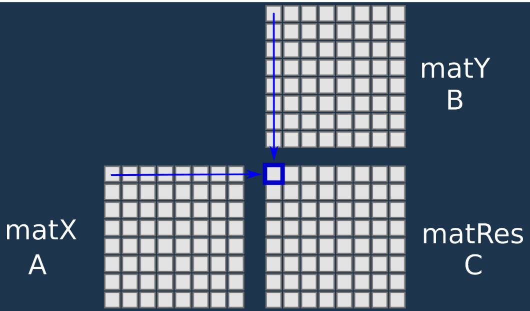
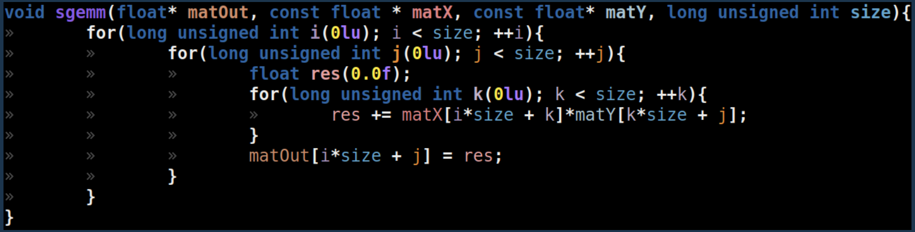
Optimisation : sgemm

Pierre Aubert

The Single precision General Matrix Multiply

$$C_{i,j} = \sum_{k=1}^{N} A_{i,k} \cdot B_{k,j}, \quad \forall i,j \in 1, N$$

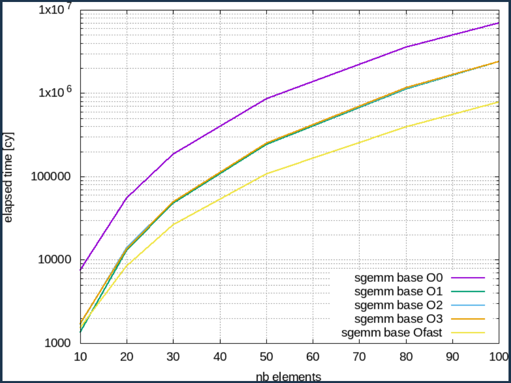




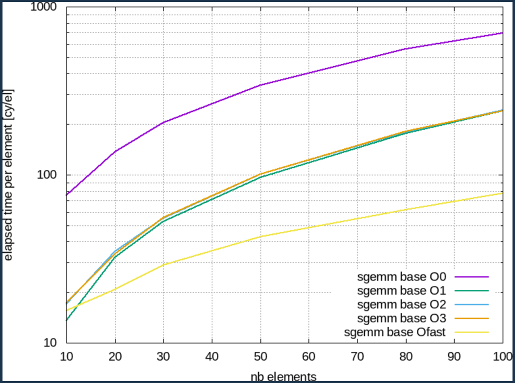
```
void sgemm(float* matOut, const float * matX, const float* matY, long unsigned int size){
»	for(long unsigned int i(0lu); i < size; ++i){
»	»	for(long unsigned int j(0lu); j < size; ++j){
»	»	»	float res(0.0f);
»	»	»	for(long unsigned int k(0lu); k < size; ++k){
»	»	»	»	res += matX[i*size + k]*matY[k*size + j];
»	»	»	}
»	»	»	matOut[i*size + j] = res;
»	»	}
»	}
}
```

SGEMM : performances

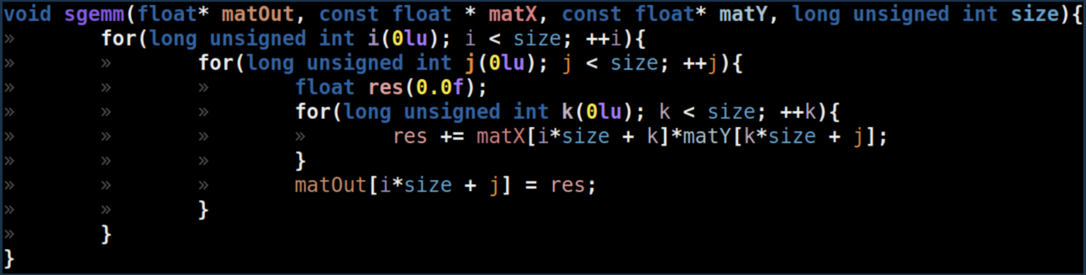
Total Elapsed Time (cy)



Elapsed Time per element (cy/el)

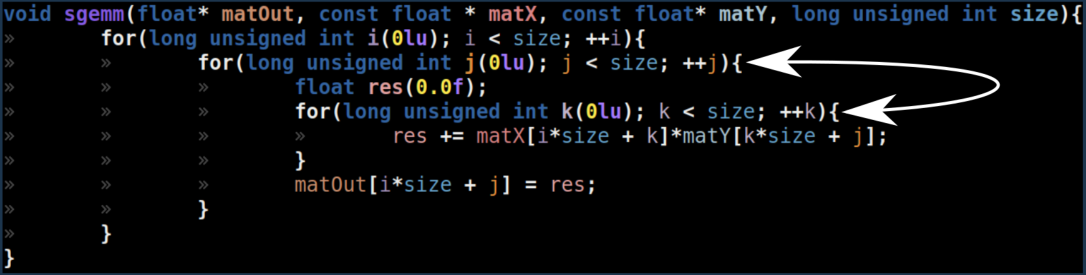


Let's swap loops over j and k



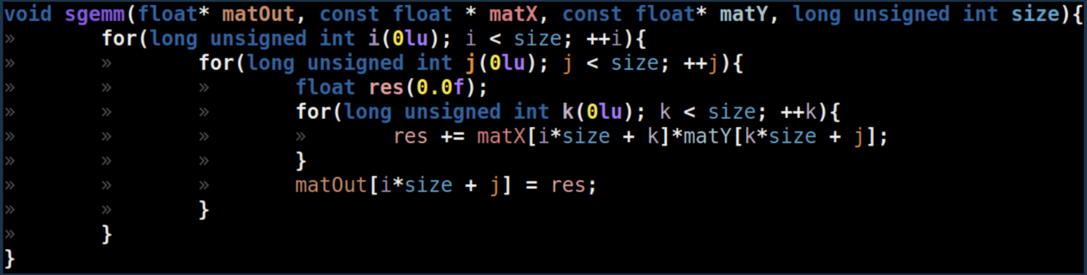
```
void sgemm(float* matOut, const float * matX, const float* matY, long unsigned int size){
	for(long unsigned int i(0lu); i < size; ++i){
		for(long unsigned int j(0lu); j < size; ++j){
			float res(0.0f);
			for(long unsigned int k(0lu); k < size; ++k){
				res += matX[i*size + k]*matY[k*size + j];
			}
			matOut[i*size + j] = res;
		}
	}
}
```

Let's swap loops over j and k

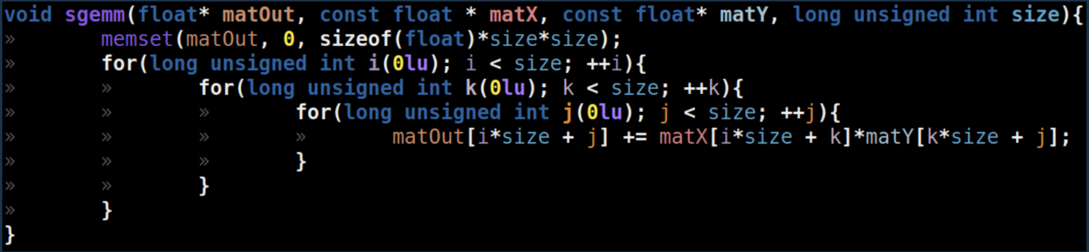


```
void sgemm(float* matOut, const float * matX, const float* matY, long unsigned int size){
	for(long unsigned int i(0lu); i < size; ++i){
		for(long unsigned int j(0lu); j < size; ++j){
			float res(0.0f);
			for(long unsigned int k(0lu); k < size; ++k){
				res += matX[i*size + k]*matY[k*size + j];
			}
			matOut[i*size + j] = res;
		}
	}
}
```

Let's swap loops over j and k

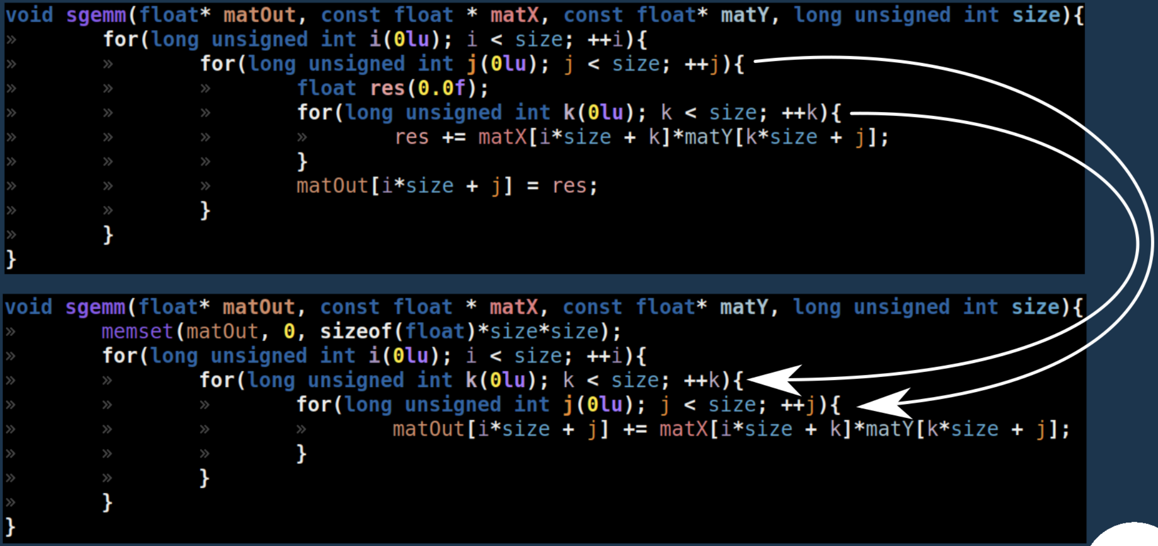


```
void sgemm(float* matOut, const float * matX, const float* matY, long unsigned int size){
»	for(long unsigned int i(0lu); i < size; ++i){
»	»	for(long unsigned int j(0lu); j < size; ++j){
»	»	»	float res(0.0f);
»	»	»	for(long unsigned int k(0lu); k < size; ++k){
»	»	»	»	res += matX[i*size + k]*matY[k*size + j];
»	»	»	}
»	»	»	matOut[i*size + j] = res;
»	»	}
»	}
}
```



```
void sgemm(float* matOut, const float * matX, const float* matY, long unsigned int size){
»	memset(matOut, 0, sizeof(float)*size*size);
»	for(long unsigned int i(0lu); i < size; ++i){
»	»	for(long unsigned int k(0lu); k < size; ++k){
»	»	»	for(long unsigned int j(0lu); j < size; ++j){
»	»	»	»	matOut[i*size + j] += matX[i*size + k]*matY[k*size + j];
»	»	»	}
»	»	}
»	}
}
```

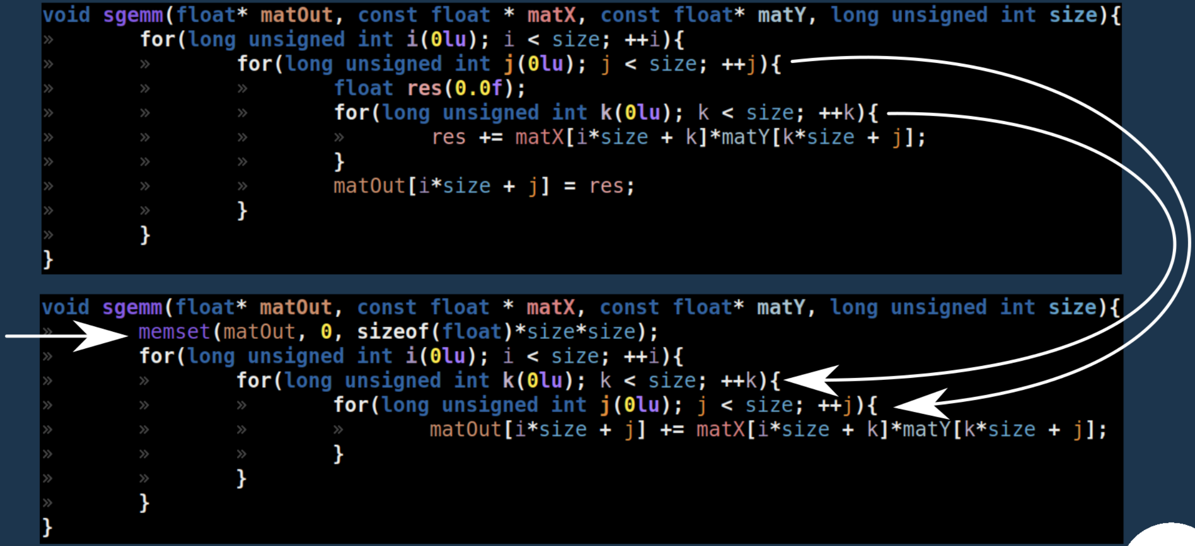
Let's swap loops over j and k



```
void sgemm(float* matOut, const float * matX, const float* matY, long unsigned int size){
	for(long unsigned int i(0lu); i < size; ++i){
		for(long unsigned int j(0lu); j < size; ++j){
			float res(0.0f);
			for(long unsigned int k(0lu); k < size; ++k){
				res += matX[i*size + k]*matY[k*size + j];
			}
			matOut[i*size + j] = res;
		}
	}
}
```

```
void sgemm(float* matOut, const float * matX, const float* matY, long unsigned int size){
	memset(matOut, 0, sizeof(float)*size*size);
	for(long unsigned int i(0lu); i < size; ++i){
		for(long unsigned int k(0lu); k < size; ++k){
			for(long unsigned int j(0lu); j < size; ++j){
				matOut[i*size + j] += matX[i*size + k]*matY[k*size + j];
			}
		}
	}
}
```

Let's swap loops over j and k

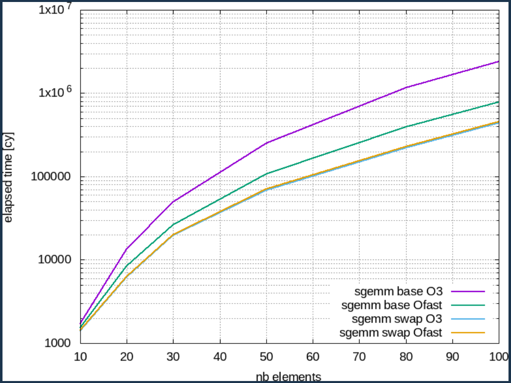


```
void sgemm(float* matOut, const float * matX, const float* matY, long unsigned int size){
»       for(long unsigned int i(0lu); i < size; ++i){
»       »       for(long unsigned int j(0lu); j < size; ++j){
»       »       »       float res(0.0f);
»       »       »       for(long unsigned int k(0lu); k < size; ++k){
»       »       »       »       res += matX[i*size + k]*matY[k*size + j];
»       »       »       }
»       »       »       matOut[i*size + j] = res;
»       »       }
»       }
}
```

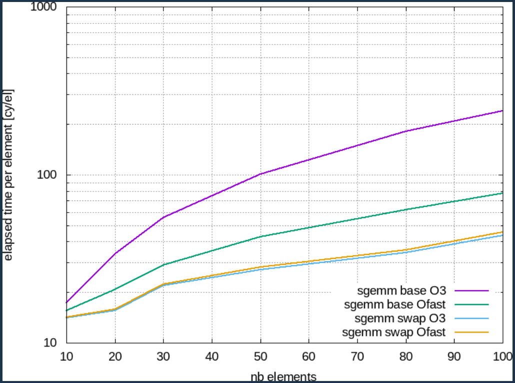
```
void sgemm(float* matOut, const float * matX, const float* matY, long unsigned int size){
»       memset(matOut, 0, sizeof(float)*size*size);
»       for(long unsigned int i(0lu); i < size; ++i){
»       »       for(long unsigned int k(0lu); k < size; ++k){
»       »       »       for(long unsigned int j(0lu); j < size; ++j){
»       »       »       »       matOut[i*size + j] += matX[i*size + k]*matY[k*size + j];
»       »       »       }
»       »       }
»       }
}
```

SGEMM : Swap loop performances

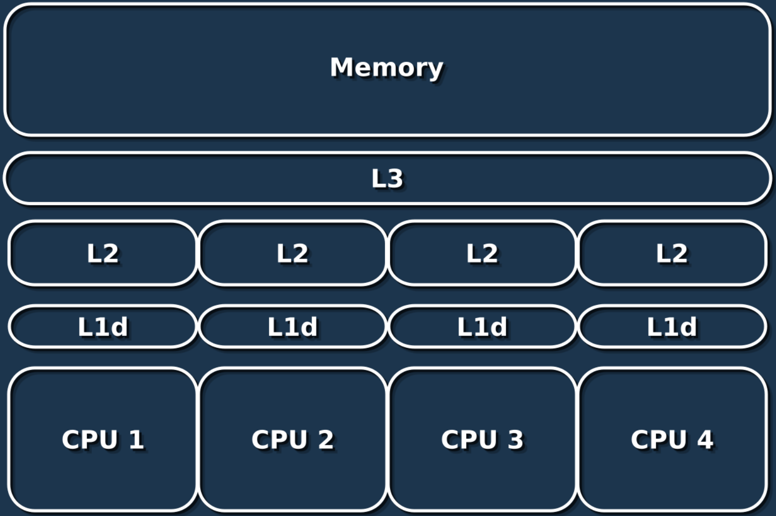
Total Elapsed Time (cy)



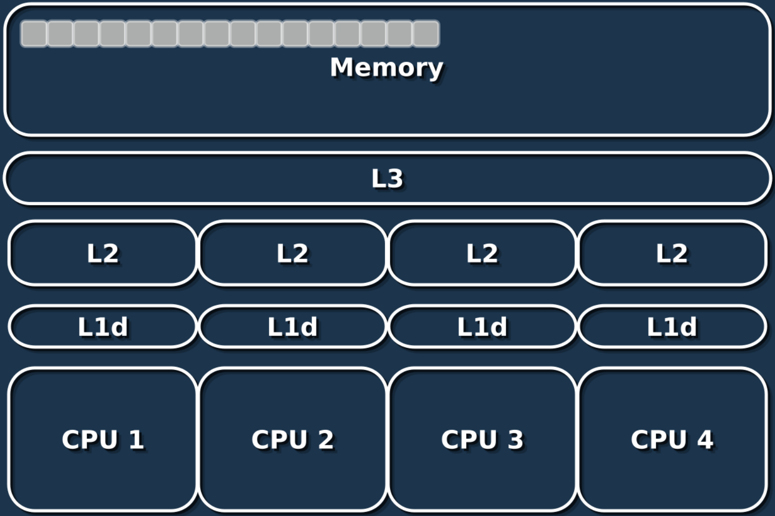
Elapsed Time per element (cy/el)



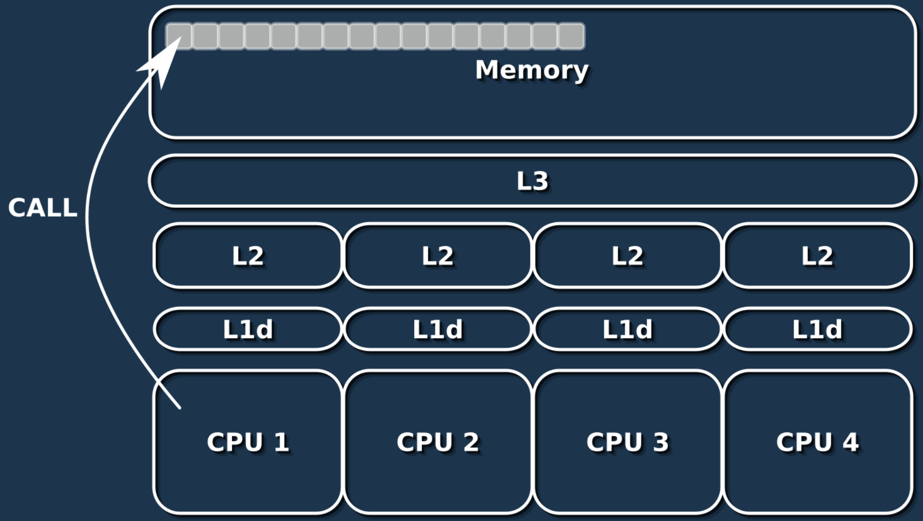
Data Fetching and pre-fetching



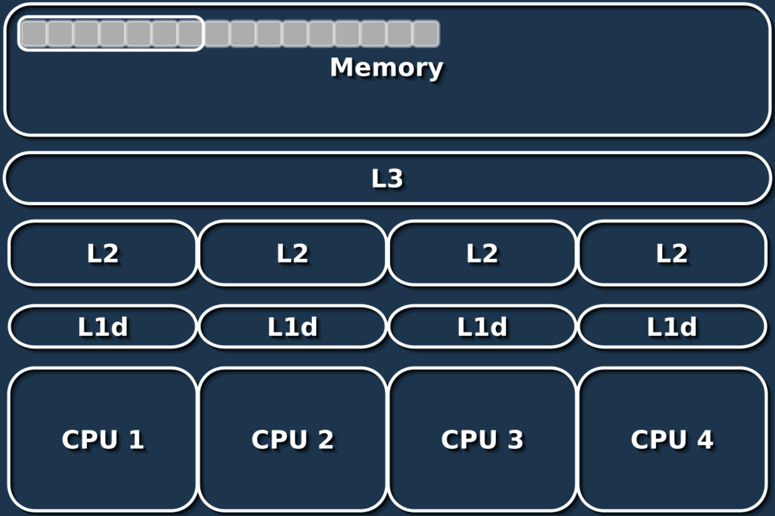
Data Fetching and pre-fetching



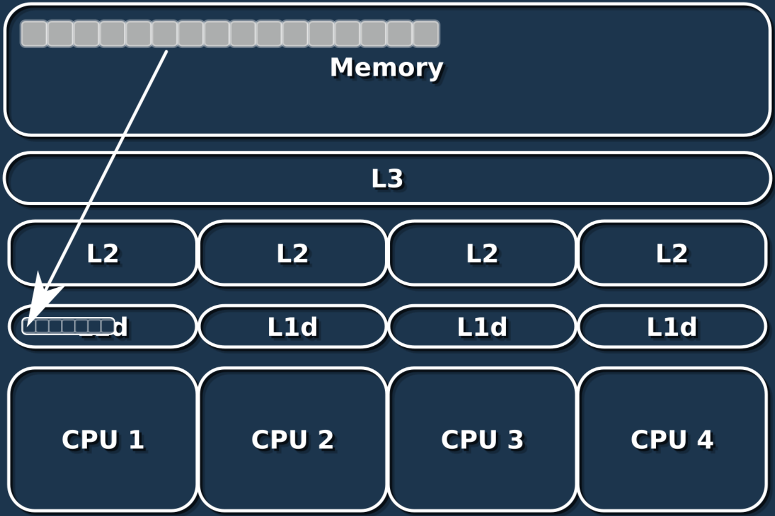
Data Fetching and pre-fetching



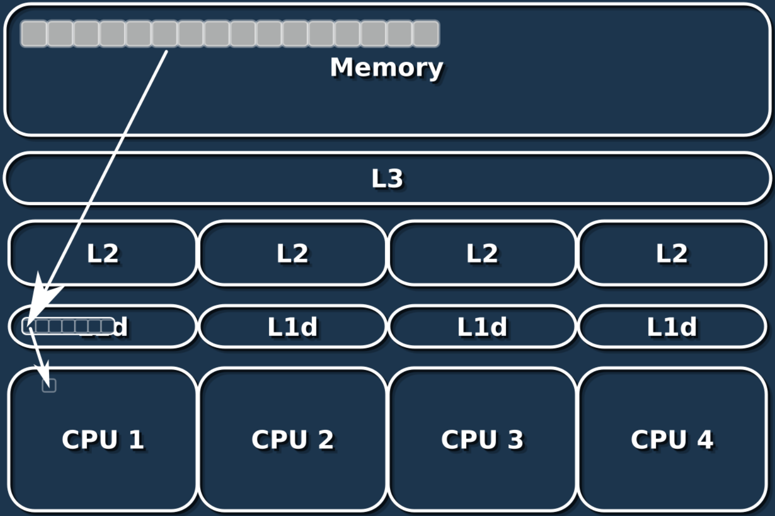
Data Fetching and pre-fetching



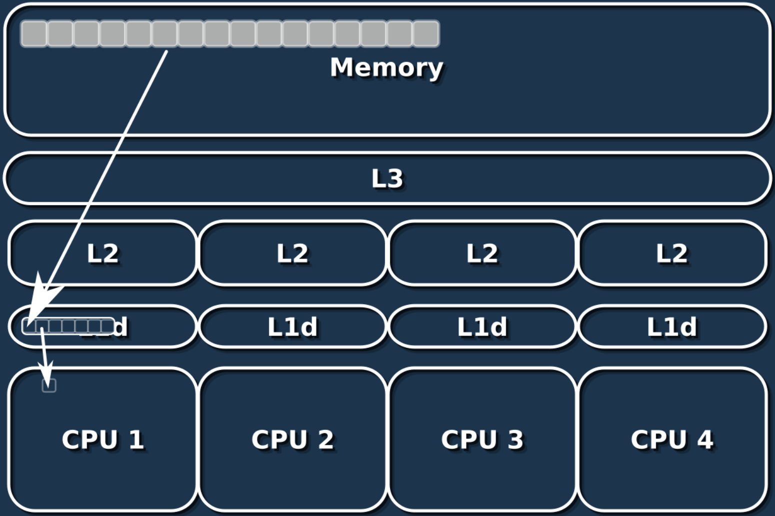
Data Fetching and pre-fetching



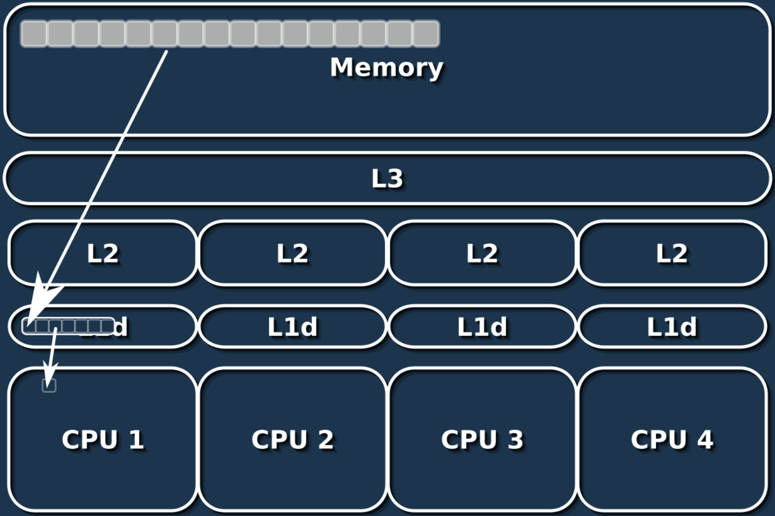
Data Fetching and pre-fetching



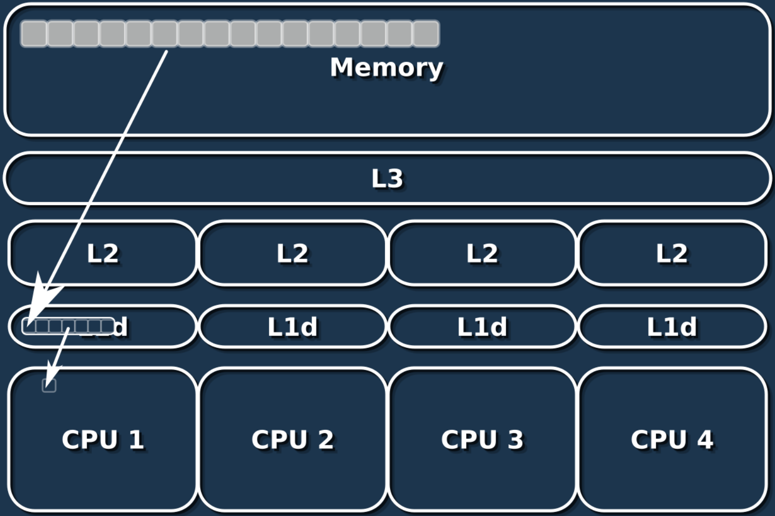
Data Fetching and pre-fetching



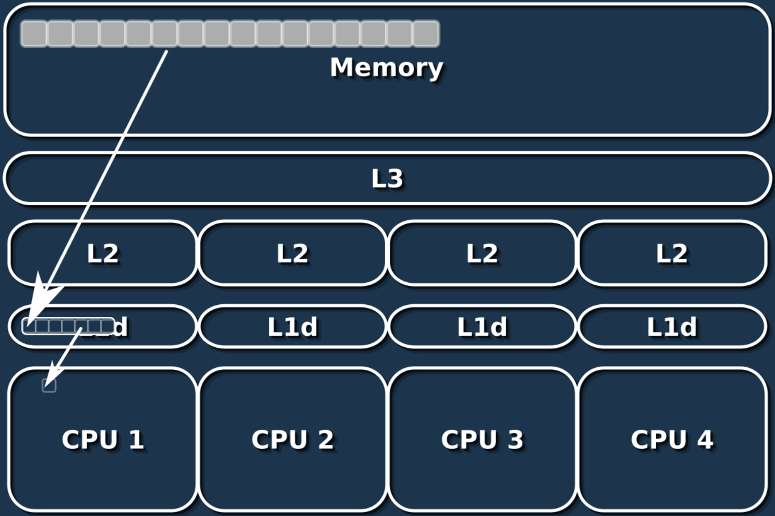
Data Fetching and pre-fetching



Data Fetching and pre-fetching

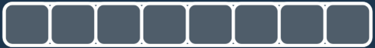


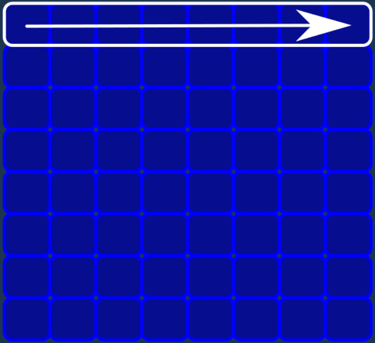
Data Fetching and pre-fetching



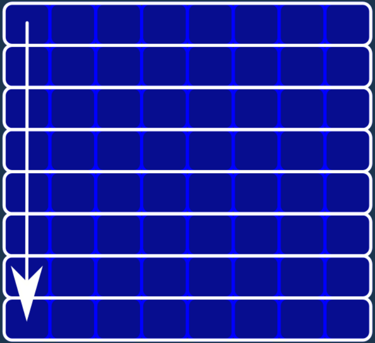
Data locality

Data Fetch Size 8 elements





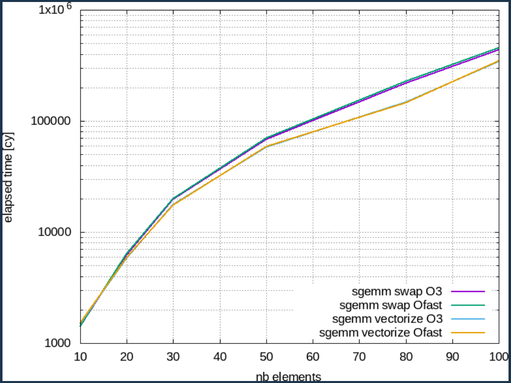
Read 8 elements with one fetch



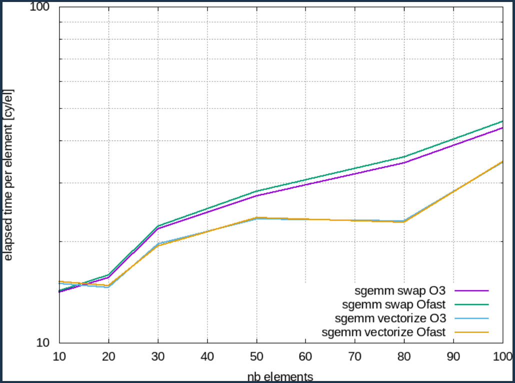
Read 8 elements with 8 fetchs
=> fecth 64 elements instead of 8

SGEMM : vectorize performances

Total Elapsed Time (cy)

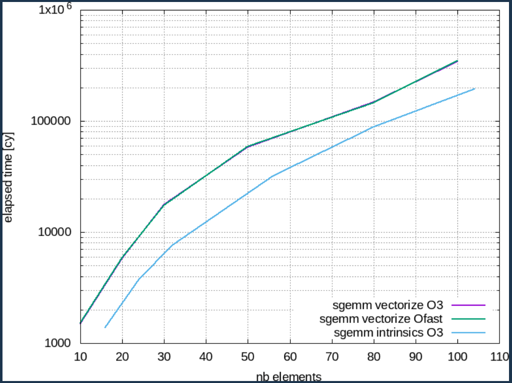


Elapsed Time per element (cy/el)

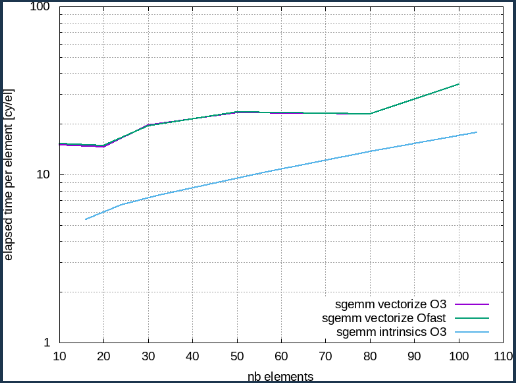


SGEMM : intrinsics performances

Total Elapsed Time (cy)



Elapsed Time per element (cy/el)



What if matrices don't have proper size ?

For our intrinsics version :

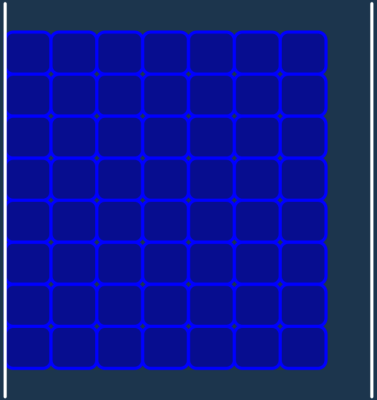
- The number of columns has to be a multiple of 8

What if matrices don't have proper size ?

For our intrinsics version :

The number of columns has to be a multiple of 8

If is it not the case :



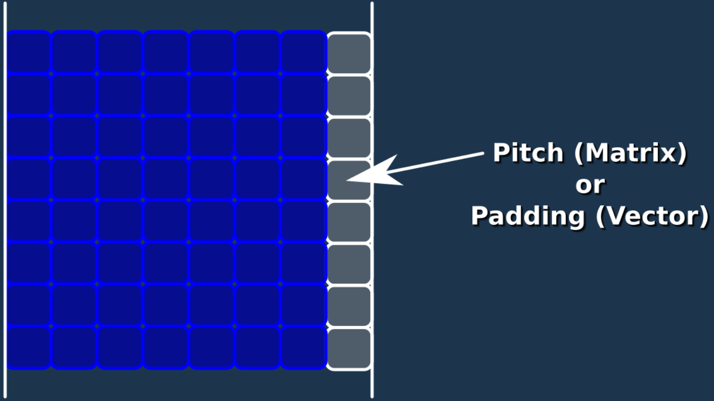
Vectorial register size

What if matrices don't have proper size ?

For our intrinsics version :

The number of columns has to be a multiple of 8

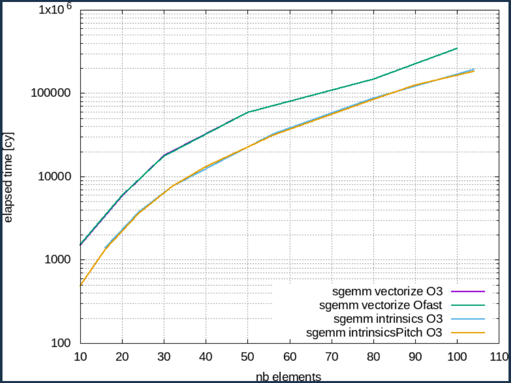
If is it not the case :



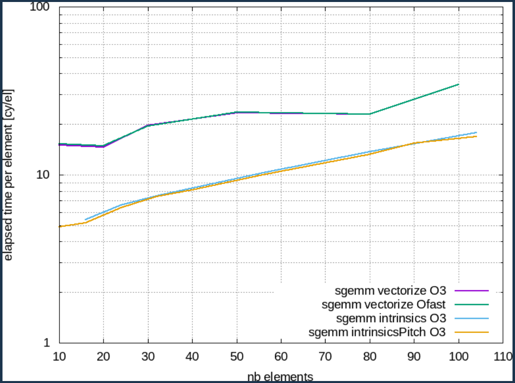
Vectorial register size

SGEMM : intrinsics pitch performances

Total Elapsed Time (cy)

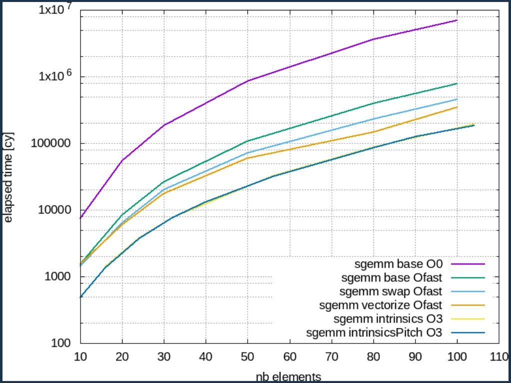


Elapsed Time per element (cy/el)

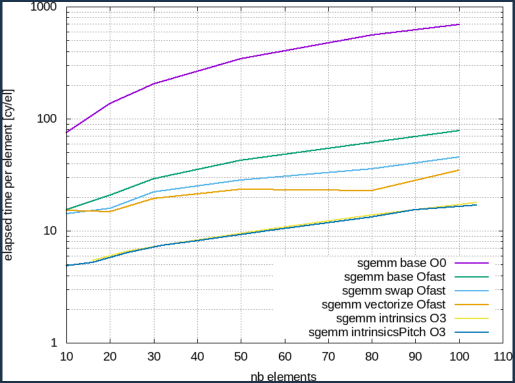


SGEMM : summary

Total Elapsed Time (cy)

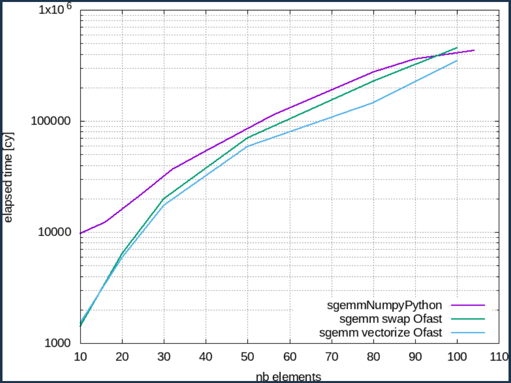


Elapsed Time per element (cy/el)

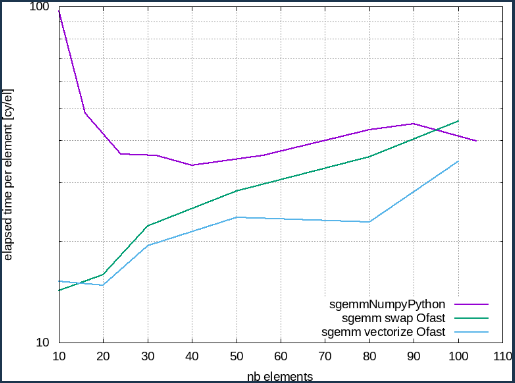


SGEMM : Python

Total Elapsed Time (cy)



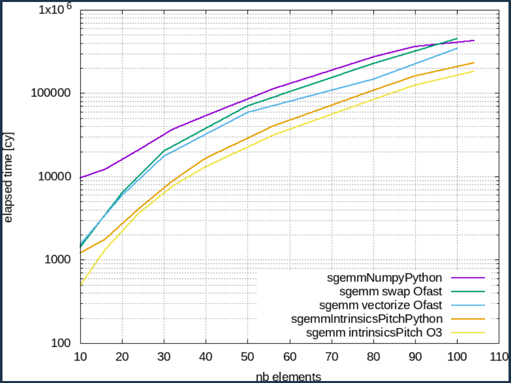
Elapsed Time per element (cy/el)

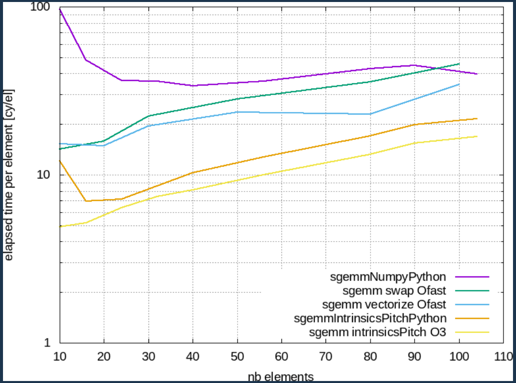


SGEMM : Python Summary

Total Elapsed Time (cy)

Elapsed Time per element (cy/el)


1x10^6
elapsed time [cy]
nb elements
sgemmNumpyPython
sgemm swap Ofast
sgemm vectorize Ofast
sgemmIntrinsicsPitchPython
sgemm intrinsicsPitch O3


elapsed time per element [cy/el]
nb elements
sgemmNumpyPython
sgemm swap Ofast
sgemm vectorize Ofast
sgemmIntrinsicsPitchPython
sgemm intrinsicsPitch O3