Performance Portability Evaluation: Non-negative Matrix Factorization as a case study

Youssef Faqir-Rhazoui, Carlos García and Francisco Tirado

Distributed and Heterogeneous Programming in C++, August 2022

Agenda

- Introduction
- What is the non-negative matrix factorization algorithm?
- Proposed implementations
- Experimental conditions
- Experimental results
- Conclusion

Introduction

- The NMF algorithm was first **proposed by Paataro and Tapper in 1994**.[1]
- The NMF is used for **dimensionality reduction** in fields such as biology or image processing among others.
- Libraries such as **scikit-learn** include it, but its use is constrained to **only CPUs**. **Pytorch** has GPU support but is restricted to **only Nvidia's**.
- This work **develops a multi-device** (CPU and GPU) **version of the NMF for SYCL, OpenMP and CUDA** (only Nvidia GPUs) and compares their performance.

1. Paatero P, Tapper U. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. Environmetrics 1994

Non-negative Matrix Factorization

- The NMF decomposition can be seen as:

$$V \approx W * H$$

- Where: $V \in \mathbb{R}_{+}^{m \times n}$, is the original matrix with ‘m’ variables and ‘n’ objects.

  $W \in \mathbb{R}_{+}^{m \times k}$, is the reduced ‘k’ vector or factor.

  $H \in \mathbb{R}_{+}^{k \times n}$, contains the coefficients of linear combinations of the basis vectors.

How to decompose the V matrix?

1. Randomly initialize W and H matrices.
2. Repeatedly modifies W and H until their product approximates to V.
3. Such modifications are derived from minimizing a cost function (Euclidean distance).

$$W_{i\alpha} \leftarrow W_{i\alpha} \frac{\sum_{\mu} H_{\alpha\mu} V_{i\mu} / (WH_{i\mu})}{\sum_{\nu} H_{\alpha\nu}}$$

$$H_{\alpha\mu} \leftarrow H_{\alpha\mu} \frac{\sum_{i} W_{i\alpha} V_{i\mu} / (WH_{i\mu})}{\sum_{k} W_{k\alpha}}$$

Pseudocode

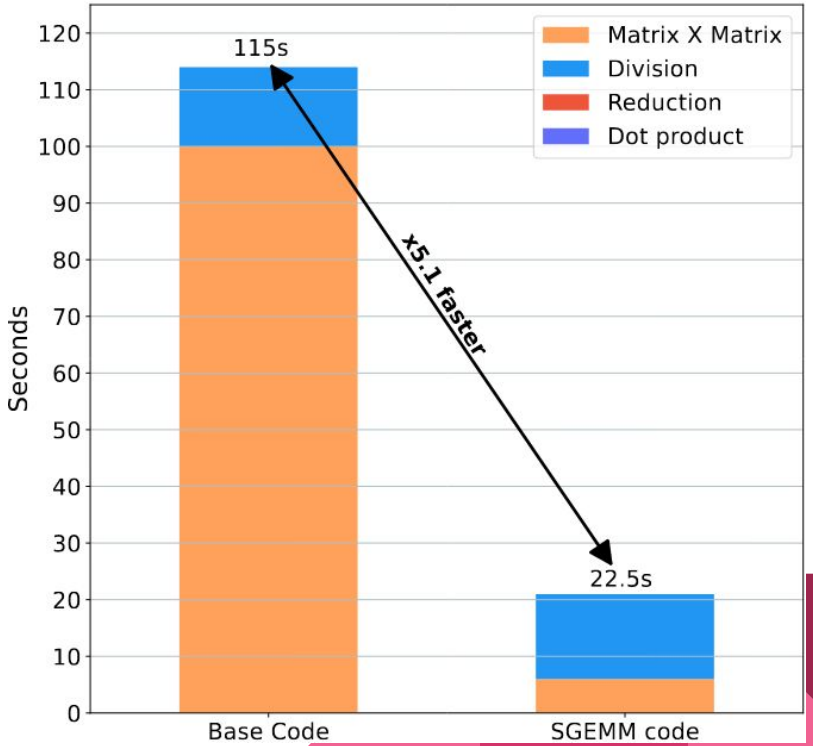
- Matrix multiplication (lines 5 and 11)
- Division (line 6)
- Multi-variable reduction (line 9)
- Dot product (line 12)

Algorithm 1 NMF($V^{n\times m}$, $W^{n\times k}$, $H^{k\times m}$, $niters$)

1: **for** $iter \leq niters$ **do**
2:
3: ▷ Get H as $H = H. * (W' * (V./(W * H)))./x_1$
4:
5: $wh = W * H$
6: $wh = V./wh$
7:
8: ▷ Reduce to one column (x_1)
9: $x_1 = repmat(sum(W, 1)', 1, m)$
10:
11: $Haux = W * wh$
12: $H = (H. * Haux)./x_1$
13:
14: ▷ Get W as $W = W. * ((V./(W * H)) * H')./x_2$
15: ...
16: **end for**

BLAS Baseline Implementation

- C++ implementation to compare the base performance on CPU.
- Only the **matrix multiplication** was **optimized with the MKL** library.
- Intel **oneAPI** were used **for the compiler** (icpx) and for the library **oneMKL**.



SYCL Implementation

- Since **oneMKL is used for the matrix multiplication**, the **other kernels** were implemented with the **‘nd-range’ scheme**.
- While **oneAPI** suit is used **for CPU and Intel GPUs**, its lack of compatibility with Nvidia GPUs makes it impossible to keep with the same set-up.
- To solve that, we used the **open version of the oneAPI’s compiler [2] and oneMKL library.**[3]

2. https://github.com/intel/llvm
3. https://github.com/oneapi-src/oneMKL

OpenMP Implementation

- The GPU implementation uses the pragmas **‘target’, ‘teams_distribute’, ‘num_teams’ and ‘thread_limit’ to spread data over GPU threads.**
- Three OpenMP implementations were developed. Since OpenMP and OpenMP offload differs from the notation, the **CPU and GPU pragmas are incompatible**.
- The other version comes from the fact that oneAPI is only compatible with Intel GPUs. So, to run OpenMP on **Nvidia GPUs we need the CUDA HPC SDK.**

Work Environment

Specs / Devices	AMD EPYC 7742	Intel Core i9-10920X	Intel Iris Xe MAX DG1	Nvidia RTX 3090
Frecuency	2.25 GHz(Base) 3.4 GHz(Boost)	3.5 GHz(Base) 4.6 GHz(Boost)	0.3 GHz(Base) 1.6 GHz(Boost)	1.4 GHz (Base) 1.7 GHz(Boost)
Cores	2 x 64	2 x 12	96 compute units	10496 cuda cores
Performance (FP32)	2.5 TFLOPS	672 GFLOPS	2.5 TFLOPS	35.6 TFLOPS
Memory BW	204.8 GB/s	94 GB/s	62.3 GB/s	963.2 GB/s
Driver	-	-	21.49.21786	515.43.04

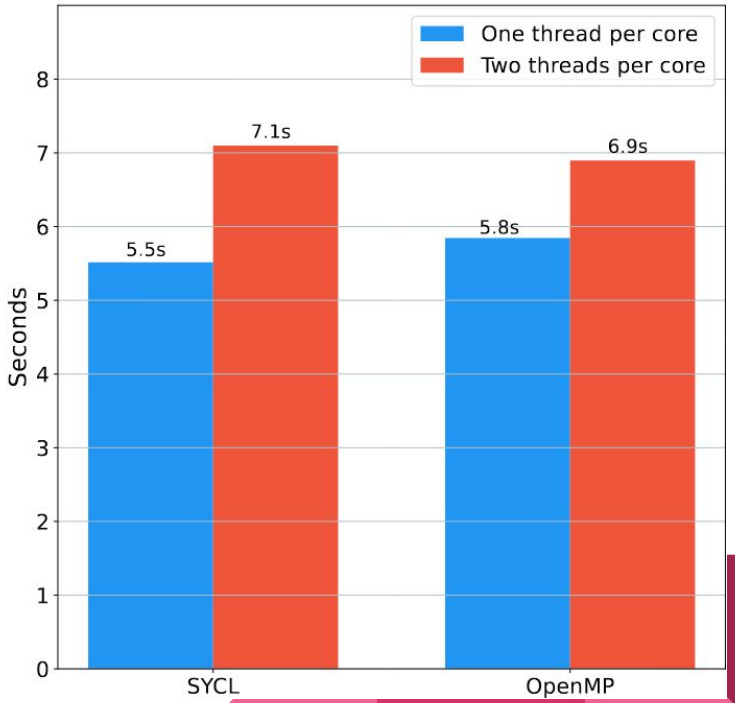
Data Description

- **Lung (16,063 × 280)**: Contains **16,063 genes** by Affymetrix Genechips of **primary tumors tissues and** poorly differentiated **adenocarcinomas**.

- **ExpO (54,675 × 1,973)**: A set of **1,973 tumor samples** obtained by the expO project.

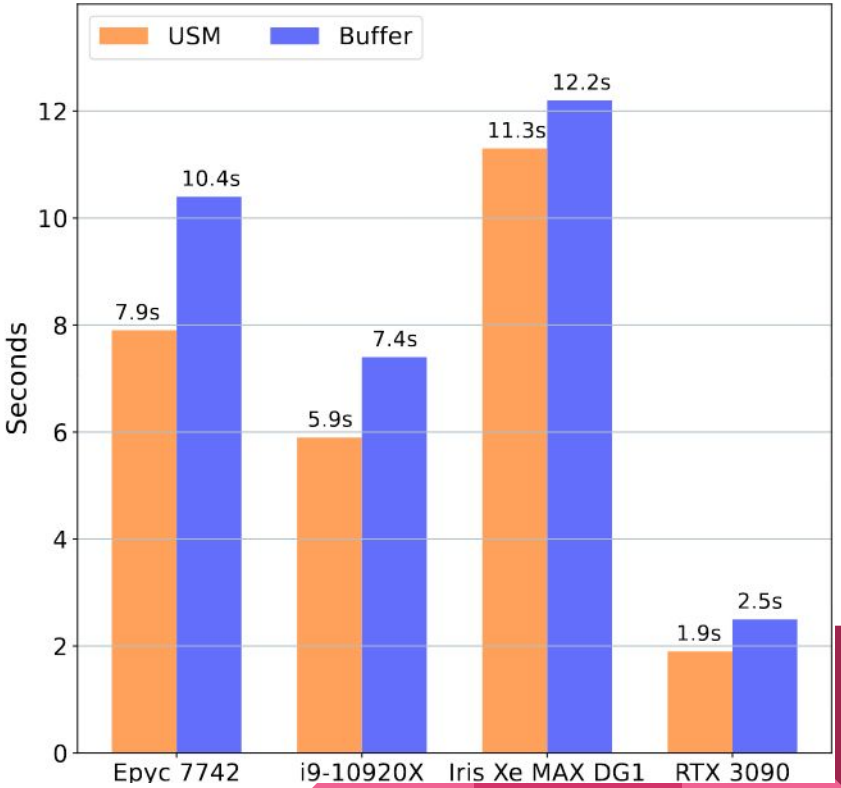
CPU Multithreading

- Tested on **i9-10920X** with the **Lung data set** (16063 × 280).
- Using one thread by physical core **increases the performance by 22%** in the case of **SYCL**, while **it is 16% in OpenMP**.
- **The issue** is that **threads compete by common CPU resources.**



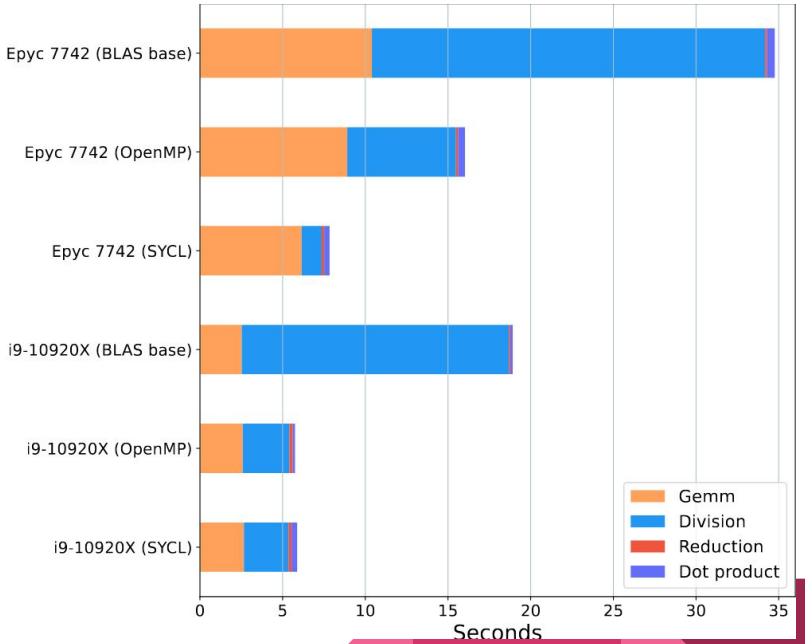
Buffers vs USM in SYCL

- The Unified Shared Memory (**USM**) is a more **lightweight model**, while the **buffer** model is a **memory abstraction**.
- Important **differences from 8% up to 24%** by choosing between USM or buffer models.



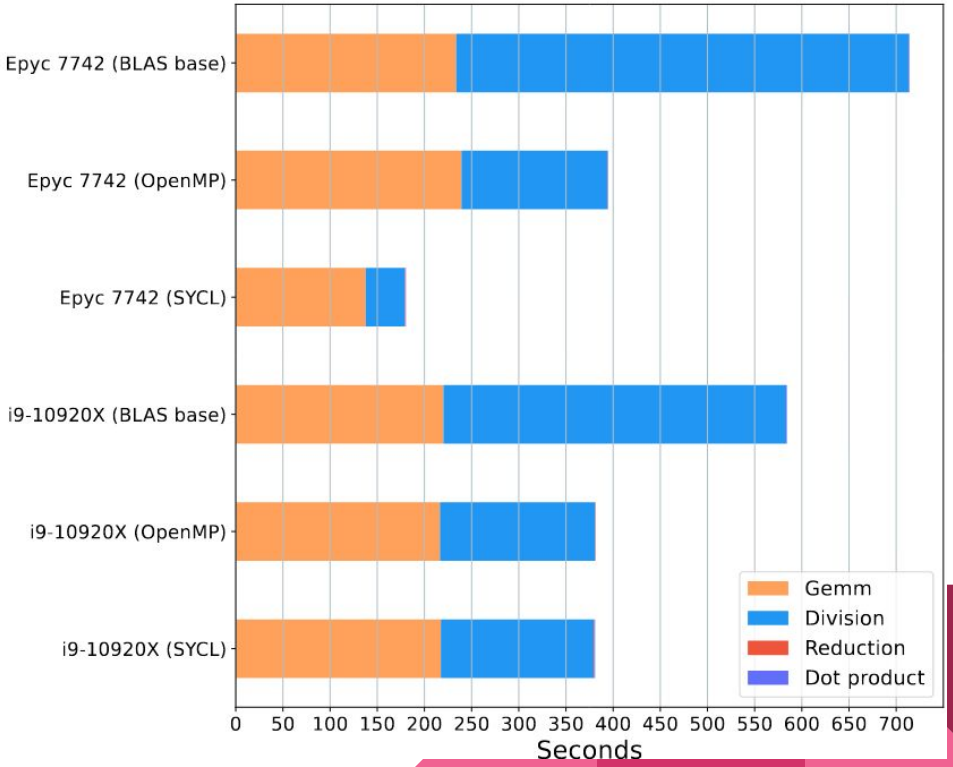
CPU results (Lung data set)

- The AMD EPYC achieved a **speedup of ≈2.2× in OpenMP and ≈4.4× in SYCL** with respect to the BLAS base version.
- **AMD processors are not optimized for oneAPI**, since Intel does not grant full compatibility with non-Intel CPUs.
- The **i9-10920X** shows a **speedup of ≈3.1x in both SYCL and OpenMP.**



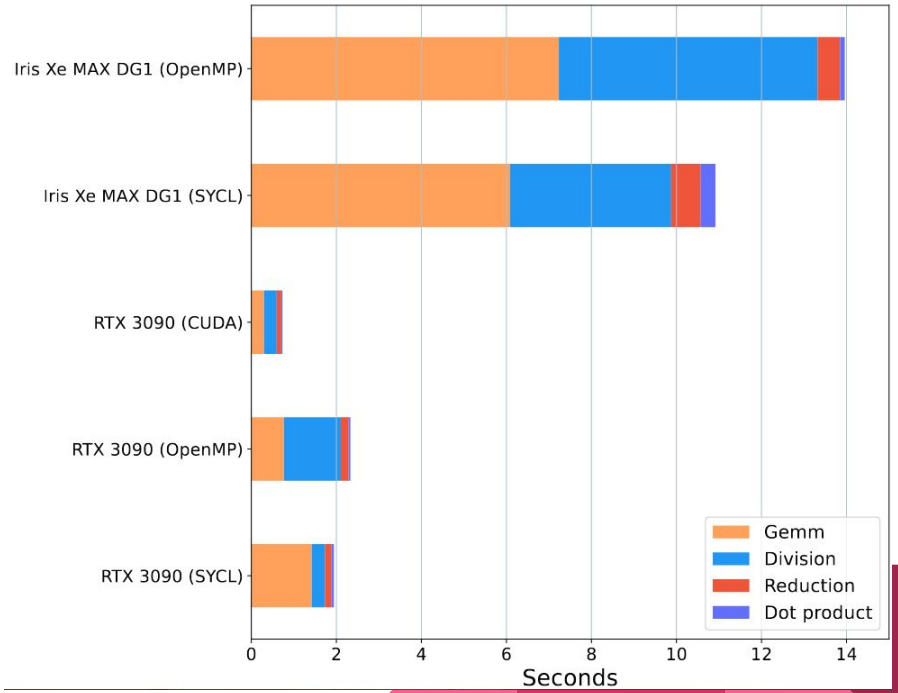
CPU results (ExpO data set)

- For the **EPYC, speedups of ≈1.8× in OpenMP and ≈4× in SYCL**, and newly, the same differences were found between OpenMP and SYCL.
- The **i9-10920X** gets a **speedup of ≈1.5×, either in SYCL or OpenMP.**



GPU results (Lung data set)

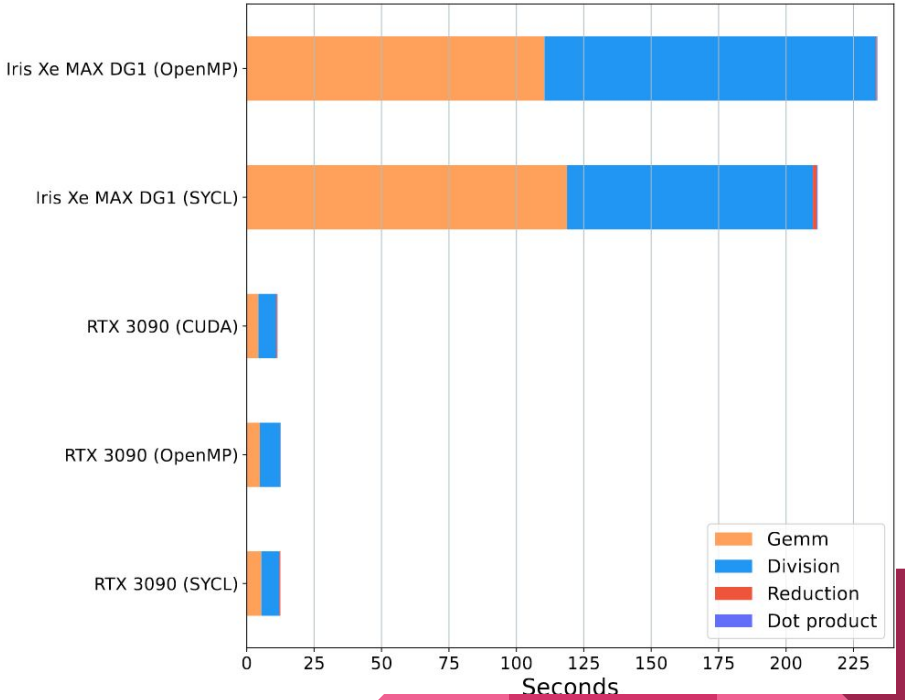
- The **Intel DG1** has a speedup of **≈1.26× on SYCL over OpenMP.**
- Concerning the **RTX 3090**, we get **0.9s for CUDA**, **2.4s for OpenMP** and **1.98s for SYCL**.
- The **GEMM kernel takes longer** to run on SYCL and OpenMP due to the **overhead of keeping the cuBLAS context.**[4]
- Both GPUs using OpenMP increases the time consumption of the division.



4. https://github.com/oneapi-src/oneMKL/issues/106

GPU results (ExpO data set)

- The **Intel DG1** has a speedup of **≈1.1× on SYCL over OpenMP.**
- In the **RTX 3090**, we get **12.2s in CUDA, 13.2s in OpenMP and 13s in SYCL.**
- **By moving to a larger data set** in the RTX 3090, **the previously cuBLAS issue disappear,** reducing the difference among versions to less than 7%.



Conclusion

- We evaluate the performance portability of **OpenMP, SYCL and CUDA** on the NMF algorithm **on different devices** (CPU and GPU) from different vendors.
- The experimental results reveal that while **CUDA offers the best performance**, its portability is **reduced just to NVIDIA GPUs.**
- **OpenMP requires some code customization** depending on the device and even the compiler. The **lack of expression of OpenMP** to exploit some GPU advantages **makes it a non-optimal implementation.**
- **SYCL code was written once** and successfully e**xecuted on different target devices** without changes. Even though **the performance is not the best** in all the architectures, its code portability **greatly reduces developing times.**

Thanks!

Supported by the EU (FEDER), the Spanish MINECO and CM under grants S2018/TCS-4423, RTI2018-093684-B-I00 and PID2021-126576NB-I00 funded by MCIN/AEI/10.13039/501100011033 and by “ERDF A way of making Europe"

https://github.com/artecs-group/nmf-dpcpp