codeplay®
Enabling AI to be Open, Safe & Accessible to All

Towards Performance Portability of Highly Parametrizable TRSM Algorithm Using SYCL

Thales Sabino

Mehdi Goli

IWOCL-SYCL 2021

Agenda

- Introduction
- The TRSM Problem
- GEMM-Based TRSM
- Performance Evaluation
- Conclusion and Future Work

Introduction

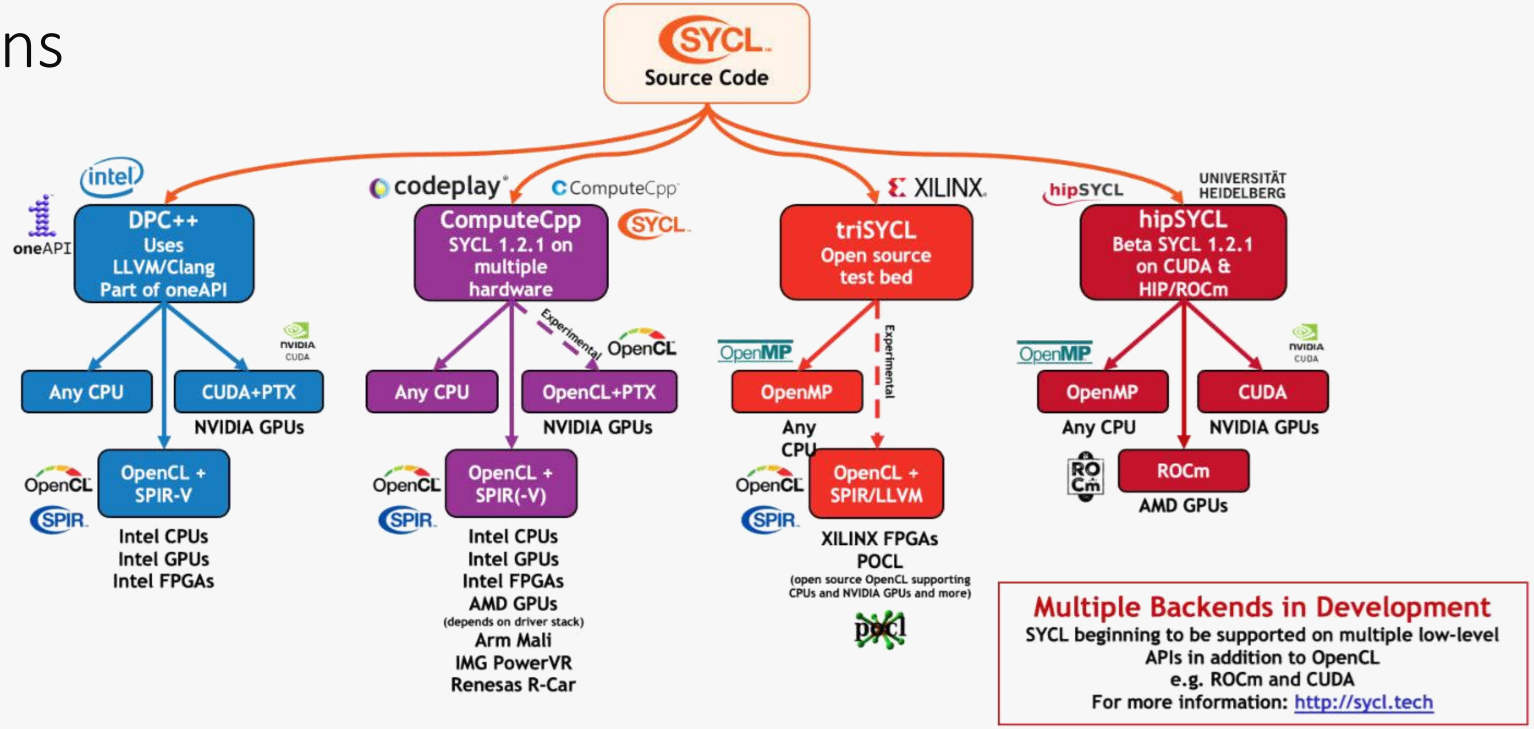
- **TR**iangular **S**olve with **M**ultiple Right-Hand Sides
- TRSM is an important operation used to solve linear systems efficiently
- Performance portable GEMM-based TRSM solver in SYCL-BLAS
- TRSM is simple to solve sequentially
- Can be solved in parallel devices using a GEMM-based approach

Motivations

- Accelerated BLAS routines
- Architectures
  - CPU/GPU/FPGA
  - Embedded Accelerators
- Performance portability
  - Library approach
    - BLAS/ MKL/cuBLAS
  - Parallel pattern abstractions
    - Raja/Kokkos/Eigen
- Challenges
  - Provide a cross-platform performance portable programming model for future development

SYCL

- C++ based open standard API introduced by Khronos
- Provides single-source programming model for accelerators
- Provides an implicit execution graph by tracking kernel dependencies
- Multiple implementations
  - ComputeCpp
  - Intel DPC++
  - hipSYCL
  - triSYCL
- Programming model
  - Kernel Scope
  - Command group scope
  - Application Scope



The TRSM Problem

- Triangular solve with multiple right-hand sides
- Solve for $\boldsymbol{X}$ in one of the following matrix equations:

$$op(\boldsymbol{A})_{(m,m)}\boldsymbol{X}_{(m,n)} = \alpha\boldsymbol{B}_{(m,n)}$$
$$\boldsymbol{X}_{(m,n)}op(\boldsymbol{A})_{(n,n)} = \alpha\boldsymbol{B}_{(m,n)}$$

- $op(\boldsymbol{A}) = \boldsymbol{A}$ or $op(\boldsymbol{A}) = \boldsymbol{A}^T$
- $\boldsymbol{A}$ can be upper or lower triangular
- $\boldsymbol{A}$ can have a unit or non-unit diagonal
- $\boldsymbol{A}$ can be on the left or right side of $\boldsymbol{X}$
- $\boldsymbol{\alpha}$ is a scalar

Solving a TRSM Problem Sequentially

$$\begin{array}{ccc} \boldsymbol{A} & \boldsymbol{X} & \boldsymbol{B} \\ \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ 0 & a_{11} & a_{12} & a_{13} \\ 0 & 0 & a_{22} & a_{23} \\ 0 & 0 & 0 & a_{33} \end{bmatrix} & \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = & \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \end{array}$$

$$x_3 = \frac{b_3}{a_{33}}$$

Retro-substitution

$$x_2 = \frac{b_2 - a_{23}x_3}{a_{22}}$$

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}}$$

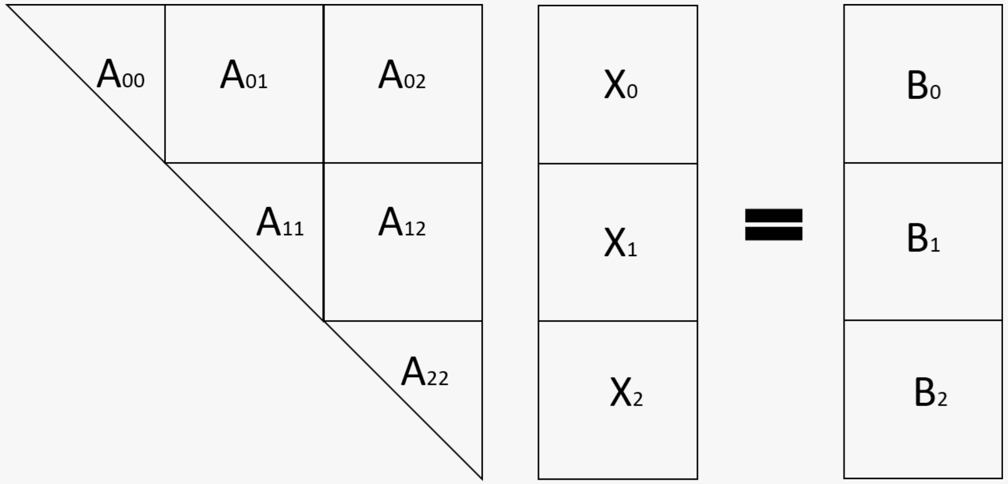
$$x_0 = \frac{b_0 - a_{01}x_1 - a_{02}x_2 - a_{03}x_3}{a_{00}}$$

Solving a TRSM Problem using GEMM

$$\boldsymbol{AX} = \alpha\boldsymbol{B}$$
$$\boldsymbol{X} = \alpha\boldsymbol{A}^{-\mathbf{1}}\boldsymbol{B}$$

- Two operations: matrix inversion and GEMM
- Matrix inversion has complexity in the order of solving a linear system
- Basically solving the problem twice!

Solving a TRSM Problem using GEMM



$$\boldsymbol{A}_{00}\boldsymbol{X}_0 + \boldsymbol{A}_{01}\boldsymbol{X}_1 + \boldsymbol{A}_{02}\boldsymbol{X}_2 = \boldsymbol{\alpha}\boldsymbol{B}_0$$
$$\boldsymbol{A}_{11}\boldsymbol{X}_1 + \boldsymbol{A}_{12}\boldsymbol{X}_2 = \boldsymbol{\alpha}\boldsymbol{B}_1$$
$$\boldsymbol{A}_{22}\boldsymbol{X}_2 = \boldsymbol{\alpha}\boldsymbol{B}_2$$

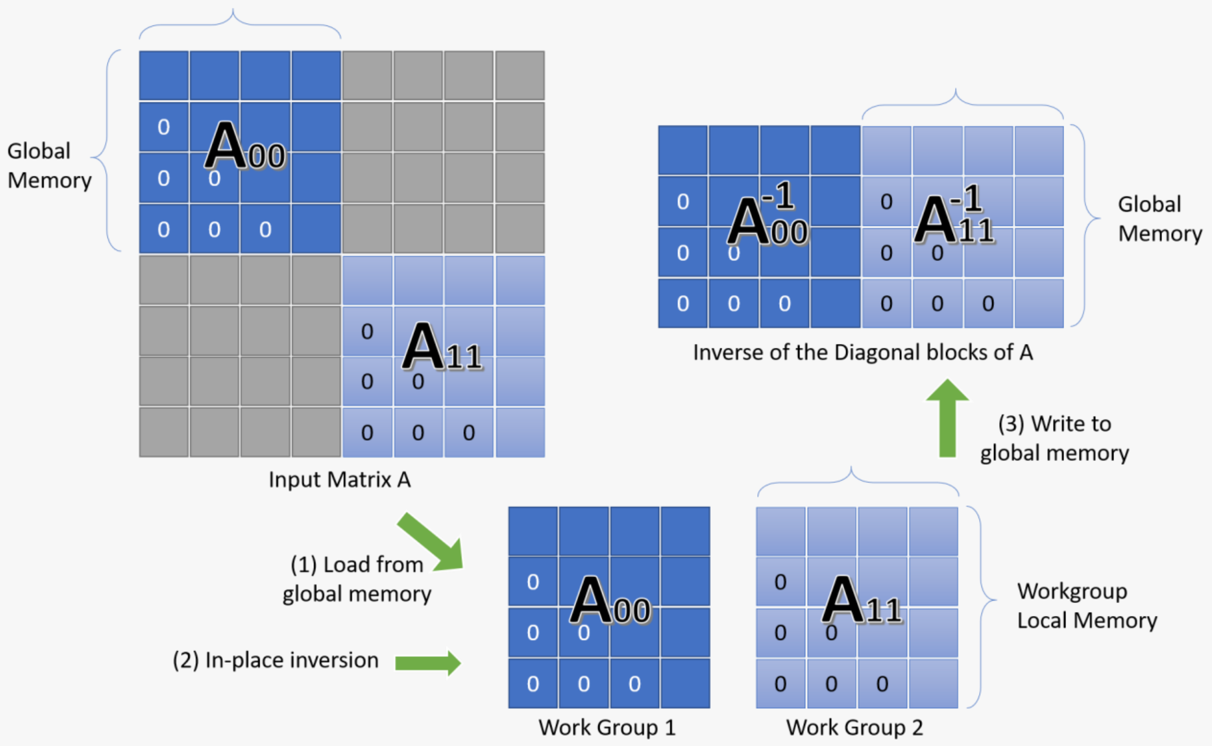
$$\Downarrow$$

$$\boldsymbol{X}_0 = \boldsymbol{A}_{00}^{-1}(-\boldsymbol{A}_{01}\boldsymbol{X}_1 - \boldsymbol{A}_{02}\boldsymbol{X}_2 + \boldsymbol{\alpha}\boldsymbol{B}_0)$$
$$\boldsymbol{X}_1 = \boldsymbol{A}_{11}^{-1}(-\boldsymbol{A}_{12}\boldsymbol{X}_2 + \boldsymbol{\alpha}\boldsymbol{B}_1)$$
$$\boldsymbol{X}_2 = \alpha\boldsymbol{A}_{22}^{-1}\boldsymbol{B}_2$$

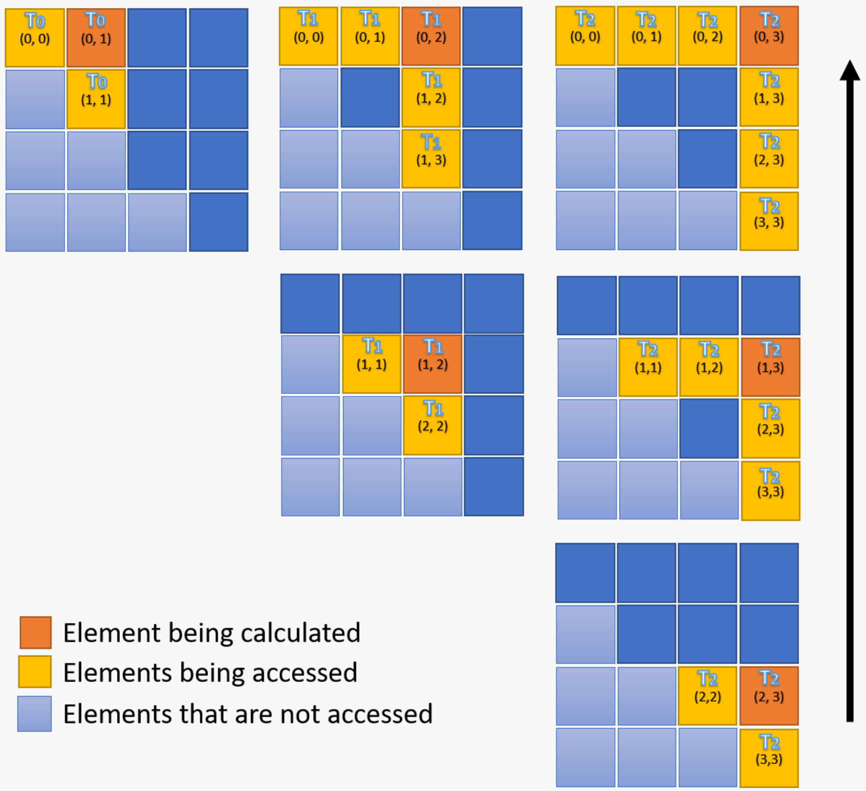
- Small blocks leads to fast inversion
- Each block can be inverted in parallel
- Can leverage existing optimized GEMM
- Data stays on the device memory

Solving a TRSM Problem using GEMM

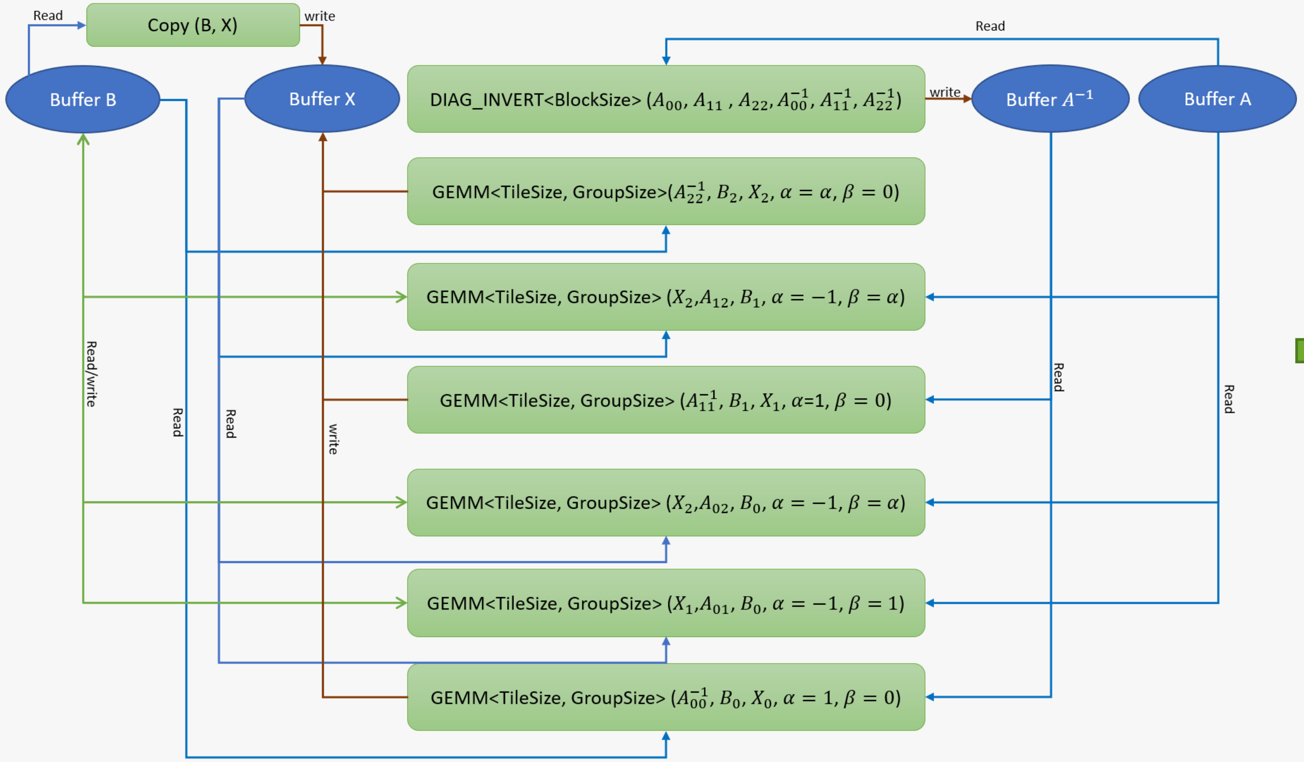
Diagonal Blocks Inversion



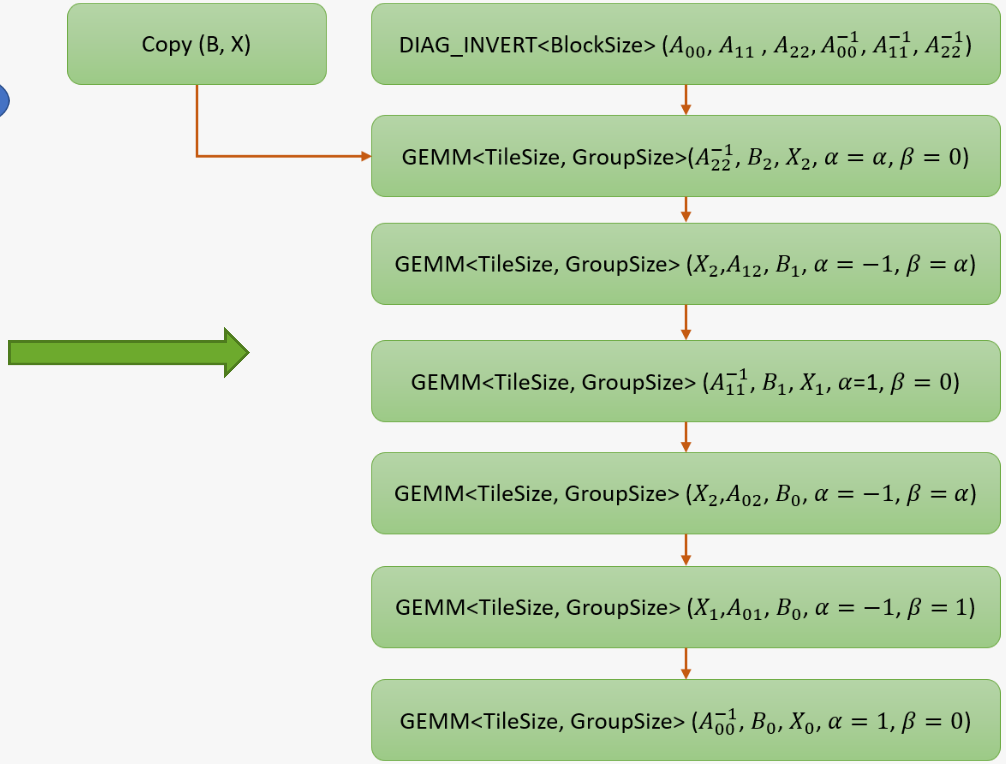
Data Locality Pattern on GPU



SYCL Kernel Execution



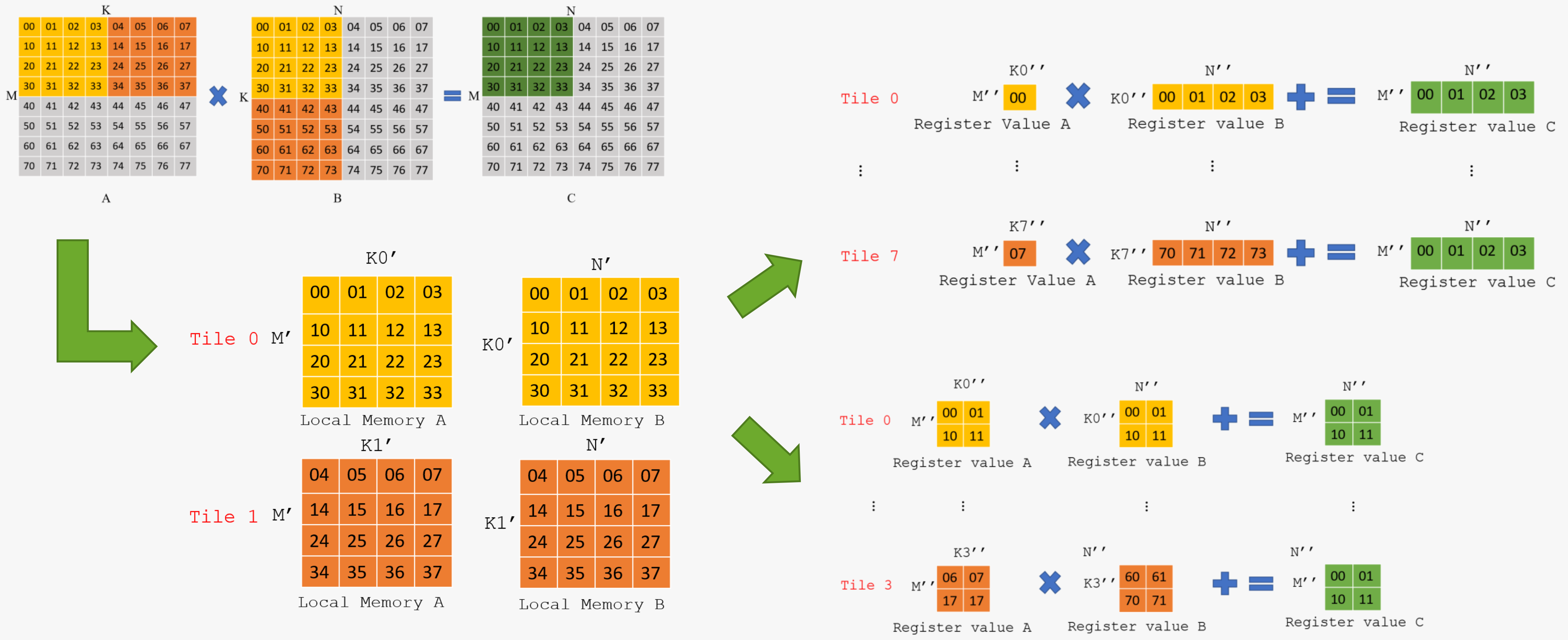
Runtime graph dependency



Runtime automated kernel scheduling

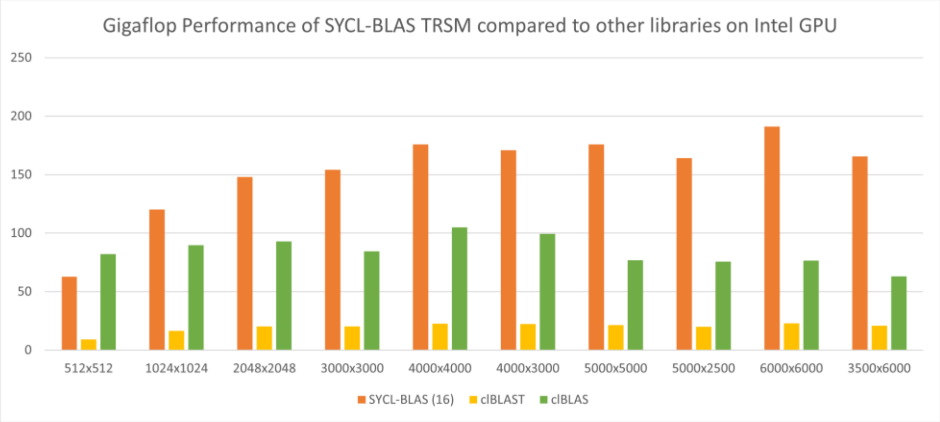
GEMM-Based TRSM

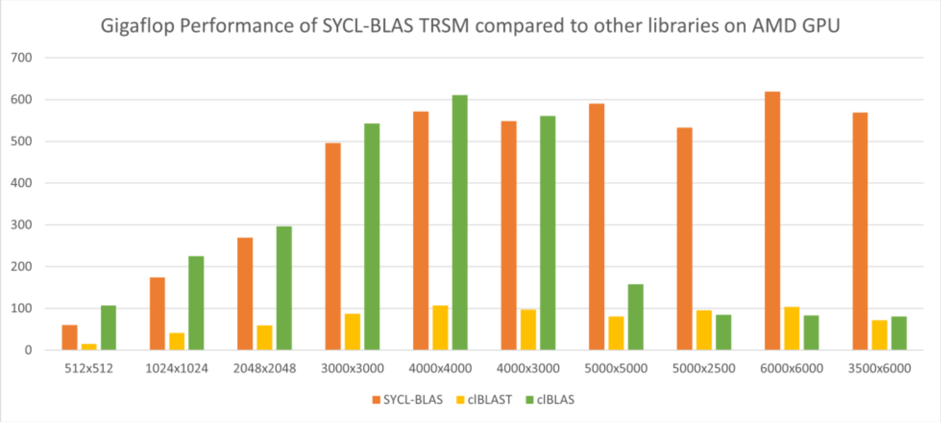
- Matrix Multiplication

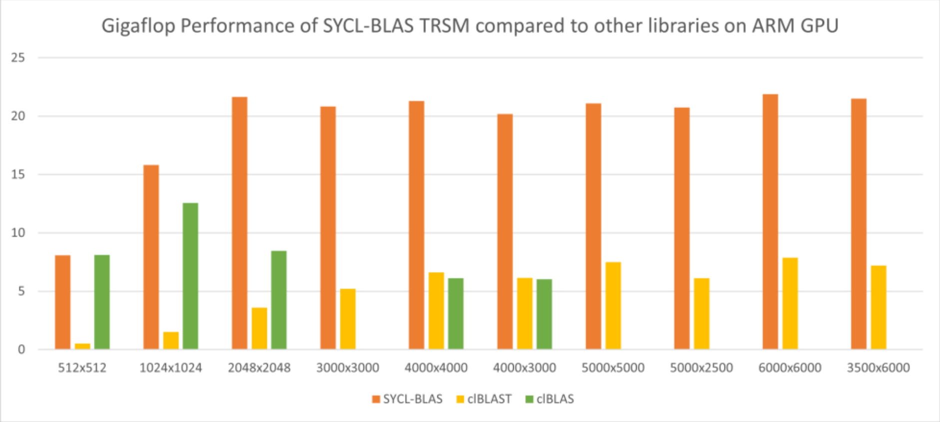


Performance Evaluation

- Hardware
  - Intel GPU UHD 630
  - ARM Mali G71
  - AMD Radeon RX460
- Libraries
  - SYCL-BLAS
  - clBLAS
  - clBlast





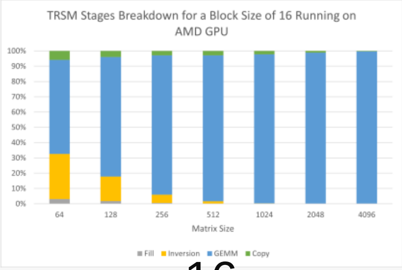


codeplay®

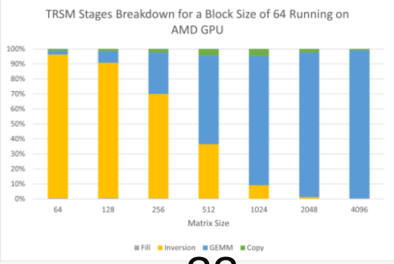
Kernel breakdown for different block sizes

Matrix sizes: 64, 128, 256, 512, 1024, 2048, 4096

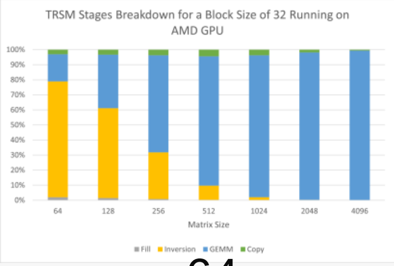
AMD Radeon RX 460



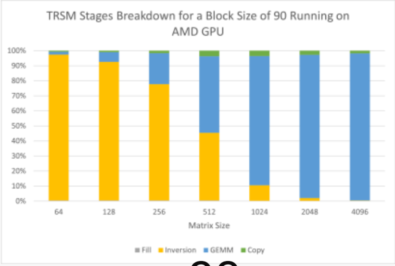
16



32

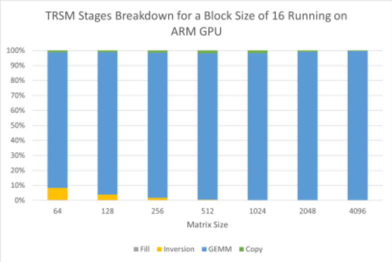


64

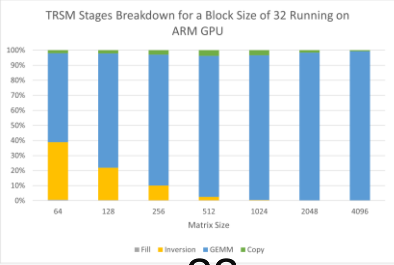


90

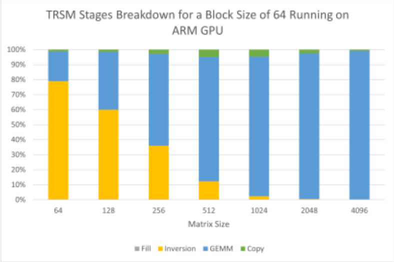
ARM Mali G71



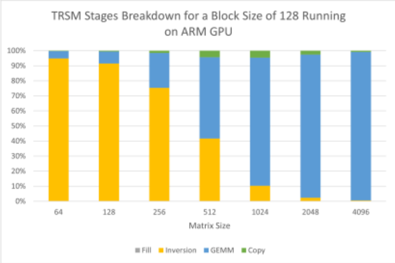
16



32

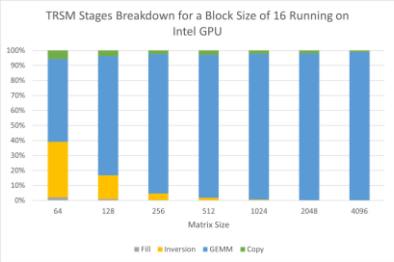


64

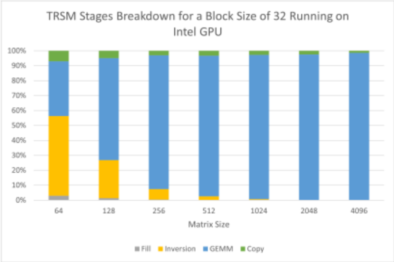


128

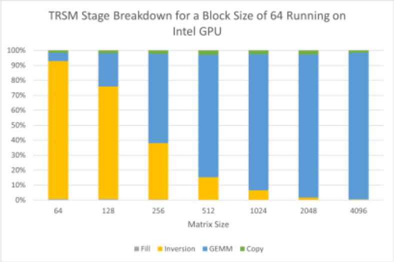
Intel GPU UHD 630



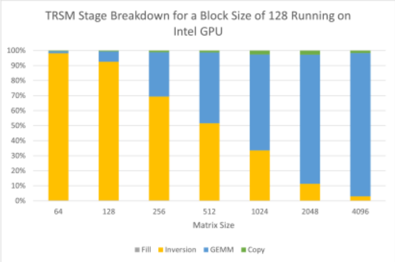
16



32



64



128

Conclusion

- Parametrizable TRSM implementation in SYCL-BLAS
- Step towards performance portability of BLAS operations
  - key component in modern HPC and embedded environments.
- Competitive performance against optimized, vendor-specified libraries
  - clBLAS and clBlast

Future Work

- Diagonal blocks of arbitrary size
- Vectorization of Block Inversion
- No local memory version for devices like the ARM Mali
- Use batched GEMM to accelerate the solver
- Evaluate performance on CPU devices

We're Hiring!
codeplay.com/careers/

codeplay®
Enabling AI to be Open, Safe & Accessible to All

Thank you!

@codeplaysoft

info@codeplay.com

codeplay.com