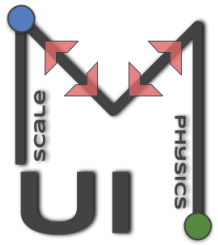
Science and Technology Facilities Council

ExCALIBUR

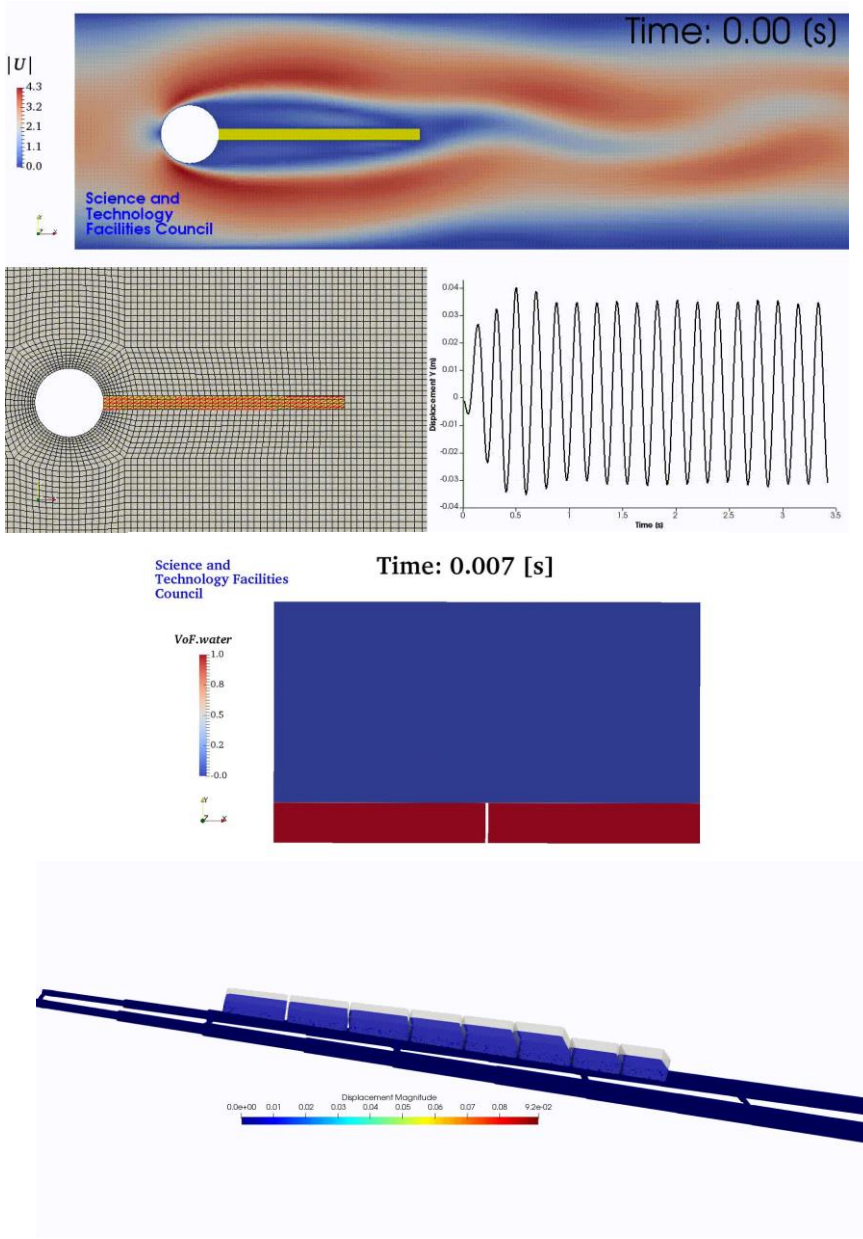
Using SYCL to accelerate the Multiscale Universal Interface coupling library

Mayank Kumar, Wendi Liu, Stephen M. Longshaw, Omar Mahfoze
UKRI Science & Technology Facilities Council, Daresbury Laboratory, UK



Motivation

- MUI is a portable, easy to use library that can couple softwares that simulate processes involving multiple phases and scales
  - Fluid-structure interaction (ParaSiF)
  - Molecular dynamics and DSMC (MNF)
- Exa-scale implementations of most scientific softwares are either ongoing or already complete
- Without a parallel implementation MUI potentially can be the limiting step in the complete simulation workflow

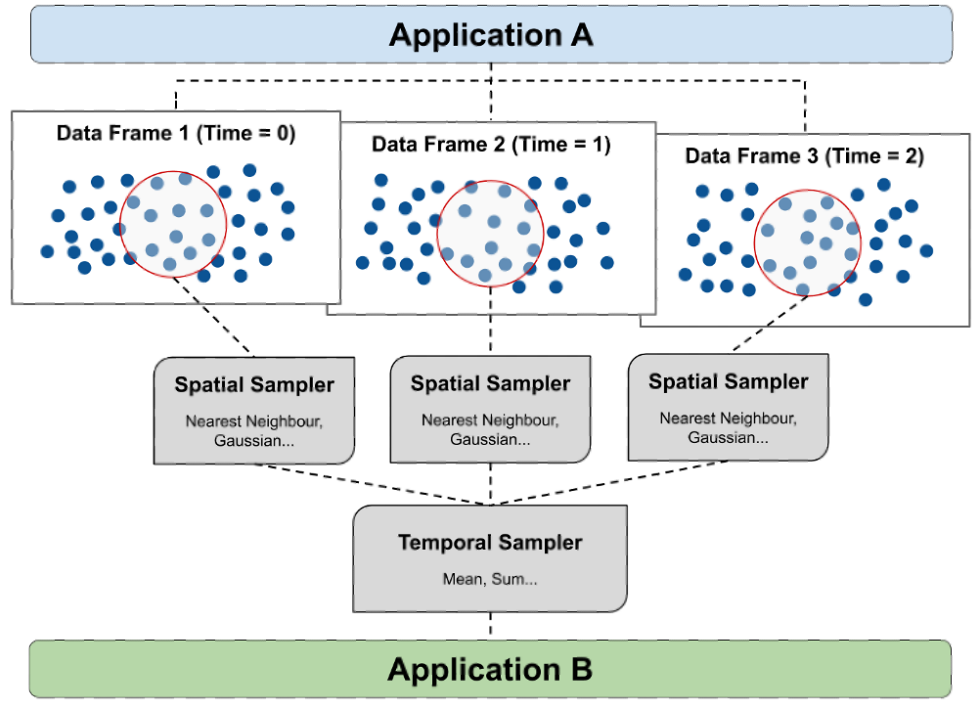


Science and Technology Facilities Council

Scientific Computing

MUI

- A concurrent interface for coupling heterogeneous solvers[1,2]
- A header-only code-coupling library written in C++
- MUI transfers data through MPI MPMD as a cloud of points, submitted as frames of time and provides spatial and temporal data sampling
- Scales to over 100,000 MPI ranks with parallel efficiency in excess of 90% depending on the codes



- Spatial Samplers
  - Exact
  - Linear
  - Quintic
  - RBF
- Temporal Samplers
  - Exact
  - Mean
  - Sum
- Algorithms
  - Fixed Relaxation
  - Aitken

Science and Technology Facilities Council

Scientific Computing

RBF filter for FSI applications

Radial Basis Function

- General form of RBF[3] on maintaining high-order consistent/conservative black-box coupling

$$s(\boldsymbol{r}) = \sum_{i=1}^{N} \alpha_i \, \varphi(\|\boldsymbol{r} - \boldsymbol{r_i}\|) + p(\boldsymbol{r})$$

- In MUI context:

$$s_i = \sum_{j=1}^{M} H_{i,j} \, \alpha_j \quad \textbf{OR} \quad \boldsymbol{s} = \boldsymbol{H} \, \boldsymbol{\alpha}$$

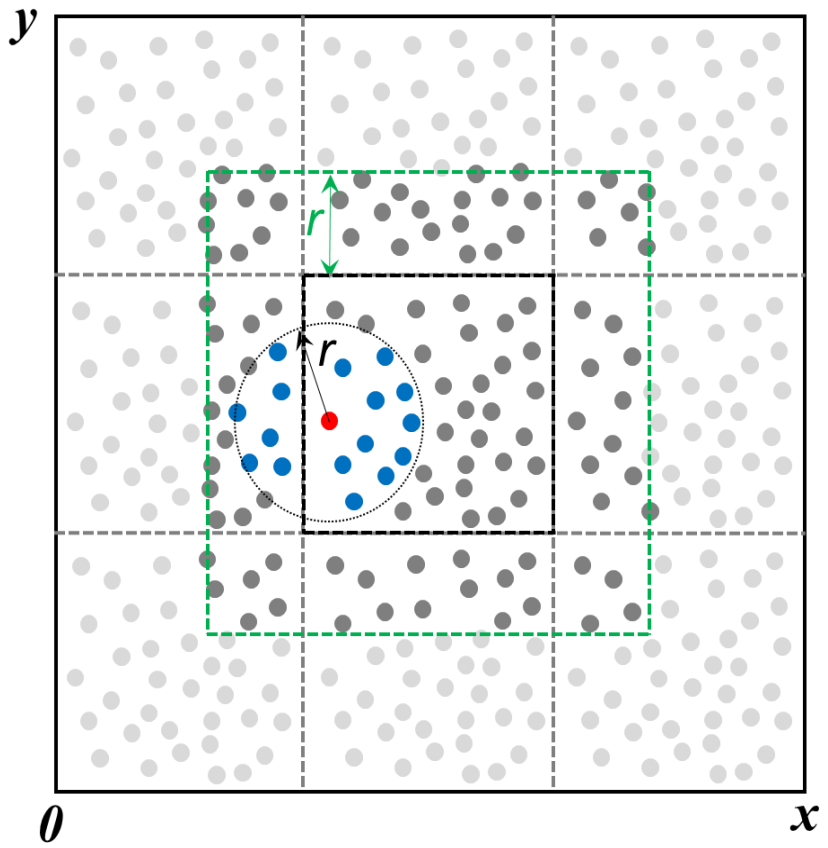
$$\boldsymbol{H} = \boldsymbol{A_{as}} \, \boldsymbol{C_{ss}^{-1}}$$

Since $\boldsymbol{C_{ss}}$ is symmetric matrix:

$$\boldsymbol{H^T} = (\boldsymbol{A_{as}} \, \boldsymbol{C_{ss}^{-1}})^T$$
$$= (\boldsymbol{C_{ss}^{-1}})^T \, \boldsymbol{A_{as}^T}$$
$$= \boldsymbol{C_{ss}^{-1}} \, \boldsymbol{A_{as}^T}$$

In which: $\boldsymbol{H^T} = \boldsymbol{C_{ss}^{-1}} \, \boldsymbol{A_{as}^T}$ can be written as: $\boldsymbol{C_{ss}} \, \boldsymbol{H^T} = \boldsymbol{A_{as}^T}$

Conjugate Gradient solver can be used to solve $\boldsymbol{H^T}$



Science and Technology Facilities Council

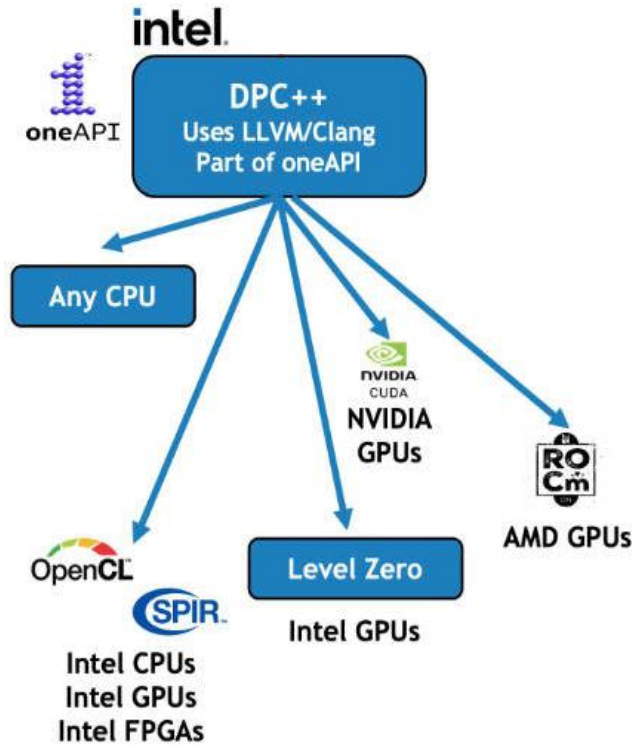
Scientific Computing

SYCL Programming model

SYCL (pronounced ‘sickle’) is a royalty-free, cross-platform abstraction layer.

Enables code for heterogeneous and offload processors to be written using modern ISO C++

Provides APIs and abstractions to find devices (e.g. CPUs, GPUs, FPGAs) on which code can be executed, and to manage data resources and code execution on those devices



SYCL USM

Allows reading and writing data using conventional pointers

Memory allocations can be explicit (device,host) or managed by SYCL runtime (shared)

SYCL BUFFER

Container for data accessed by both host and device

Memory managed by SYCL runtime using accessors

SYCL Implementation

CG Algorithm[4]

$r_0 = b - Ax_0$

$p_0 = r_0$

$i=0$

while (r_i > tol)

{

$y_i = Ap_i$

$\alpha = r_i.r_i \,/\, p_i.y_i$

$x_{i+1} = x_i + \alpha\, p_i$

$r_{i+1} = r_i - \alpha\, y_i$

$\beta = r_{i+1} \,.\, r_{i+1}/r_i \,.\, r_i$

$p_{i+1} = \beta\, p_i + r_i$

$i = i+1$

}

Serial Implementation

```
dotp_vec_vec()
{
    size_t rows = size_row;
    product = 0.;
    for (idx = 0; idx < rows; idx++)
    {
         product +=
             vec1_value[idx] * vec2_value[idx];
    });
    return (product);
}
```

Hotspots in MUI

- Generation of C_{ss} and A_{as} matrix
- Solution of H^T

SYCL Implementation

```
sycl_dotp_vec_vec()
{
    size_t rows = size_row;
    VTYPE *prod;
    prod = (VTYPE *)malloc(1);
    prod[0] = 0.;
    VTYPE *dotp;
    dotp = sycl::malloc_device<VTYPE>(1,defaultQueue);
    defaultQueue.memcpy(dotp, prod(sizeof(VTYPE))).wait();
    auto chg = [&](sycl::handler &hc)
    {
        hc.parallel_for(sycl::range(rows),[=](sycl::id<1> idx)
        {
            auto product = 0.;
            product = vec1_value[idx] * vec2_value[idx];
            auto v = sycl::atomic_ref<
                VTYPE, sycl::memory_order::relaxed,
                sycl::memory_scope::device,
                sycl::access::address_space::global_space>(*dotp);
            v.fetch_add(product);
        });
    };
    defaultQueue.submit(chg).wait();

    defaultQueue.memcpy(prod,dotp,(sizeof(VTYPE))).wait();
    return (prod[0]);
}
```

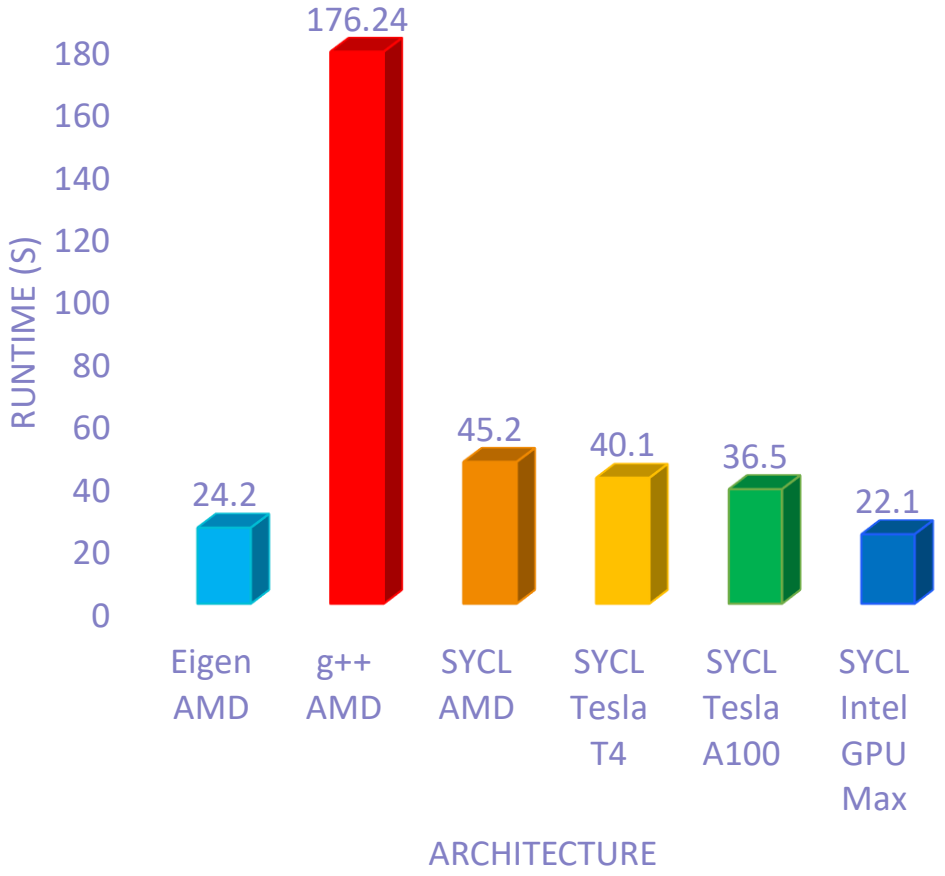
Science and Technology Facilities Council

Scientific Computing

Results

- Using a test matrix of 576*576 the SYCL code was tested for multiple architectures.
- The results are compared with benchmark linear solver Eigen running on 32 core AMD processor
- Profiling results show GPU occupancy less than 10%.

Matrix Vector product	**52.7 %**
Dot product	**42.5%**



Science and Technology Facilities Council

Scientific Computing

Conclusion and future work

- SYCL provides a promising approach for the architecture independent parallelisation of MUI
- Further optimization of the linear algebra library
- Parallelization of Css and Aas matrix generation

References

[1] Tang, Y.H., Kudo, S., Bian, X., Li, Z. and Karniadakis, G.E., 2015. Multiscale universal interface: a concurrent framework for coupling heterogeneous solvers. Journal of Computational Physics, 297, pp.13-31.

[2] Tang, Y.H., Kudo, S., Longshaw, S.M., Liu, W., Skillen, A., Mahfoze, O., Bian, X., Li, Z., Karniadakis, G.E., Nash, R.W., Richardson, C., Manson-Sawko, R., 2023. The Multiscale Universal Interface 2.0, https://doi.org/10.5281/zenodo.10054600

[3] Rendall, T.C.S. and Allen, C.B. 2009. Improved radial basis function fluid–structure coupling via efficient localized implementation. Int. J. Numer. Meth. Engng., 78: 1188-1208. https://doi.org/10.1002/nme.2526

[4] Baratta, I., Richardson, C., & Wells, G. 2022. Performance analysis of matrix-free conjugate gradient kernels using SYCL. Apollo - University of Cambridge Repository. https://doi.org/10.17863/CAM.83190