The 11th International workshop on OpenCL and SYCL

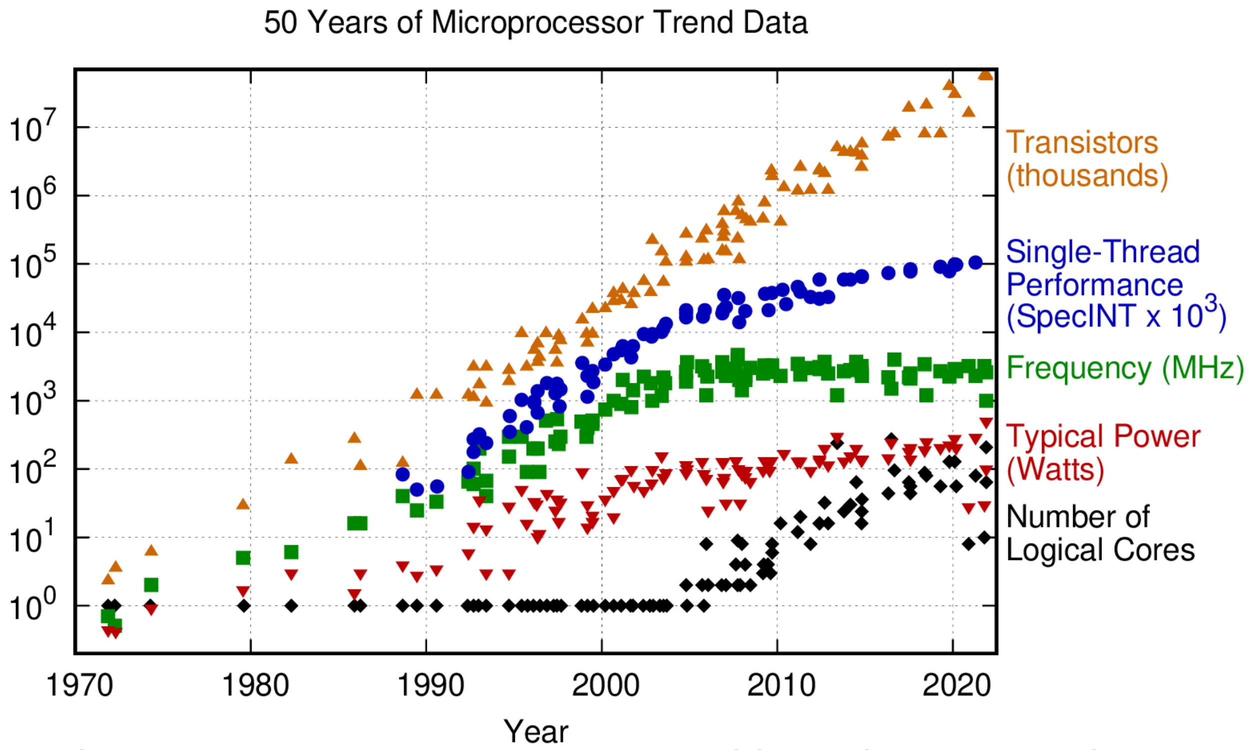
IWOCL & SYCLcon 2023

Transforming Fortran weather and climate applications to OpenCL using PSyclone

Sergi Siso, Hartree Centre STFC UKRI

with Andrew Porter and Rupert Ford, Hartree Centre STFC UKRI

The future (and present) of HPC is heterogeneous



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

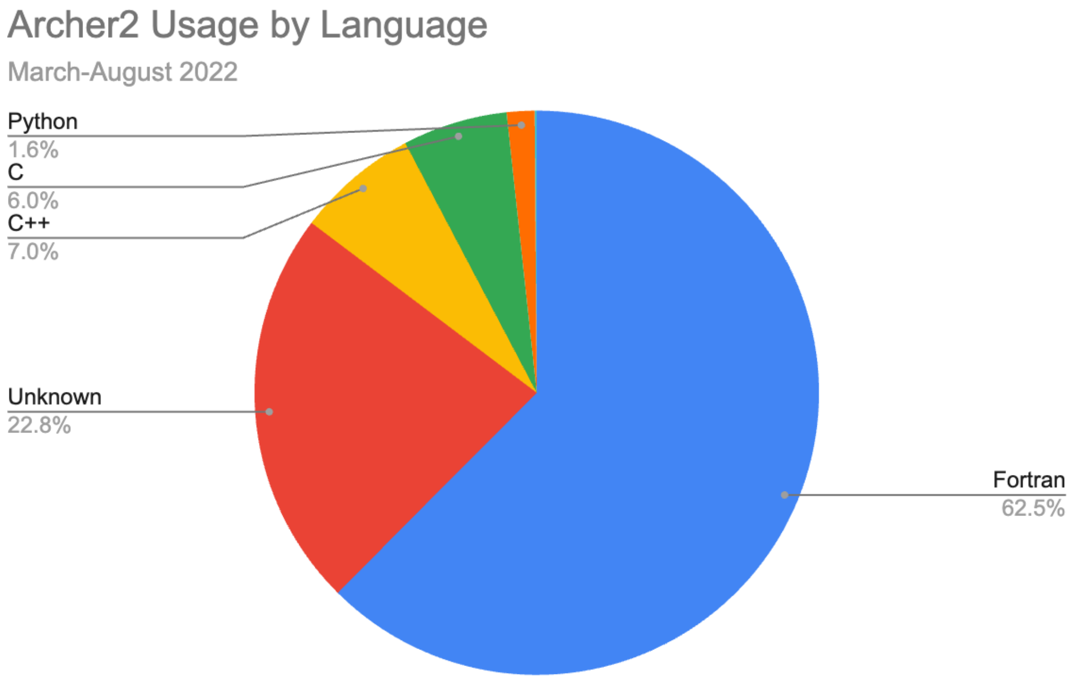
Source: https://github.com/karlrupp/microprocessor-trend-data

Top 500 list November 2022

Position	Name	Processor	Linpack (PFlop/s)
#1	Frontier	AMD EPYC 64 cores AMD Instinct MI250X	1,102
#2	Fugaku	Fujitsu A64FX 48C	442
#3	Lumi	AMD EPYC 64 cores AMD Instinct MI250X	309
#4	Leonardo	Xeon Platinum 8358 32C Nvidia A100 SMX4	174
#5	Summit	IBM POWER9 22C Nvidia V100	148
#7	Taihulight	Sunway SW26010 260C	93
#10	Tianhe-2A	Intel Xeon E5-2692v2 12C MATRIX-2000	61

And upcoming Intel GPUs, Nvidia CPUs, RISC-V, FPGAs, …

Large number of HPC applications use Fortran



Code	Language	Percentage use
VASP	Fortran	27.29%
Unidentified	Unknown	22.35%
CP2K	Fortran	6.28%
GROMACS	C	4.53%
CASTEP	Fortran	4.03%
Met Office UM	Fortran	3.10%
SENGA	Fortran	2.92%
Nektar++	C++	2.81%
NEMO	Fortran	2.46%
LAMMPS	C++	2.40%
PDNS3D	Fortran	1.89%
iIMB	Fortran	1.71%

Source: https://cpufun.substack.com/p/is-fortran-a-dead-language - Jim Cownie
https://www.archer2.ac.uk/support-access/status.html#:~:text=0.0-,Historical%20usage%20data,-Period

Software Sustainability

- HPC scientific applications are large and complex software projects.
  - Coupling of many different areas of expertise.
  - Large number of contributors from multiple institutions.
  - Some have millions of LOC.

- Productivity, readability, maintainability are essential for the sustainability of large software projects.

- Community effort: hard to maintain multiple implementations.

Ideally single source, with performance and parallelisation details abstracted.

Performance Portability Strategies

- **Maintain multiple implementations:** e.g., CUDA, HIP, OpenCL. Requires re-implementing the application in a new programming model and maintaining it over time.

- **Compiler hints/keywords**: e.g., OpenMP, OpenACC. Provide descriptive constructs. The compiler has flexibility to decide how to implement them for the target architecture.

- **Compile-time abstractions**: e.g., SYCL, Kokkos, Raja. Use C++ template metaprogramming to abstract the parallelisation API, the parallel execution order, how data structures are laid out in memory and on which space data resides.

- **Task-based parallelism**: e.g., Legion, Cabana, OmpSs, DaCe. Data-centric programming. The developer describes the dependencies between tasks and a runtime system decide how to execute them.

and what about Fortran?

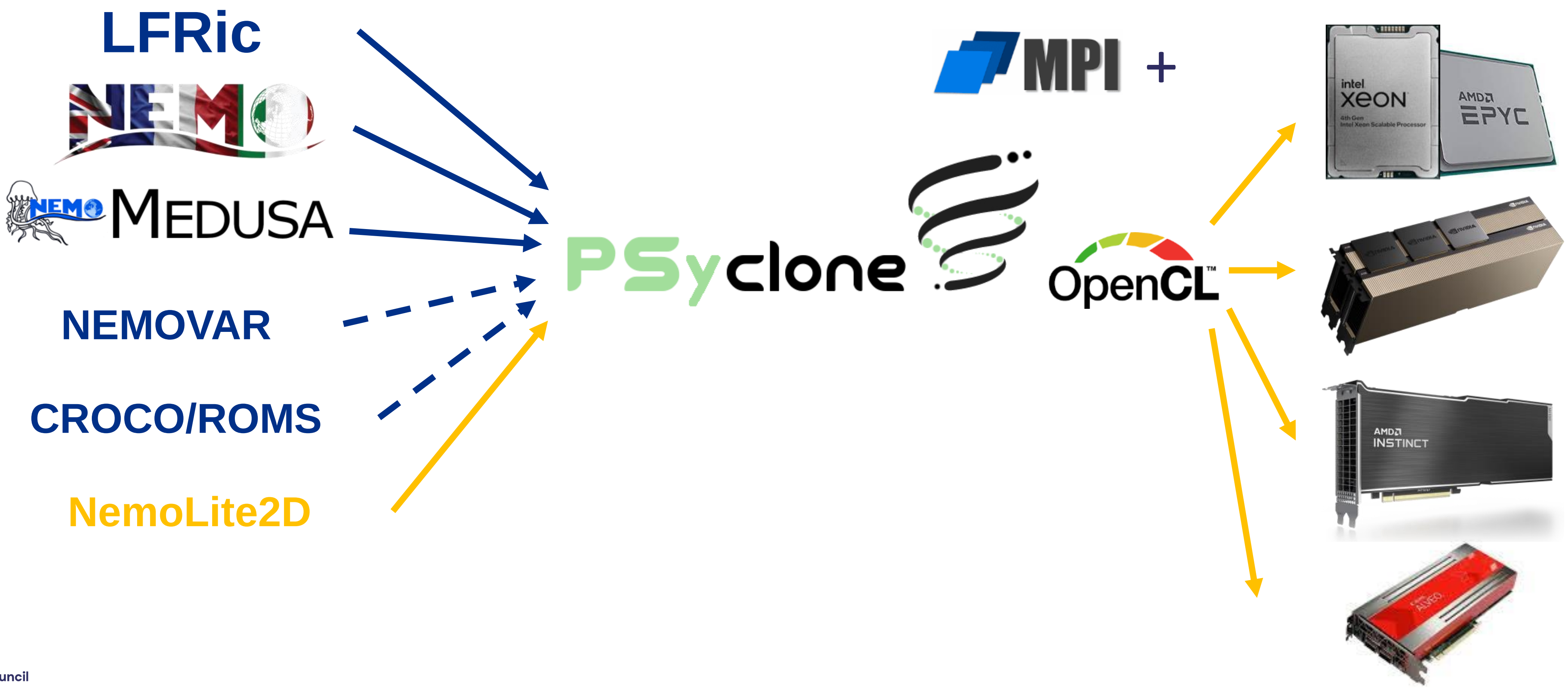
Fortran has limited heterogeneous programming capabilities and lacks the powerful compile-time mechanisms that C++ performance portability frameworks use.

- **OpenMP 5 and OpenACC**: Still used differently on CPU and GPUs. Irregular vendor and compiler support.
- **CUDA Fortran:** Proprietary, single vendor and compiler support.
- **HPF/do concurrent**: Not widely adopted. Irregular compiler support.
- **Pre-processor macros**: Sometimes used in HPC codes but impacts software sustainability.

Can performance portability be achieved by source-to-source transformations?

PSyclone: a code generation and transformation system for weather and climate Fortran applications

This work: a new PSyclone backend for OpenCL



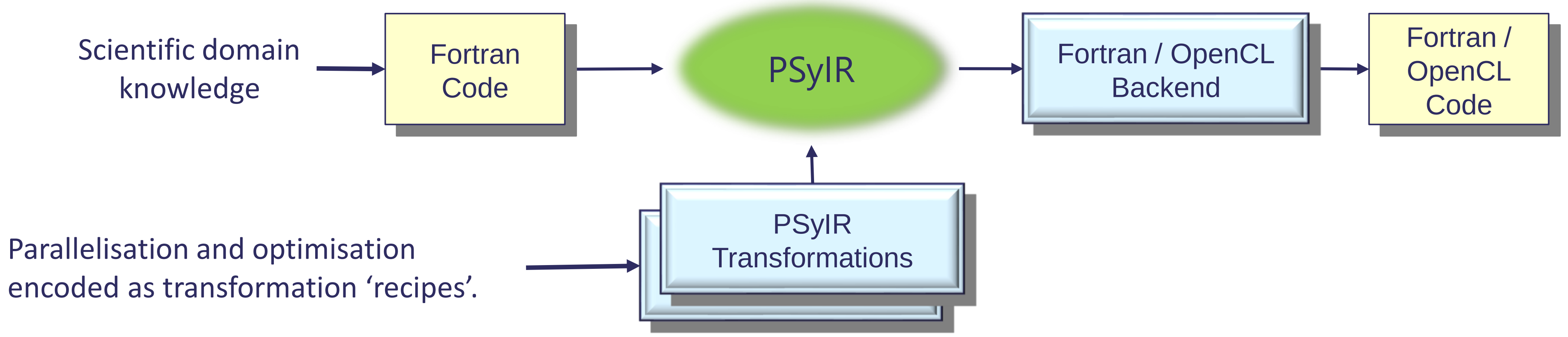
Portability != Performance Portability

- A direct mapping to a portable language backend is not enough!
- CPU, GPUs and especially FPGAs require different implementations.
- Performance portability can be improved by providing a list of code transformations (a PSyclone recipe) specific to each target platform.

PSyIR: PSyclone Intermediate Representation

- It is a **mutable representation** intended to be **programmatically manipulated** through transformations or PSyclone scripts.
- It **provides utilities** like DAG visualisations and automatic insertion of performance/debugging calipers to aid HPC experts.
- It gracefully **supports incomplete code information** like unsupported Fortran features and unresolved datatypes.
- It is itself **domain-agnostic**, but it is **extensible** to create the domain-specific DSLs that will be used by the applications.

The PSyclone workflow



Separation of concerns

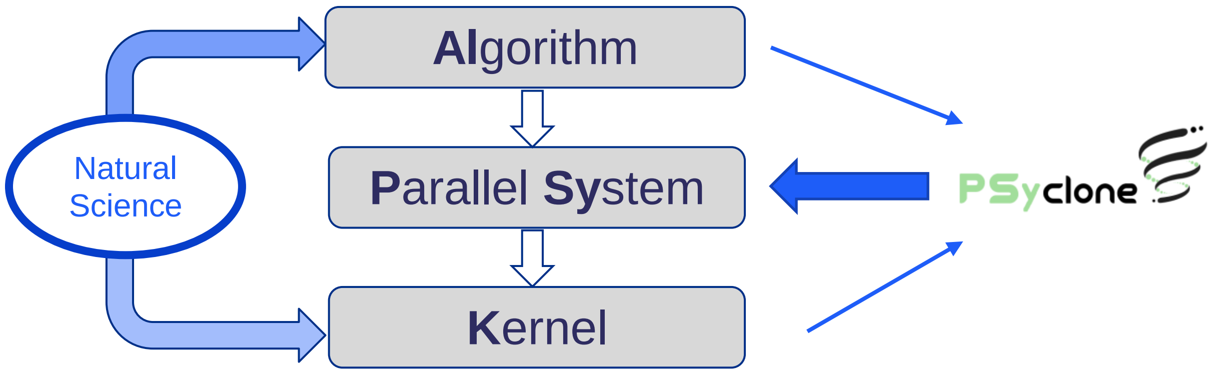
Perf portability: diff. recipes for each architecture

Visible “*readable*” output
Standard debugging/profiling

Standard output composable w. other s2s, vendor compilers

PSyKAl: a kernel-based model for Fortran

- Used in LFRic and NemoLite2D



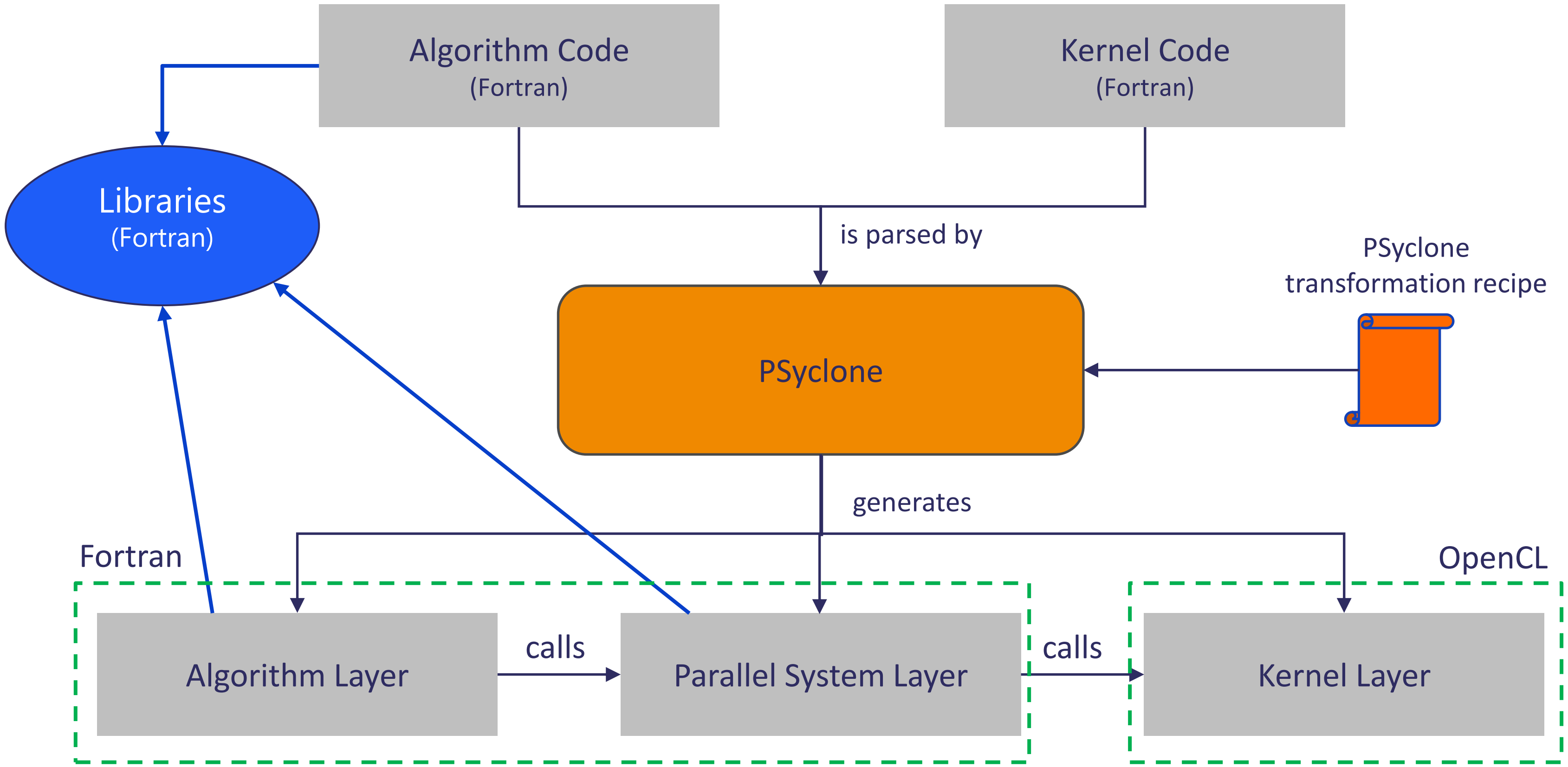

```
call invoke(continuity(ssha_t, sshn_t, sshn_u, sshn_v,    &
                       hu, hv, un, vn)                     &
            momentum_u(ua, un, vn, hu, hv, ht,             &
                       ssha_u, sshn_t, sshn_u, sshn_v),    &
            momentum_v(va, un, vn, hu, hv, ht,             &
                       ssha_v, sshn_t, sshn_u, sshn_v),    &
            bc_ssh(istp, ssha_t),                          &
            bc_solid_u(ua),                                &
            bc_solid_v(va),                                &
            bc_flather_u(ua, hu, sshn_u),                  &
            bc_flather_v(va, hv, sshn_v),                  &
            copy(un, ua),                                  &
            copy(vn, va),                                  &
            copy(sshn_t, ssha_t),                          &
            next_sshu(sshn_u, sshn_t),                     &
            next_sshv(sshn_v, sshn_t)                      &
)
```

Listing 1: PSyKAl Algorithm layer

```
type, extends(kernel_type) :: bc_solid_u
    type(go_arg), dimension(2) :: meta_args =             &
        (/ go_arg(GO_READWRITE, GO_CU, GO_POINTWISE),     &
           go_arg(GO_READ,      GO_GRID_MASK_T)           &
        /)
    integer :: ITERATES_OVER = GO_ALL_PTS
    integer :: index_offset = GO_OFFSET_NE
contains
    procedure, nopass :: code => bc_solid_u_code
end type bc_solid_u

subroutine bc_solid_u_code(ji, jj, ua, tmask)
    integer,                    intent(in)    :: ji, jj
    integer,  dimension(:,:),  intent(in)    :: tmask
    real(go_wp), dimension(:,:), intent(inout) :: ua
    if(tmask(ji,jj) * tmask(ji+1,jj) == 0)then
        ua(ji,jj) = 0._go_wp
     end if
end subroutine bc_solid_u_code
```

Listing 2: PSyKAl Kernel layer

Mapping PSyKAl to OpenCL



Simple example without optimisations

```fortran
subroutine example_kernel_code(ji, jj, array1, array2)
  use parameter_mode, only: scalar_parameter
  implicit none
  real(8),dimension(:,:),intent(inout) :: array1
  real(8),dimension(:,:),intent(in) :: array2
  array1(ji,jj) = array1(ji,jj) + array2(ji+1,jj) * &
                  scalar_parameter
end subroutine example_kernel_code
```

Listing 3: Example of a Fortran PSyKAl kernel

```python
# For each kernel in the Parallel System layer ...
for kern in schedule.kernels():
    # Convert the Globals to Arguments, since OpenCL
    # kernels do not have access to Fortran global
    # variables.
    globals_to_arguments.apply(kern)

# Transform the whole Parallel System to use OpenCL
opencl_trans.apply(schedule)
```

Listing 4: PSyclone script to generate unoptimized OpenCL

$ psyclone […options…] –s opencl_trans.py source.f90

```fortran
! Initialize OpenCL device, compile kernels and set up
! buffers if it's the first time executing this PSy-layer
if (.first_time.) initialize_device_kernel_and_buffers( &
                      example_kernel, array1, array2)

! Set up arguments in case they changes from previous
! execution of this PSy-layer
array1_cl_mem = TRANSFER(array1, array1_cl_mem)
array2_cl_mem = TRANSFER(array2, array2_cl_mem)
! Call clSetKernelArg for each OpenCL kernel argument
CALL kernel_set_args(example_kernel, array1_cl_mem,      &
                     array2_cl_mem, scalar_parameter)

! Launch the kernel
ierr = clEnqueueNDRangeKernel(cmd_queues(1),              &
            example_kernel, 2,                            &
            (/xstart - 1, ystart -1/),                    &
            (/xlen - xstart - 1, ylen - ystart - 1/),     &
            C_NULL_PTR, 0, C_NULL_PTR, C_NULL_PTR)
```

Listing 5: Generated Fortran OpenCL PSy layer

```c
__kernel void example_kernel_code(
  __global double * restrict array1,
  __global double * restrict array2,
  double scalar_parameter
  ){
  int LEN1 = get_global_size(0);
  int ji = get_global_id(0);
  int jj = get_global_id(1);
  array1[ji+jj*LEN1] = array1[ji+jj*1LEN1] + \
    array2[(ji+1)+jj*1LEN1] * scalar_parameter;
}
```

Listing 6: Generated OpenCL code

*simplified representation of the generated OpenCL code

OpenCL Optimisations

- Kernel Blocking
- Boundary Masking

```python
# For each kernel in the Parallel System layer ...
for kern in schedule.kernels():
    # Convert the Globals to Arguments, since OpenCL
    # kernels do not have access to Fortran global
    # variables.
    globals_to_arguments.apply(kern)

    # Make kernels traverse the whole domain and mask
    # out the computations in the boundary values
    move_boundaries_trans.apply(kern)

    # Provide a block size
    koptions['local_size'] =  64

    kern.set_opencl_options(koptions)

# Transform the whole Parallel System to use OpenCL
opencl_trans.apply(schedule)
```

Listing 7: PSyclone script to generate OpenCL

```fortran
! Set up arguments in case they changes from previous
! execution of this PSy-layer
globalsize = (/grid%nx, grid%ny/)
localsize = (/64, 1/)
array1_cl_mem = TRANSFER(array1, array1_cl_mem)
array2_cl_mem = TRANSFER(array2, array2_cl_mem)
! Call clSetKernelArg for each OpenCL kernel argument
CALL kernel_set_args(example_kernel, array1_cl_mem, &
&          array2_cl_mem, scalar_parameter, &
&          xstart - 1, xstop - 1, ystart - 1, ystop - 1)

! Launch the kernel
ierr = clEnqueueNDRangeKernel(cmd_queues(1), &
&            example_kernel, 2, C_NULL_PTR, &
&            C_LOC(globalsize), C_LOC(localsize), 0, &
&            C_NULL_PTR, C_NULL_PTR)
```

Listing 8: Generated Fortran OpenCL Parallel-System layer

```c
__attribute__((reqd_work_group_size(64, 1, 1)))
__kernel void example_kernel_code(
  __global double * restrict array1,
  __global double * restrict array2,
  double scalar_parameter,
  int xstart, int xstop, int ystart, int ystop
  ){
  int LEN1 = get_global_size(0);
  int ji = get_global_id(0);
  int jj = get_global_id(1);
  if ((((ji < xstart) || (ji > xstop)) || \
       ((jj < ystart) || (jj > ystop)))) {
    return;
  }
  array1[ji+jj*LEN1] = array1[ji+jj*1LEN1] + \
    array2[(ji+1)+jj*1LEN1] * scalar_parameter;
}
```

Listing 9: Generated OpenCL code

NemoLite2D (https://github.com/stfc/PSycloneBench/tree/master/benchmarks/nemo/nemolite2d)

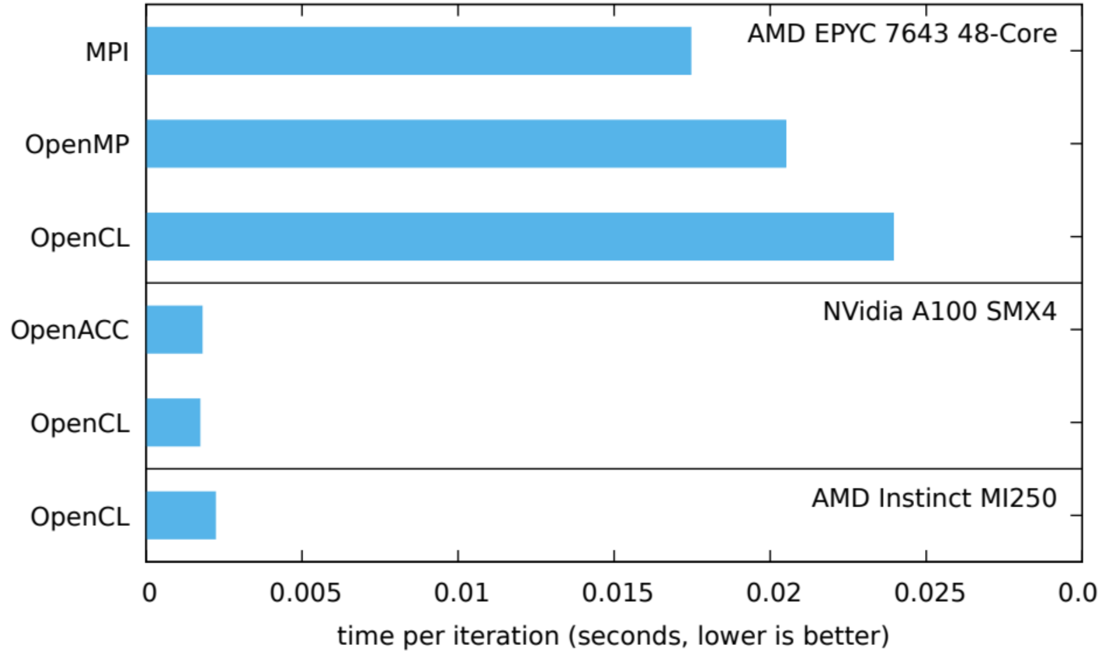
- A vertically-averaged version of the free-surface component of the NEMO model.

- Implements a continuity equation for the update of the sea-surface height and two vertically-integrated momentum equations for the two velocity components.

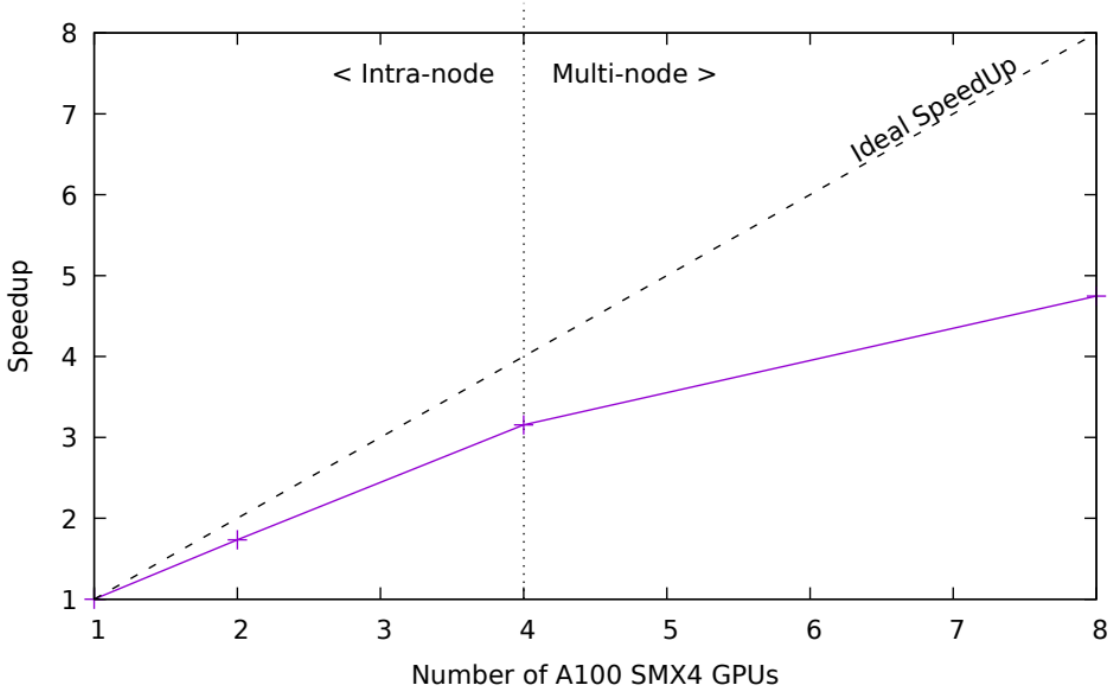
Kernel Name	Description	Fortran Lines
continuity	Eulerian forward time stepping method.	21
momentum_u	Compute advection, viscosity, coriolis force, ... on the u-field.	112
momentum_v	Compute advection, viscosity, coriolis force, ... on the v-field.	117
bc_ssh	Clamped boundary conditions.	26
bc_solid_u	Solid boundary conditions for u-velocity.	11
bc_solid_v	Solid boundary conditions for v-velocity.	11
bc_flather_u	Flather open boundary condition for u-velocity.	29
bc_flather_v	Flather open boundary condition for v-velocity.	29
copy	Copy the value from one array to another.	10
next_sshu	Time update of the u-field.	21
next_sshv	Time update of the v-field.	21

Performance Results

Performance comparison of NEMOLite2D (size 2048^2) with multiple parallel programming models on multiple devices



Strong scalability of NEMOLite2D (size 6000^2) with hybrid MPI and OpenCL



Best results for each device corresponds to 63, 60, 32 % of peak bandwidth respectively

- 48-core AMD EPYC 7643 CPU using the Intel OpenCL Runtime for CPUs and compiled with gfortran 9.4
- NVIDIA A100 SMX4 GPU using the NVIDIA OpenCL drivers and compiled with nvfortran 22.5
- AMD Instinct MI250 GPU using the ROCM 5.4 OpenCL drivers and compiled with gfortran 9.4

Dynamic Evaluation of Runtime Invariants

```
subroutine example_kernel(ji, jj, array, ct1, ct2)
  implicit none
  integer, intent(in) :: ji, jj, ct1, ct2
  real(go_wp), dimension(:,:), intent(inout) :: array
  integer :: i

  do i = 1, ct1
      va(ji,jj) = va(ji,jj) / ct2
  enddo
end subroutine kernel
```

Listing 10: Test kernel for dynamic optimizations

```
# Transform the Parallel System to use OpenCL
# with captured runtime invariants provided as
# pre-processor constants
opencl_trans.apply(
    schedule, {'define_runtime_invariants': True})
```

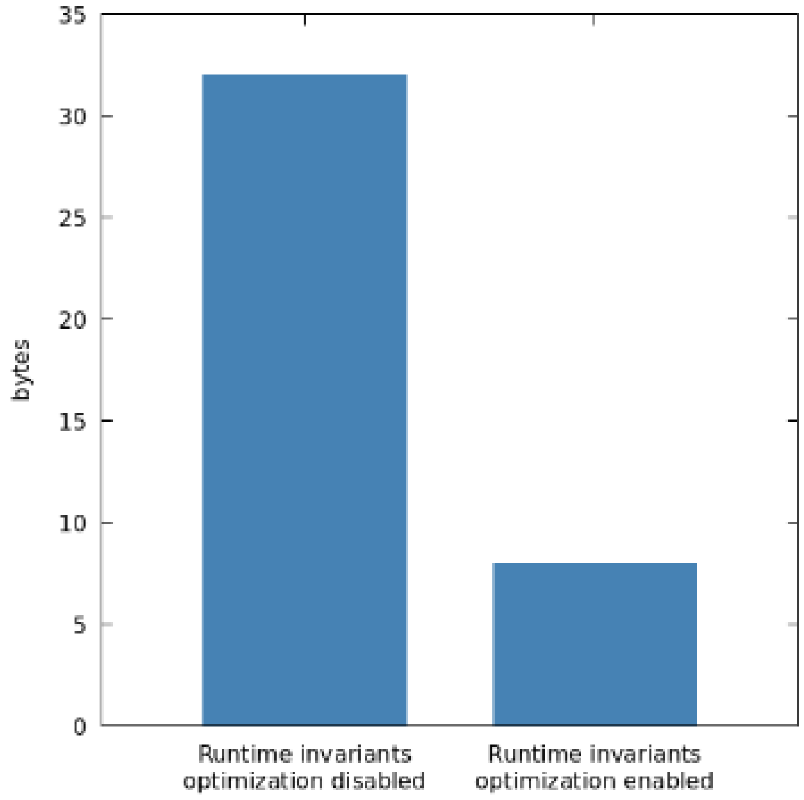
Listing 11: PSyclone recipe with OpenCL dynamic optimizations

The generated OpenCL code replaces ct1 and ct2 with undeclared file scope symbols

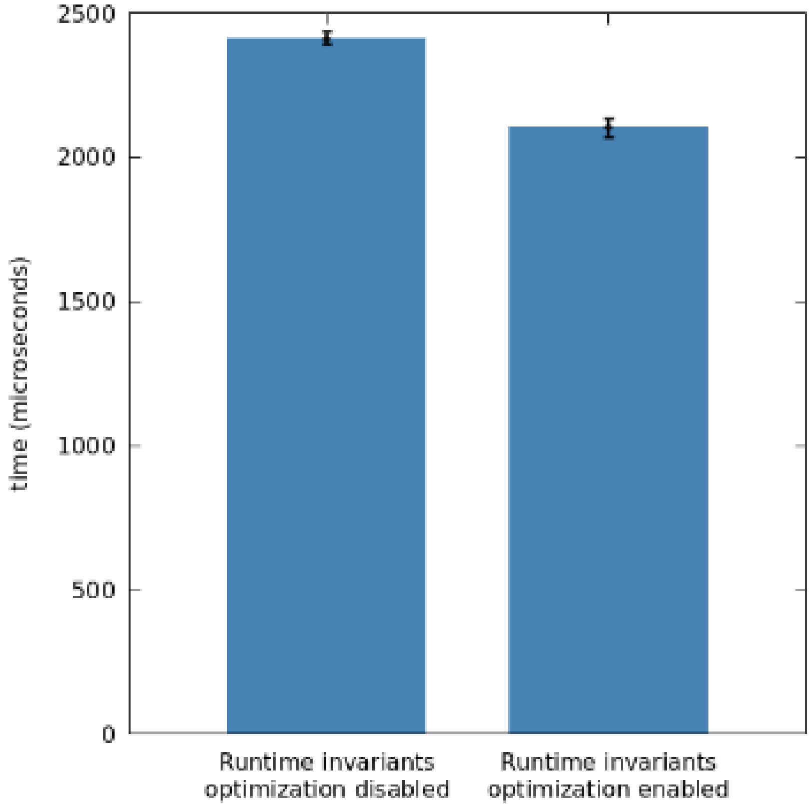
```
CHARACTER(LEN=4096) compiler_flags
WRITE (compiler_flags, *) ""
WRITE (compiler_flags, '(A,A,I0)') TRIM(compiler_flags),
    " -Dxstart_example_kernel=", xstart
WRITE (compiler_flags, '(A,A,I0)') TRIM(compiler_flags),
    " -Dxstop_example_kernel=", xstop
WRITE (compiler_flags, '(A,A,I0)') TRIM(compiler_flags),
    " -Dystart_example_kernel=", ystart
WRITE (compiler_flags, '(A,A,I0)') TRIM(compiler_flags),
    " -Dystop_example_kernel=", ystop
WRITE (compiler_flags, '(A,A,F0)') TRIM(compiler_flags),
    " -Dcaptured_ct1=", ct1
WRITE (compiler_flags, '(A,A,F0)') TRIM(compiler_flags),
    " -Dcaptured_ct2=", ct2
kernel_names(1) = 'example_kernel'

! OpenCL Runtime Compilation
CALL add_kernels(1, kernel_names, &
                 compiler_flags=compiler_flags)
```

Listing 12: OpenCL driver code generated to capture runtime invariant values

Dynamic Evaluation of Runtime Invariants



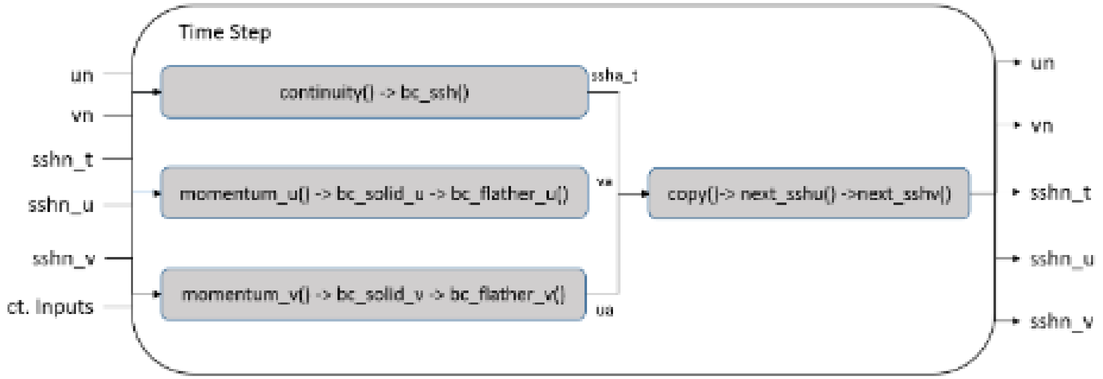
Number of bytes sent by clSetKernelArg



Execution time of the test kernel on an 8-core Intel Xeon Silver 4215

Targeting FPGAs: the EuroEXA project

- Required significant transformations from CPU/GPU code:
  - Functional parallelism (OCL queues)
  - Duplicate kernels
  - Inline loops into kernel (taskify)
  - Buffer burst memory operations to local memory.



This research has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 754337.

Performance on Xilinx U200 FPGA

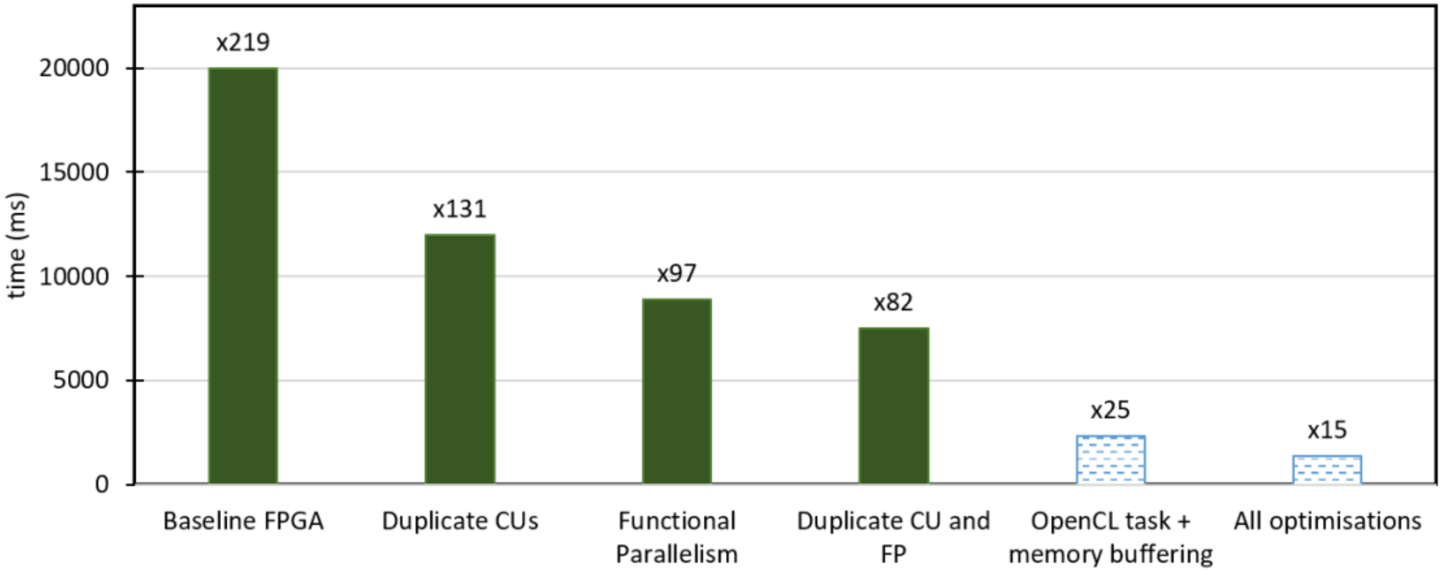


Figure: Execution time (Y axis) and slowdown compared to 1 Xeon Silver 4215 core (inverse of speedup - top of the bars) of multiple OpenCL optimizations on a Xilinx U200 FPGA. Solid bars are as generated by PSyclone, dashed bars required manual tweaks.

Current limitations:

- Only using 1 DDR memory bank and 1 SLR in the Xilinx U200 (out of 4 DDR memory banks and 3 SLR)
- Not using OpenCL pipes for faster communication between kernels.
- Not using OpenCL vendor extensions such as xcl_dataflow, xcl_pipeline_loop or xcl_pipeline_workitems.

Science and Technology Facilities Council
Hartree Centre

This research has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 754337.

EUROEXA
FUNDED BY THE EUROPEAN UNION

General applicability of Fortran-to-OpenCL

Application specific

- NemoLite2D
- PSyKAl
- Any Numerical Operations

Today

Standard Fortran

NEMO example:

- Infer kernels from Fortran array notation and dependency analysis
- Deduces domain specific knowledge from loop patterns and naming conventions of the code style-guide

```
5: If[]
    BinaryOperation[operator:'AND']
        Reference[name:'ln_wave']
        Reference[name:'ln_sdw']
    Schedule[]
        0: Loop[type='levels', field_space='None', it_space='None']
            Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
            Reference[name:'jpkm1']
            Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
            Schedule[]
                0: Loop[type='lat', field_space='None', it_space='None']
                    Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                    Reference[name:'jpj']
                    Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                    Schedule[]
                        0: Loop[type='lon', field_space='None', it_space='None']
                            Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                            Reference[name:'jpi']
                            Literal[value:'1', Scalar<INTEGER, UNDEFINED>]
                            Schedule[]
                                0: InlinedKern[]
                                    Schedule[]
                                        0: Assignment[]
                                            ArrayReference[name:'zun']
                                                Reference[name:'ji']
                                                Reference[name:'jj']
                                                Reference[name:'jk']
                                            BinaryOperation[operator:'MUL']
```

Conclusion

PSyclone enables automatic **Fortran to OpenCL** transformation for codes adhering to the **PSyKAl kernel-based** parallelism model. **Separation of concerns** and **performance-portability** are achieved by providing a recipe of code transformations.

Future work

- Generalise solution to support any numerical operations
- More performance portability IR transformations
- SYCL backend

Science and Technology Facilities Council

Hartree Centre

Thank you

sergi.siso@stfc.ac.uk