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

Top 500 list November 2022

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

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

And upcoming Intel GPUs, Nvidia CPUs, RISC-V, FPGAs, ...
Large number of HPC applications use Fortran

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%20to%20Historical%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

- Scientific domain knowledge
- Fortran Code
- PSyIR
- Fortran / OpenCL Backend
- Fortran / OpenCL Code

Parallelisation and optimisation encoded as transformation ‘recipes’.

Separation of concerns
Perf portability: diff. recipes for each architecture
Visible “readable” output
Standard debugging/profiling
Standard output composable w. other s2s, vendor compilers

PSyIR

Transformations
PSyKAI: a kernel-based model for Fortran

- Used in LFRic and NemoLite2D

**Listing 1: PSyKAI Algorithm layer**

```
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), &
bcsolid_u(ua), &
bcsolid_v(va), &
bc_flather_u(ua, hu, sshn_u), &
bcsolid_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 2: PSyKAI Kernel 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(jj, jj) = 0.0_go_wp
  end if
end subroutine bc_solid_u_code
```
Mapping PSyKAI to OpenCL

PSyclone transformation recipe

Algorithm Code (Fortran) is parsed by PSyclone, which generates a transformation recipe.

Libraries (Fortran) provide Fortran code that is called by Algorithm Layer. Algorithm Layer calls Parallel System Layer, which in turn calls Kernel Layer in OpenCL.

Kernel Code (Fortran) is also called by the Parallel System Layer.

The diagram illustrates the flow of code and the layers that form the mapping from PSyKAI to OpenCL.
Simple example without optimisations

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

Listing 3: Example of a Fortran PSyKAL kernel

# For each kernel in the Parallel System layer ...
for kern in schedule.kernels():
  # Convert theGlobals 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
openc1_trans.apply(schedule)

Listing 4: PSyclone script to generate unoptimized OpenCL

$ psyclone [...options...] --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

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

Listing 6: Generated OpenCL code

*simplified representation of the generated OpenCL code*
OpenCL Optimisations

- Kernel Blocking
- Boundary Masking

Listing 7: Psyclone script to generate OpenCL

```
# 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 8: Generated Fortran OpenCL Parallel-System layer

```
! 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 c1SetKernelArg 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 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.
Performance Results

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

- 48-core AMD EPYC 7643 48-Core 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

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
- NVIDIA A100 SMX4 GPU
- AMD Instinct MI250 GPU

![Graph showing performance comparison and scalability](image-url)
Dynamic Evaluation of Runtime Invariants

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

```
1 subroutine example_kernel(ji, jj, array, ct1, ct2)
2 implicit none
3 integer, intent(in) :: ji, jj, ct1, ct2
4 real(go_wp), dimension(:,,:), intent(inout) :: array
5 integer :: i
6 do i = 1, ct1
7    va(ji,jj) = va(ji,jj) / ct2
8  enddo
9 end subroutine kernel

Listing 10: Test kernel for dynamic optimizations

```

```
1 CHARACTER(LEN=4096) compiler_flags
2 WRITE (compiler_flags, *) ''
3 WRITE (compiler_flags, '(A,A,I0)') TRIM(compiler_flags),
4   ''-Dxstart_example_kernel=', xstart
5 WRITE (compiler_flags, '(A,A,I0)') TRIM(compiler_flags),
6   ''-Dxstop_example_kernel=', xstop
7 WRITE (compiler_flags, '(A,A,I0)') TRIM(compiler_flags),
8   ''-Dystart_example_kernel=', ystart
9 WRITE (compiler_flags, '(A,A,I0)') TRIM(compiler_flags),
10  ''-Dystop_example_kernel=', ystop
11 WRITE (compiler_flags, '(A,A,F0)') TRIM(compiler_flags),
12  ''-Dcaptured_ct1=', ct1
13 WRITE (compiler_flags, '(A,A,F0)') TRIM(compiler_flags),
14  ''-Dcaptured_ct2=', ct2
15 kernel_names(1) = 'example_kernel'
16
17 ! OpenCL Runtime Compilation
18 CALL add_kernels(1, kernel_names, &
19    compiler_flags=compiler_flags)

Listing 11: PSyCline recipe with OpenCL dynamic optimizations

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

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.

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.

This research has received funding from the European Union’s Horizon 2020 research and innovation programme under Grant Agreement no. 754337.
General applicability of Fortran-to-OpenCL

Application specific

• NemoLite2D

• PSyKAI

• Any Numerical Operations

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
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
Thank you

sergi.siso@stfc.ac.uk