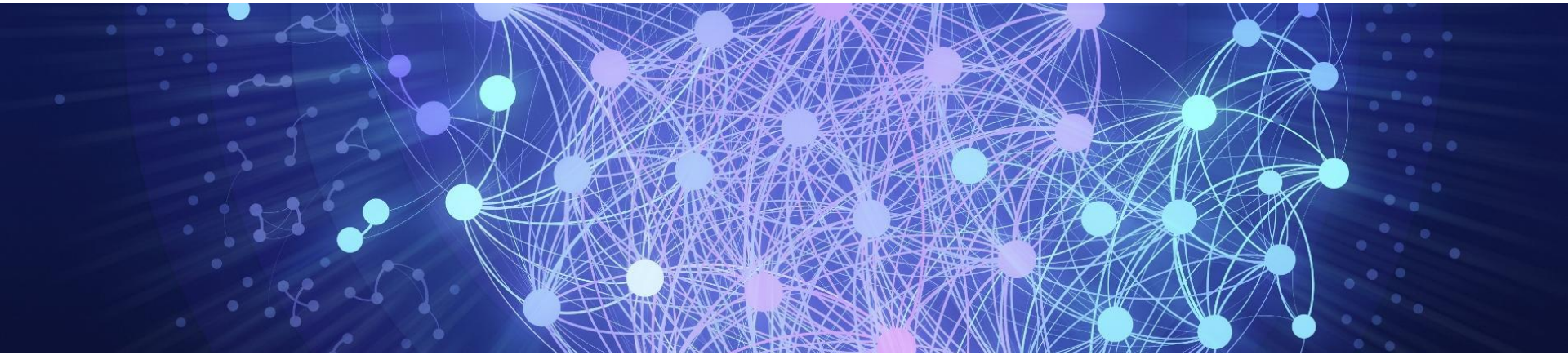
Hartree Centre
Science & Technology Facilities Council

NEMO-DSL Developments

Andrew Porter & Rupert Ford
STFC Hartree Centre | UK



What is a Domain-Specific Language and Why Do We Care?

- Relatively large code base
  - Continuously evolving as new science added
- Majority of code performs stencil operations
- ~Identical operations performed at a very large number of grid points
  - Data parallelism
- Very flat performance profile
  - Optimisations typically must be applied across ***whole*** code base
  - Different machines will benefit from different optimisations!
- Many users and therefore many different computers

Hartree Centre
Science & Technology Facilities Council

Domain-Specific Language

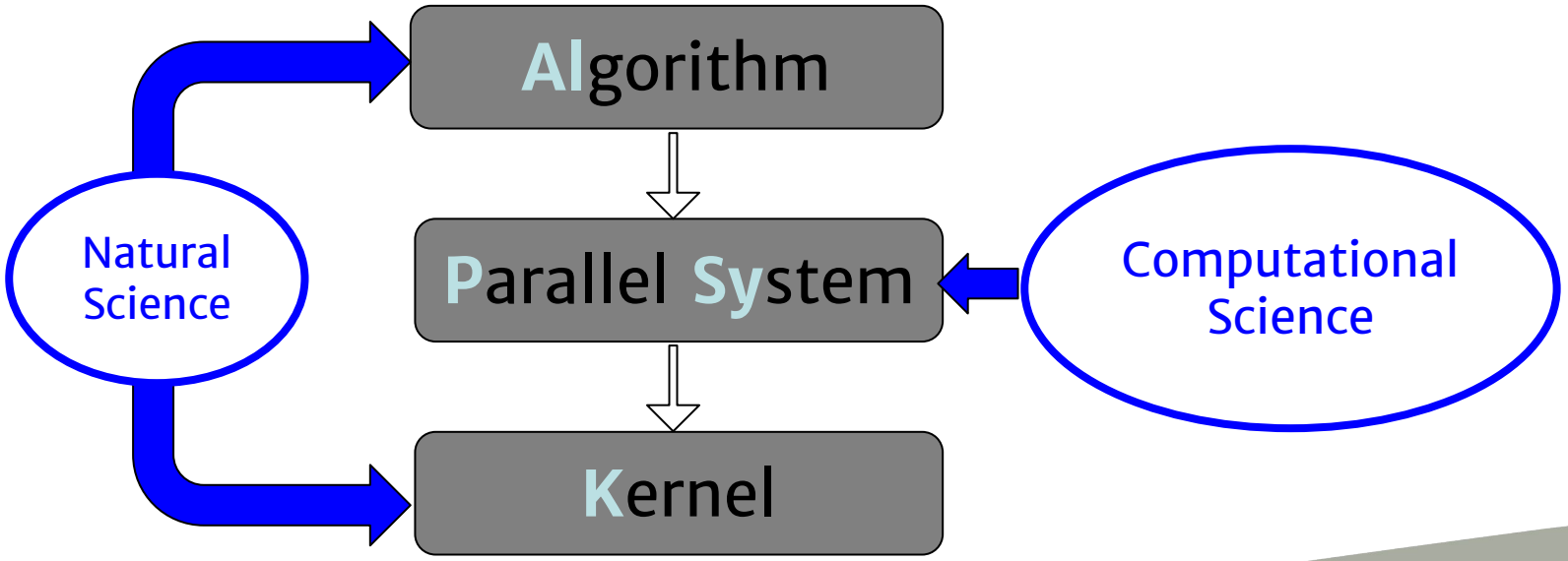
High-level representation of the scientific problem which captures what needs to be done but (ideally) not how it is implemented

Increase the productivity of natural scientists by removing implementation-specific considerations (e.g. halo swaps)

Allow HPC experts to produce different implementations for different computer architectures

PSyclone

- A Domain-Specific Compiler designed for finite-{element, volume, difference} Earth-System models
- Targets Fortran code structured as:

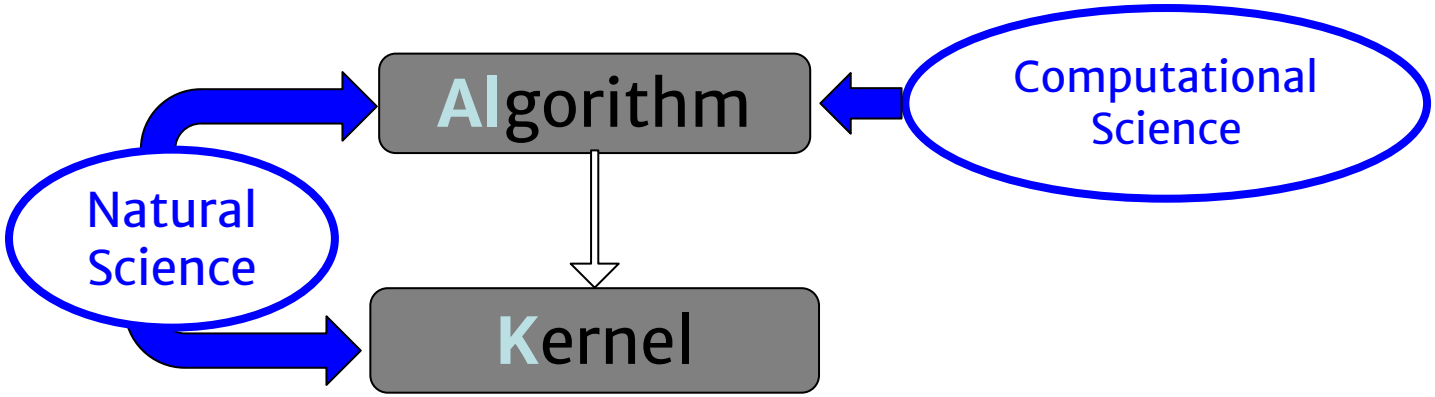


NEMO does not look like this!

Hartree Centre
Science & Technology Facilities Council

PSyclone + NEMO

We are building an interface to PSyclone based on the existing NEMO coding standards

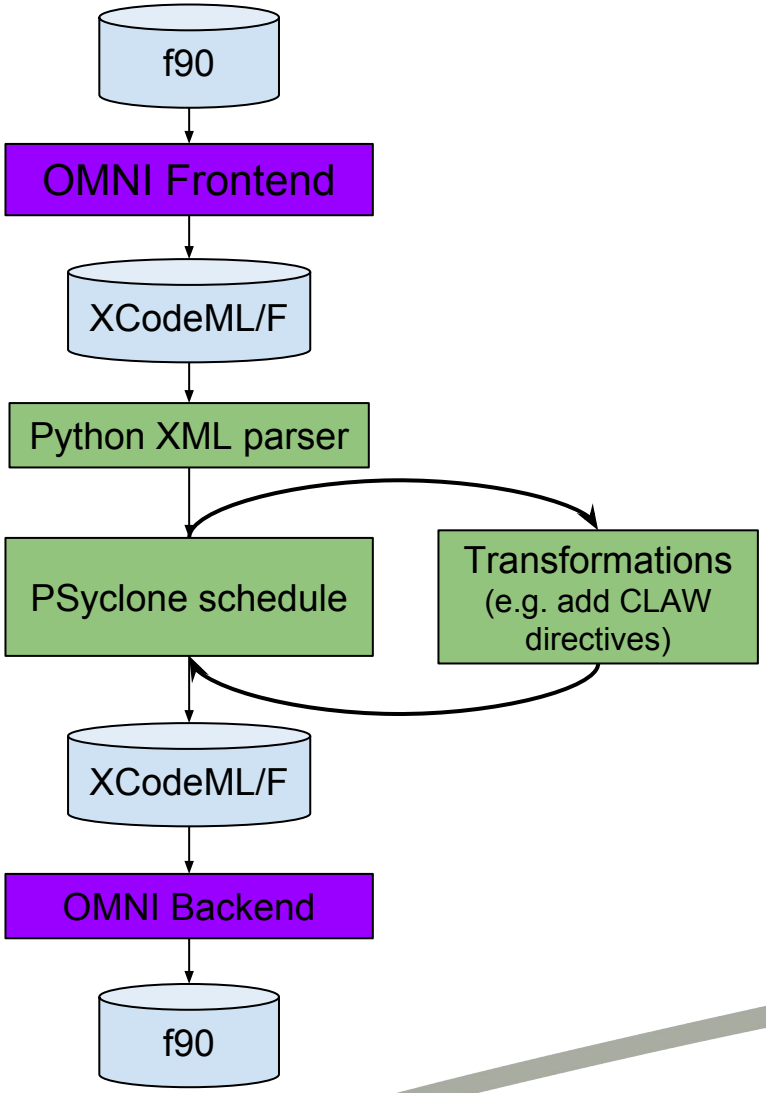


Hartree Centre
Science & Technology Facilities Council

Aims

- Allow NEMO developers to follow existing coding structure rules
  - can act as a style checker by reporting 'non-conformant' code
- Introduce OpenACC for all recognised ‘kernels’
  - NVIDIA have previously shown significant performance gains when running NEMO on GPUs
- Leave distributed memory parallelism unchanged for the moment

Approach



- Process NEMO code and create an internal PSyclone representation of it.
- Representation can then be manipulated with PSyclone transformations.

Hartree Centre
Science & Technology Facilities Council

Example: Tracer Advection

```
DO jk = 1, jpk-1
   DO jj = 2, jpj
      DO ji = 2, jpi
         zslpx(ji,jj,jk) = SIGN( 1.d0, zslpx(ji,jj,jk) ) * &
         &    MIN( ABS(zslpx(ji  ,jj,jk) ),                &
         &         2.d0*ABS( zwx  (ji-1,jj,jk) ),          &
         &         2.d0*ABS( zwx  (ji  ,jj,jk) ) )
         zslpy(ji,jj,jk) = SIGN( 1.d0, zslpy(ji,jj,jk) ) * &
         &    MIN( ABS(zslpy(ji,jj  ,jk) ),                &
         &         2.d0*ABS( zwy  (ji,jj-1,jk) ),          &
         &         2.d0*ABS( zwy  (ji,jj  ,jk) ) )
      END DO
   END DO
END DO

DO jk = 1, jpk-1
   zdt  = 1
   DO jj = 2, jpj-1
      DO ji = 2, jpi-1
          z0u = SIGN( 0.5d0, pun(ji,jj,jk) )
          zalpha = 0.5d0 - z0u
          zu  = z0u - 0.5d0 * pun(ji,jj,jk) * zdt

          zzwx = mydomain(ji+1,jj,jk) + zind(ji,jj,jk) * (zu * zslpx(ji+1,jj,jk))
          zzwy = mydomain(ji  ,jj,jk) + zind(ji,jj,jk) * (zu * zslpx(ji  ,jj,jk))

          zwx(ji,jj,jk) = pun(ji,jj,jk) * ( zalpha * zzwx + (1.-zalpha) * zzwy )

          z0v = SIGN( 0.5d0, pvn(ji,jj,jk) )
          zalpha = 0.5d0 - z0v
          zv  = z0v - 0.5d0 * pvn(ji,jj,jk) * zdt
```

PSyclone Schedule View

```
        Loop[type='lon',field_space='None',it_space='None']
            KernCall[]
    NemoCodeBlock[<type 'instance'>]
    Loop[type='levels',field_space='None',it_space='None']
        Loop[type='lat',field_space='None',it_space='None']
            Loop[type='lon',field_space='None',it_space='None']
                KernCall[]
    Loop[type='levels',field_space='None',it_space='None']
        Loop[type='lat',field_space='None',it_space='None']
            Loop[type='lon',field_space='None',it_space='None']
                KernCall[]
    Loop[type='levels',field_space='None',it_space='None']
        NemoCodeBlock[<type 'instance'>]
        Loop[type='lat',field_space='None',it_space='None']
            Loop[type='lon',field_space='None',it_space='None']
                KernCall[]
    Loop[type='levels',field_space='None',it_space='None']
        Loop[type='lat',field_space='None',it_space='None']
            Loop[type='lon',field_space='None',it_space='None']
                KernCall[]
    NemoCodeBlock[<type 'instance'>]
    Loop[type='levels',field_space='None',it_space='None']
        KernCall[]
    NemoCodeBlock[<type 'instance'>]
    Loop[type='levels',field_space='None',it_space='None']
        Loop[type='lat',field_space='None',it_space='None']
            Loop[type='lon',field_space='None',it_space='None']
                KernCall[]
```

Hartree Centre
Science & Technology Facilities Council

Transform the Schedule

```
# Use the resulting DOM to create our psy object
psy = PSyFactory("nemo0.1").create(dom)

sched = psy.invokes.get('invoke_0').schedule

sched.view()

# Find all loops in the Schedule
loops = sched.walk(sched.children, NemoLoop)

# Apply a transformation to every loop over levels
for loop in loops:
    if loop.loop_type == 'levels':
        omptrans.apply(loop)

sched.view()
```

Hartree Centre
Science & Technology Facilities Council

Transformed Schedule

```
            Loop[type='lon',field_space='None',it_space='None']
                KernCall[]
Directive[OMP parallel do]
    Loop[type='levels',field_space='None',it_space='None']
        Loop[type='lat',field_space='None',it_space='None']
            Loop[type='lon',field_space='None',it_space='None']
                KernCall[]
Directive[OMP parallel do]
    Loop[type='levels',field_space='None',it_space='None']
        NemoCodeBlock[<type 'instance'>]
        Loop[type='lat',field_space='None',it_space='None']
            Loop[type='lon',field_space='None',it_space='None']
                KernCall[]
Directive[OMP parallel do]
    Loop[type='levels',field_space='None',it_space='None']
        Loop[type='lat',field_space='None',it_space='None']
            Loop[type='lon',field_space='None',it_space='None']
                KernCall[]
NemoCodeBlock[<type 'instance'>]
Directive[OMP parallel do]
    Loop[type='levels',field_space='None',it_space='None']
        KernCall[]
NemoCodeBlock[<type 'instance'>]
Directive[OMP parallel do]
    Loop[type='levels',field_space='None',it_space='None']
        Loop[type='lat',field_space='None',it_space='None']
            Loop[type='lon',field_space='None',it_space='None']
                KernCall[]
Directive[OMP parallel do]
    Loop[type='levels',field_space='None',it_space='None']
        Loop[type='lat',field_space='None',it_space='None']
            Loop[type='lon',field_space='None',it_space='None']
```

Transformed Fortran

```
!$omp end parallel do
!$omp parallel do default(shared), private(jk,jj,ji), schedule(static)
  DO jk = 1 , jpk - 1 , 1
   DO jj = 2 , jpj , 1
    DO ji = 2 , jpi , 1
     zslpx ( ji , jj , jk ) = sign ( 1.d0 , zslpx ( ji , jj , jk ) ) * min ( abs ( zslpx ( ji , jj , jk
) ) , 2.d0 * abs ( zwx (&
      ji - 1 , jj , jk ) ) , 2.d0 * abs ( zwx ( ji , jj , jk ) ) )
     zslpy ( ji , jj , jk ) = sign ( 1.d0 , zslpy ( ji , jj , jk ) ) * min ( abs ( zslpy ( ji , jj , jk
) ) , 2.d0 * abs ( zwy (&
      ji , jj - 1 , jk ) ) , 2.d0 * abs ( zwy ( ji , jj , jk ) ) )
    END DO
   END DO
  END DO
!$omp end parallel do
!$omp parallel do default(shared), private(jk,jj,ji,z0u,z0v,zalpha,zu,zv,zzwy,zzwx), schedule(static)
  DO jk = 1 , jpk - 1 , 1
   zdt = 1
   DO jj = 2 , jpj - 1 , 1
    DO ji = 2 , jpi - 1 , 1
     z0u = sign ( 0.5d0 , pun ( ji , jj , jk ) )
     zalpha = 0.5d0 - z0u
     zu = z0u - 0.5d0 * pun ( ji , jj , jk ) * zdt
     zzwx = mydomain ( ji + 1 , jj , jk ) + zind ( ji , jj , jk ) * ( zu * zslpx ( ji + 1 , jj , jk ) )
     zzwy = mydomain ( ji , jj , jk ) + zind ( ji , jj , jk ) * ( zu * zslpx ( ji , jj , jk ) )
     zwx ( ji , jj , jk ) = pun ( ji , jj , jk ) * ( zalpha * zzwx + ( 1. - zalpha ) * zzwy )
     z0v = sign ( 0.5d0 , pvn ( ji , jj , jk ) )
     zalpha = 0.5d0 - z0v
     zv = z0v - 0.5d0 * pvn ( ji , jj , jk ) * zdt
     zzwx = mydomain ( ji , jj + 1 , jk ) + zind ( ji , jj , jk ) * ( zv * zslpy ( ji , jj + 1 , jk ) )
     zzwy = mydomain ( ji , jj , jk ) + zind ( ji , jj , jk ) * ( zv * zslpy ( ji , jj , jk ) )
     zwy ( ji , jj , jk ) = pvn ( ji , jj , jk ) * ( zalpha * zzwx + ( 1.d0 - zalpha ) * zzwy )
    END DO
   END DO
  END DO
```

(an end-user wouldn't normally see this)

```
!$omp end parallel do
!$omp parallel do default(shared), private(jk,jj,ji), schedule(static)
  DO jk = 1 , jpk - 1 , 1
   DO jj = 2 , jpj , 1
    DO ji = 2 , jpi , 1
     zslpx ( ji , jj , jk ) = sign ( 1.d0 , zslpx ( ji , jj , jk ) ) * min ( abs ( zslpx ( ji , jj , jk
) ) , 2.d0 * abs ( zwx (&
      ji - 1 , jj , jk ) ) , 2.d0 * abs ( zwx ( ji , jj , jk ) ) )
     zslpy ( ji , jj , jk ) = sign ( 1.d0 , zslpy ( ji , jj , jk ) ) * min ( abs ( zslpy ( ji , jj , jk
) ) , 2.d0 * abs ( zwy (&
      ji , jj - 1 , jk ) ) , 2.d0 * abs ( zwy ( ji , jj , jk ) ) )
    END DO
   END DO
  END DO
!$omp end parallel do
!$omp parallel do defa                                                zwx), schedule(static)
  DO jk = 1 , jpk - 1
   zdt = 1
   DO jj = 2 , jpj - 1
    DO ji = 2 , jpi - 1 , 1
     z0u = sign ( 0.5d0 , pun ( ji , jj , jk ) )
     zalpha = 0.5d0 - z0u
     zu = z0u - 0.5d0 * pun ( ji , jj , jk ) * zdt
     zzwx = mydomain ( ji + 1 , jj , jk ) + zind ( ji , jj , jk ) * ( zu * zslpx ( ji + 1 , jj , jk ) )
     zzwy = mydomain ( ji , jj , jk ) + zind ( ji , jj , jk ) * ( zu * zslpx ( ji , jj , jk ) )
     zwx ( ji , jj , jk ) = pun ( ji , jj , jk ) * ( zalpha * zzwx + ( 1. - zalpha ) * zzwy )
     z0v = sign ( 0.5d0 , pvn ( ji , jj , jk ) )
     zalpha = 0.5d0 - z0v
     zv = z0v - 0.5d0 * pvn ( ji , jj , jk ) * zdt
     zzwx = mydomain ( ji , jj + 1 , jk ) + zind ( ji , jj , jk ) * ( zv * zslpy ( ji , jj + 1 , jk ) )
     zzwy = mydomain ( ji , jj , jk ) + zind ( ji , jj , jk ) * ( zv * zslpy ( ji , jj , jk ) )
     zwy ( ji , jj , jk ) = pvn ( ji , jj , jk ) * ( zalpha * zzwx + ( 1.d0 - zalpha ) * zzwy )
    END DO
   END DO
  END DO
```

(but they can if they want)

```
!$omp end parallel do
!$omp parallel do default(shared), private(jk,jj,ji), schedule(static)
  DO jk = 1 , jpk - 1 , 1
   DO jj = 2 , jpj , 1
    DO ji = 2 , jpi , 1
     zslpx ( ji , jj , jk ) = sign ( 1.d0 , zslpx ( ji , jj , jk ) ) * min ( abs ( zslpx ( ji , jj , jk
) ) , 2.d0 * abs ( zwx (&
      ji - 1 , jj , jk ) ) ,
     zslpy ( ji , jj , jk ) =                                                  ( abs ( zslpy ( ji , jj , jk
) ) , 2.d0 * abs ( zwy (&
      ji , jj - 1 , jk ) ) ,
    END DO
   END DO
  END DO
!$omp end parallel do
!$omp parallel do default(sha                                                  zwy,zzwx), schedule(static)
  DO jk = 1 , jpk - 1 , 1
   zdt = 1
   DO jj = 2 , jpj - 1 , 1
    DO ji = 2 , jpi - 1 , 1
     z0u = sign ( 0.5d0 , pun
     zalpha = 0.5d0 - z0u
     zu = z0u - 0.5d0 * pun (
     zzwx = mydomain ( ji + 1                                                  zslpx ( ji + 1 , jj , jk ) )
     zzwy = mydomain ( ji , j                                                  px ( ji , jj , jk ) )
     zwx ( ji , jj , jk ) = p                                                  - zalpha ) * zzwy )
     z0v = sign ( 0.5d0 , pvn
     zalpha = 0.5d0 - z0v
     zv = z0v - 0.5d0 * pvn ( ji , jj , jk ) * zdt
     zzwx = mydomain ( ji , jj + 1 , jk ) + zind ( ji , jj , jk ) * ( zv * zslpy ( ji , jj + 1 , jk ) )
     zzwy = mydomain ( ji , jj , jk ) + zind ( ji , jj , jk ) * ( zv * zslpy ( ji , jj , jk ) )
     zwy ( ji , jj , jk ) = pvn ( ji , jj , jk ) * ( zalpha * zzwx + ( 1.d0 - zalpha ) * zzwy )
    END DO
   END DO
  END DO
```

Outlook

- PSyclone remains a key part of the UK Met Office's LFRic project
  - Development effort being provided by STFC, UK Met Office + others (not directly funded)
- OpenACC support for PSyclone is currently being implemented (GOcean API as first target)
  - Mostly working for NEMOLite2D
- PSyclone being extended to generate OpenCL for NEMOLite2D in the EuroEXA project
  - Will work on full NEMO support if time allows