Towards Performance Portability in Earth-System Modelling with GOcean.

Andrew Porter, Rupert Ford & Mike Ashworth
(*STFC Daresbury Laboratory*)
Graham Riley
(*University of Manchester*)
Manish Modani
(*STFC Hartree Centre*)

Science & Technology Facilities Council

Contents

- Motivation (the GungHo Project)
- The GOcean Project
- Separation of Concerns (PSyKAl)
- Application of PSyKAl to shallow-water models
- Performance Impact of PSyKAl
  - Serial
  - OpenMP
  - GPU
- Summary

Science & Technology Facilities Council

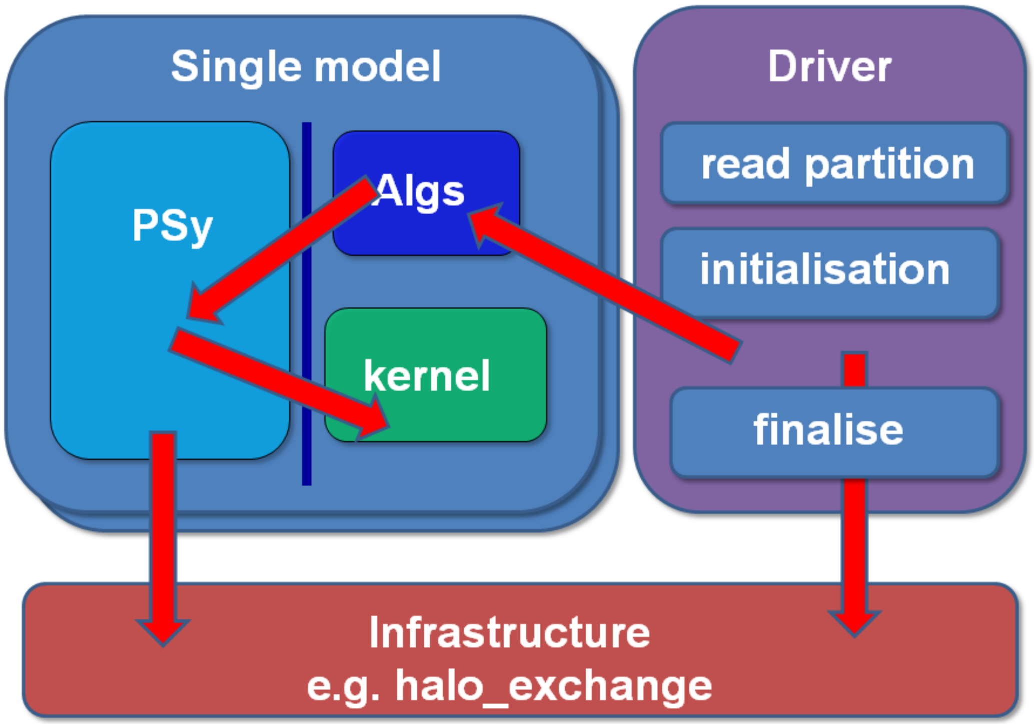
GungHo – What and Why?

- GungHo is a UK Met Office/NERC project aiming to research, design and develop a new dynamical core suitable for operational, global and regional, weather and climate simulation
- Computer architectures are in a state of flux with a variety of competing technologies
  - GungHo is developing code for a computer that does not yet exist
  - Many cores, accelerators (PCIe or socket), FPGAs…
- How can we produce maintainable, scientifically-correct code that will perform well on a range of future architectures?

工合海洋 GOcean

- NERC funded, 03/2014 – 02/2015
- Collaboration between NOC and STFC
- Investigate the feasibility of applying technology from the GungHo project to ocean modelling
- Extend the developing GungHo infrastructure to support finite difference on regular, lat-lon grids

Separation of Concerns in GungHo



Science & Technology Facilities Council

The Parallel System, Kernel, Algorithm (PSyKAl) Approach…

- Oceanographer writes the algorithm (top) and kernel (bottom) layers, following certain rules
  - no need to worry about relative indexing of various fields
  - no need to worry about parallelism (algorithm layer deals with **logically global field quantities**)
- A code-generation system (PSyclone) generates the PSy middle layer
  - glues the algorithm and kernels together
  - incorporates **all code related to parallelism**

Science & Technology Facilities Council

Two shallow-water codes…

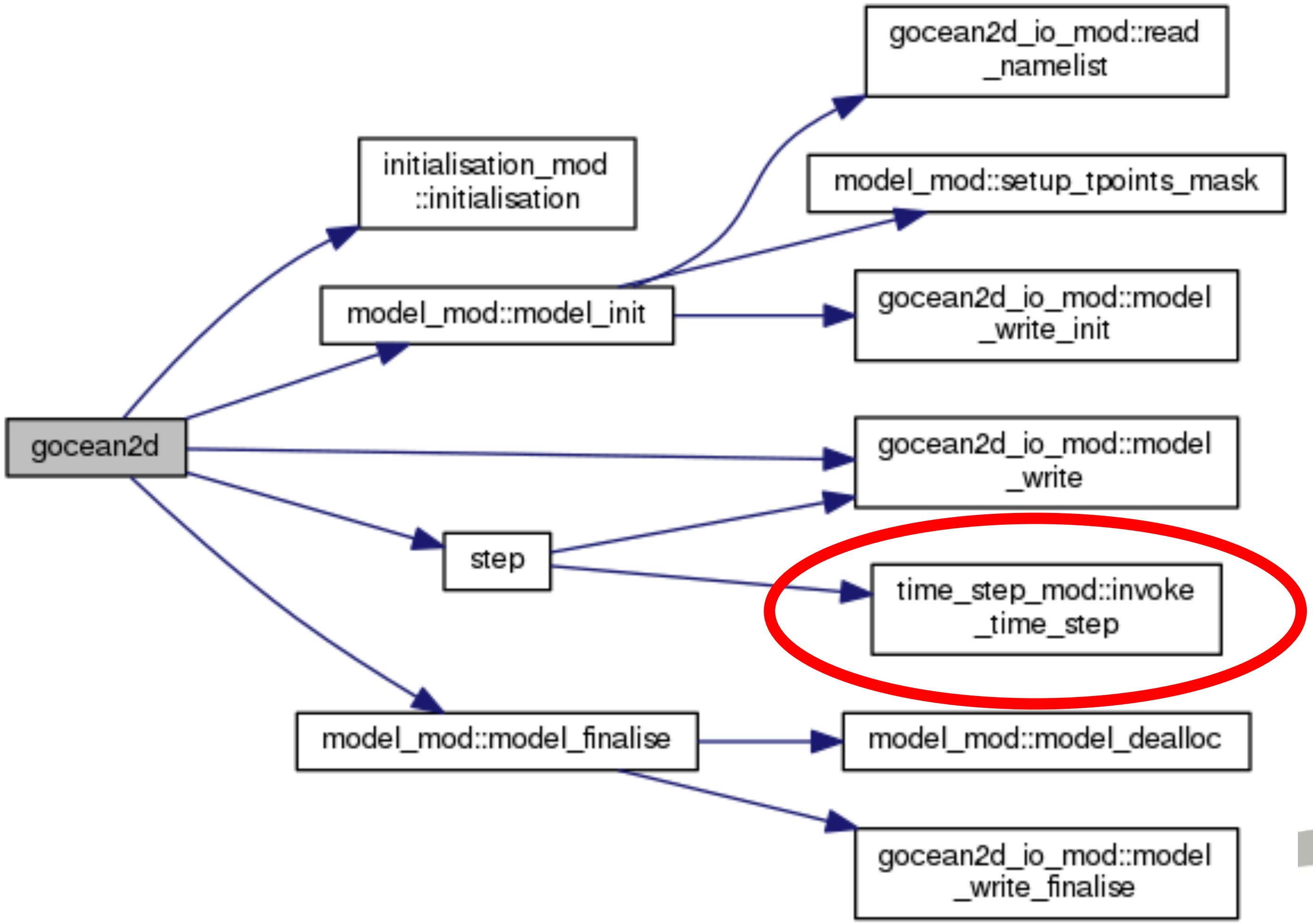
- We have applied the PSyKAl approach to two codes:
  - ‘Shallow’ originally written by Swarztrauber, NCAR
  - ‘NEMOLite2D’ 2D, free-surface part of NEMO extracted by NOC
- Both use Finite Difference on Arakawa C grid
- But there are important differences:
  - Boundary conditions (bi-periodic vs. forced/closed)
  - Relative indexing of variables on the grid
- Understanding and expressing these differences is essential for correct code generation

Two benchmarks…

- Have re-structured both Shallow and NEMOLite2D following PSyKAl separation of concerns
- Manually optimise Shallow while obeying PSyKAl rules
  - Serial benchmark
- Manually optimise NEMOLite2D and implement OpenMP parallelisation
  - Parallel benchmark
- These benchmarks have also been supplied to vendors (IBM, NVIDIA) for them to optimise for their hardware while obeying PSyKAl rules

Science & Technology Facilities Council

NEMOLite2D - structure



Science & Technology Facilities Council

Body of time-stepping loop consists of kernel calls:

```
call invoke(                                                   &
        continuity(ssha_t, sshn_t, sshn_u, sshn_v,             &
                   hu, hv, un, vn, rdt),                       &

        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)                              &
       )
```

A kernel looks like:

```
subroutine continuity_code(ji, jj,                         &
                           ssha, sshn, sshn_u, sshn_v, &
                           hu, hv, un, vn, rdt, e12t)
  implicit none
  integer,                  intent(in)  :: ji, jj
  real(wp),                 intent(in)  :: rdt
  real(wp), dimension(:,:), intent(in)  :: e12t
  real(wp), dimension(:,:), intent(out) :: ssha
  real(wp), dimension(:,:), intent(in)  :: sshn, sshn_u, sshn_v, &
                                           hu, hv, un, vn
  ! Locals
  real(wp) :: rtmp1, rtmp2, rtmp3, rtmp4

  rtmp1 = (sshn_u(ji  ,jj  ) + hu(ji  ,jj  )) * un(ji  ,jj  )
  rtmp2 = (sshn_u(ji-1,jj  ) + hu(ji-1,jj  )) * un(ji-1,jj  )
  rtmp3 = (sshn_v(ji  ,jj  ) + hv(ji  ,jj  )) * vn(ji  ,jj  )
  rtmp4 = (sshn_v(ji  ,jj-1) + hv(ji  ,jj-1)) * vn(ji  ,jj-1)

  ssha(ji,jj) = sshn(ji,jj) + (rtmp2 - rtmp1 + rtmp4 - rtmp3) * &
                rdt / e12t(ji,jj)

end subroutine continuity_code
```

A kernel looks like:

A kernel operates on a single grid-point

```
subroutine continuity_code(ji, jj,                     &
                           ssha, sshn, sshn_u, sshn_v, &
                           hu, hv, un, vn, rdt, e12t)
  implicit none
  integer,                     intent(in)  :: ji, jj
  real(wp),                    intent(in)  :: rdt
  real(wp), dimension(:,:), intent(in)  :: e12t
  real(wp), dimension(:,:), intent(out) :: ssha
  real(wp), dimension(:,:), intent(in)  :: sshn, sshn_u, sshn_v, &
                                           hu, hv, un, vn
  ! Locals
  real(wp) :: rtmp1, rtmp2, rtmp3, rtmp4

  rtmp1 = (sshn_u(ji  ,jj  ) + hu(ji  ,jj  )) * un(ji  ,jj  )
  rtmp2 = (sshn_u(ji-1,jj  ) + hu(ji-1,jj  )) * un(ji-1,jj  )
  rtmp3 = (sshn_v(ji  ,jj  ) + hv(ji  ,jj  )) * vn(ji  ,jj  )
  rtmp4 = (sshn_v(ji  ,jj-1) + hv(ji  ,jj-1)) * vn(ji  ,jj-1)

  ssha(ji,jj) = sshn(ji,jj) + (rtmp2 - rtmp1 + rtmp4 - rtmp3) * &
                rdt / e12t(ji,jj)

end subroutine continuity_code
```

Science & Technology Facilities Council

Kernel meta-data

- Kernels make use of several grid-related quantities, *e.g*. area of cell around a T point, T-point mask etc.

```
subroutine next_sshu_code(ji,jj, sshn_u, sshn, &
                          tmask,e12t,e12u)
  implicit none
  integer,                   intent(in)    :: ji, jj
  integer,  dimension(:,:), intent(in)    :: tmask
  real(wp), dimension(:,:), intent(in)    :: e12t, e12u
  real(wp), dimension(:,:), intent(inout) :: sshn_u
  real(wp), dimension(:,:), intent(in)    :: sshn
```

- The algorithm layer should not/cannot supply these:

```
call invoke( next_sshu(sshn_u, sshn), &
             next_sshv(sshn_v, sshn) )
```

- Extend meta-data to specify what quantities a kernel requires from the infrastructure:

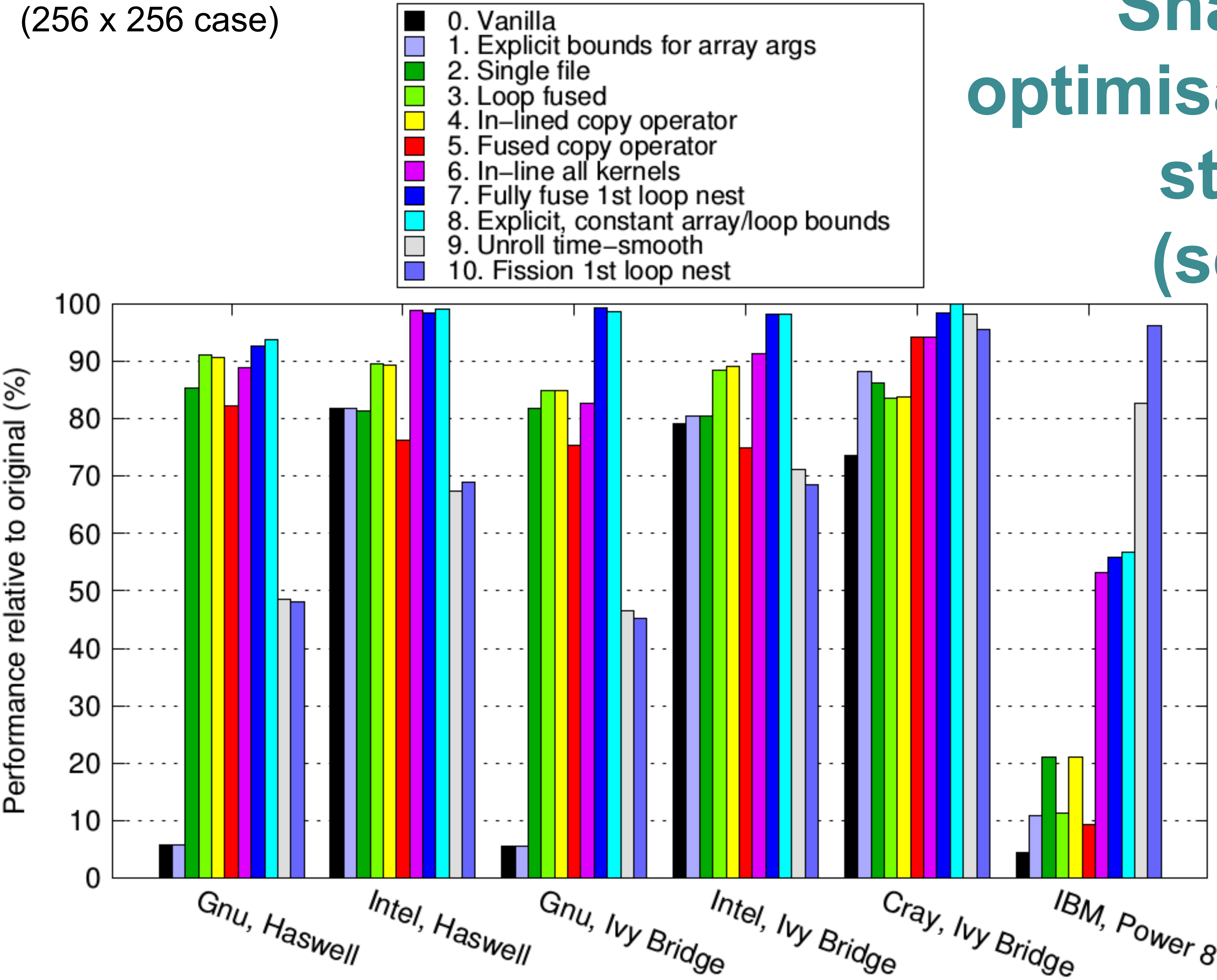
```
type, extends(kernel_type) :: next_sshu
   type(arg), dimension(5) :: meta_args =  &
        (/ arg(READWRITE, CU, POINTWISE),  &
           arg(READ,      CU, POINTWISE),  &
           arg(READ,      GRID_MASK_T),    &
           arg(READ,      GRID_AREA_T),    &
           arg(READ,      GRID_AREA_U)     &
         /)
```

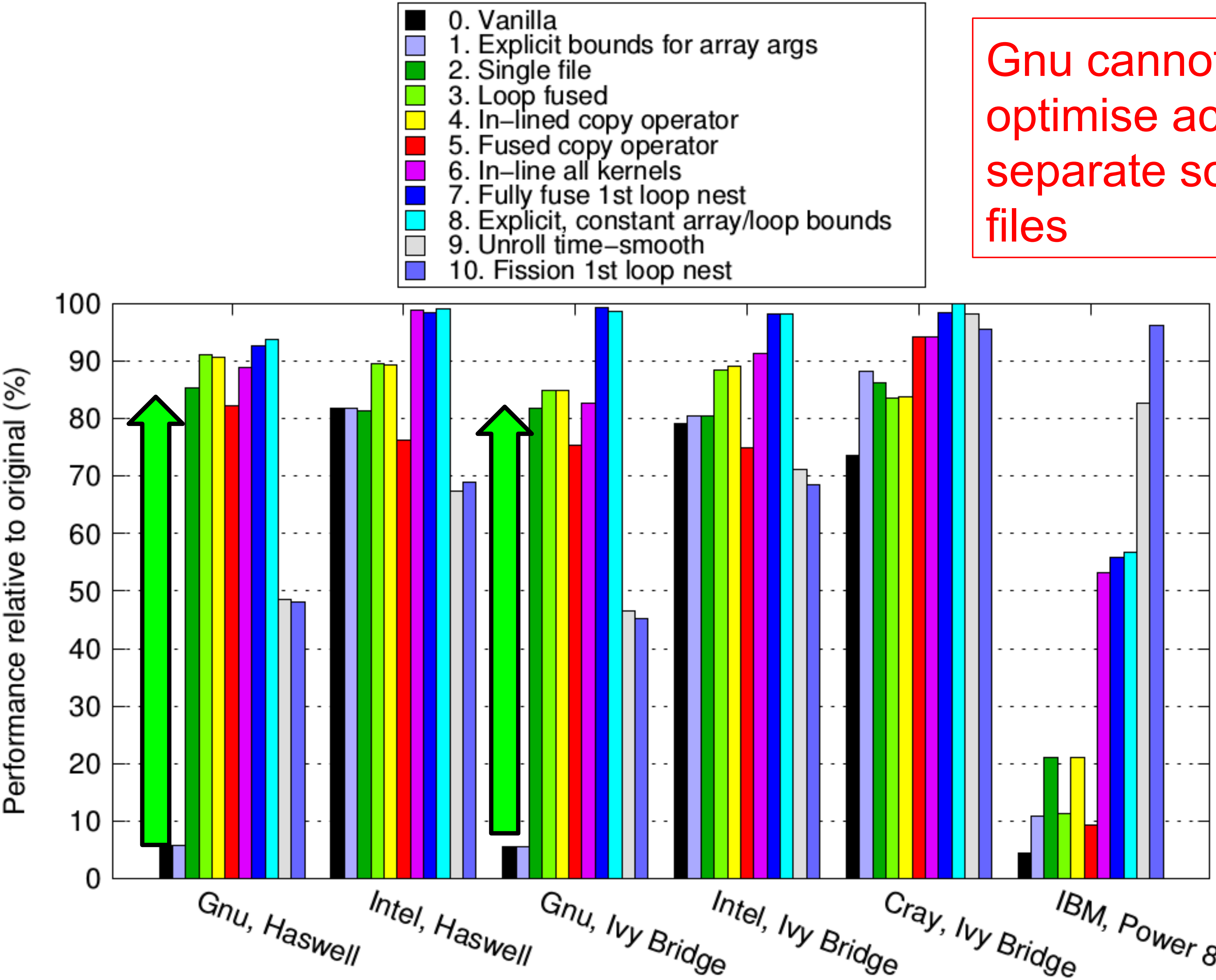
- PSyclone then supplies these quantities from the generated middle layer

What about performance?

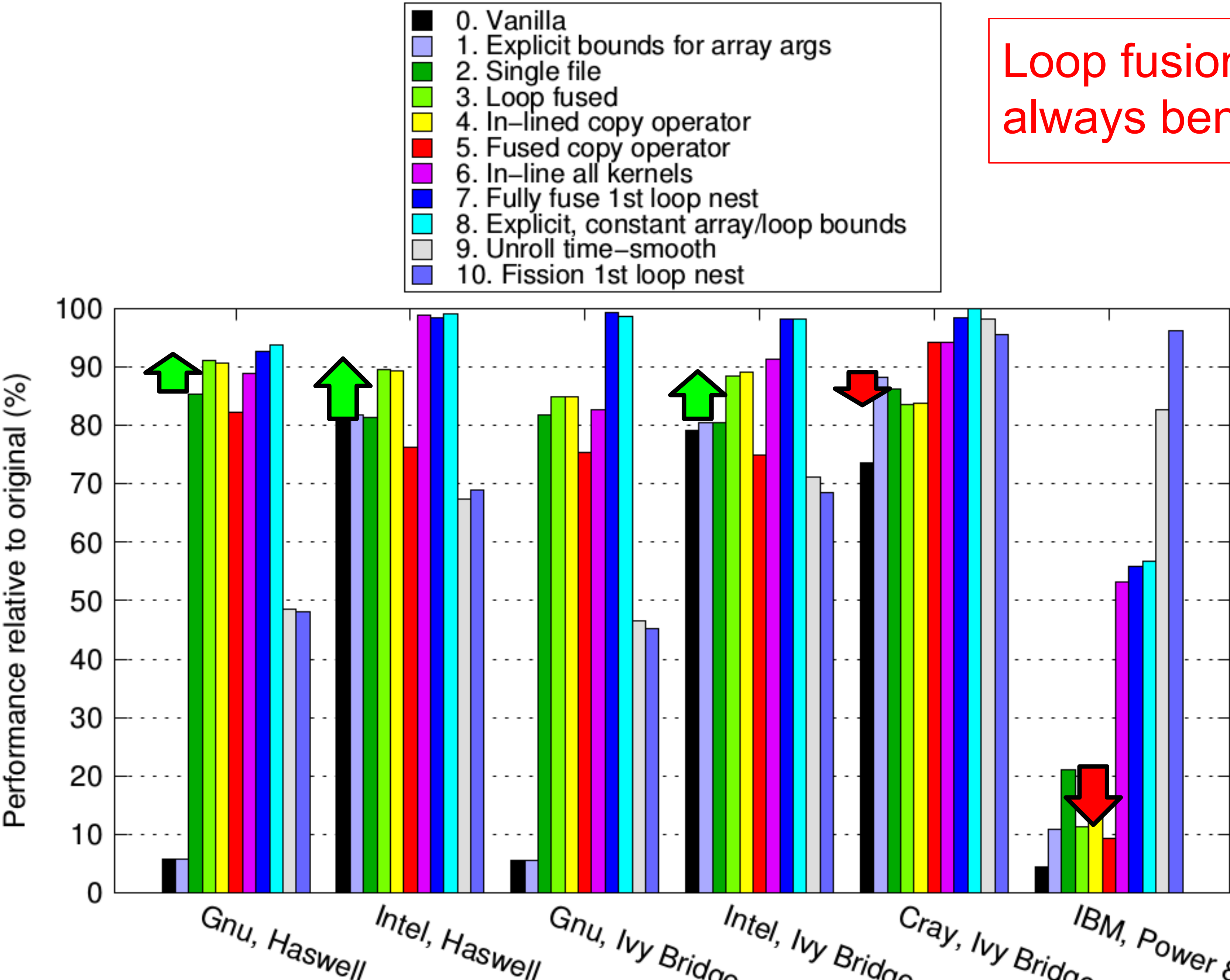
Shallow optimisation stages (serial)

(256 x 256 case)

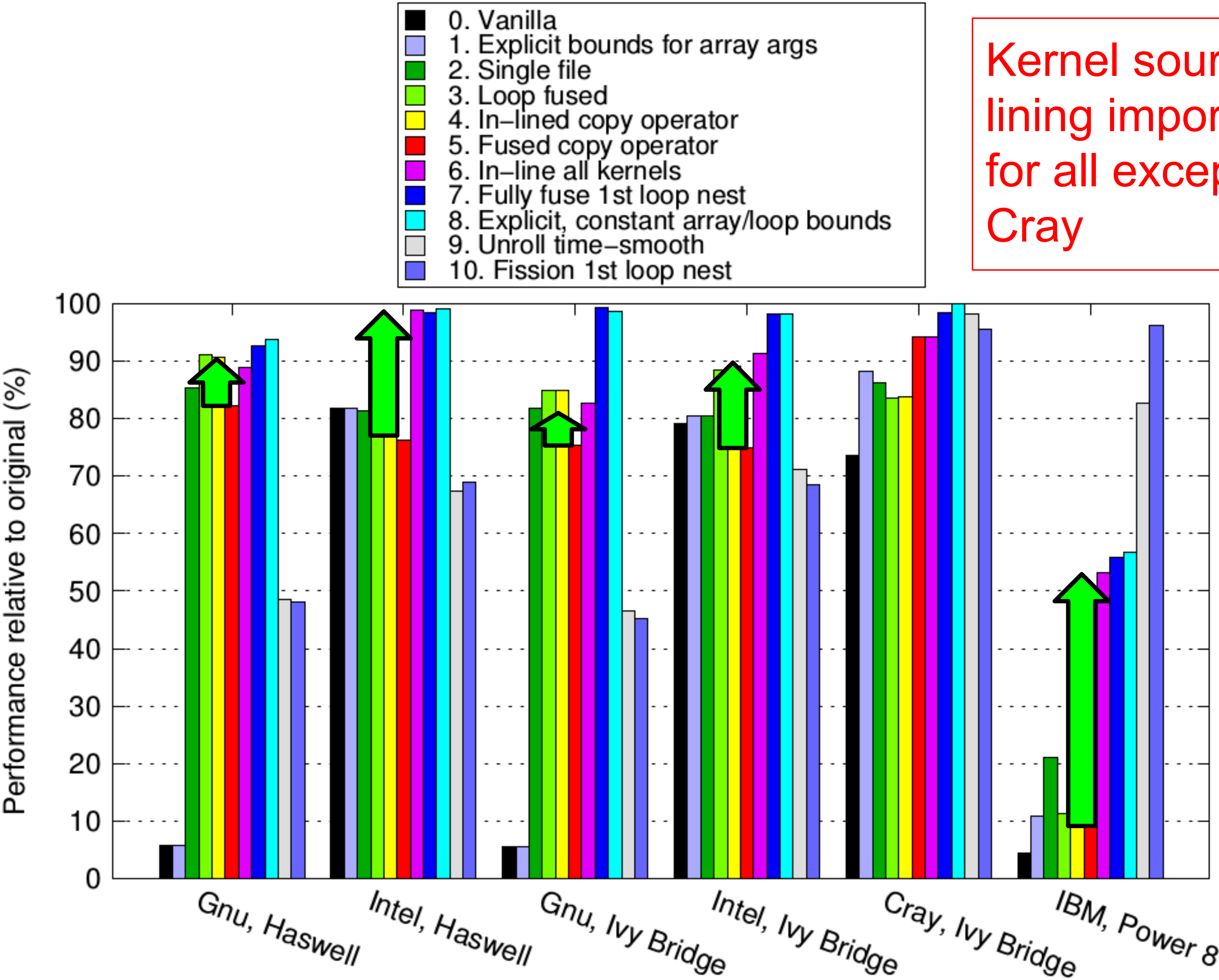




Gnu cannot optimise across separate source files

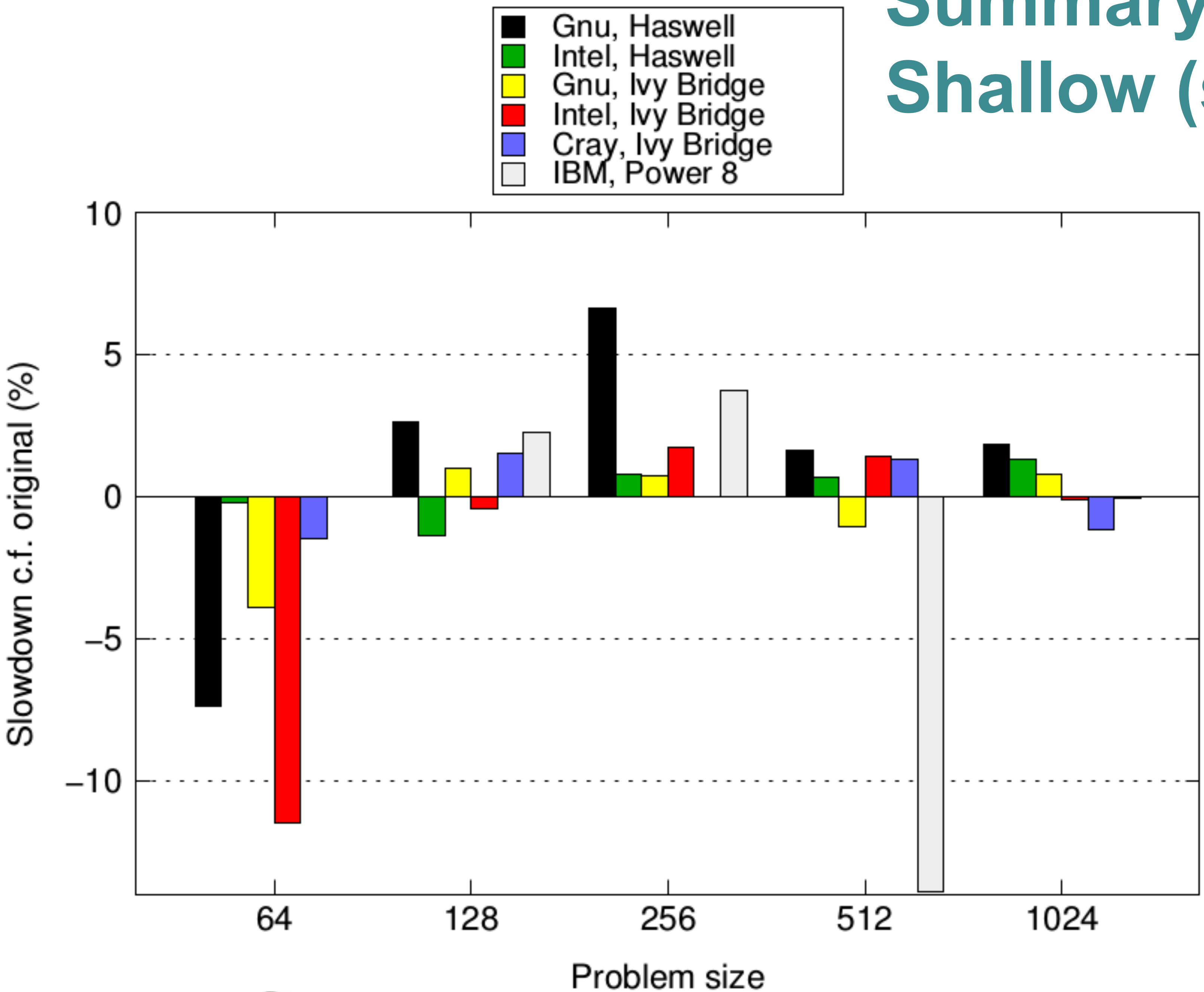


Loop fusion not always beneficial

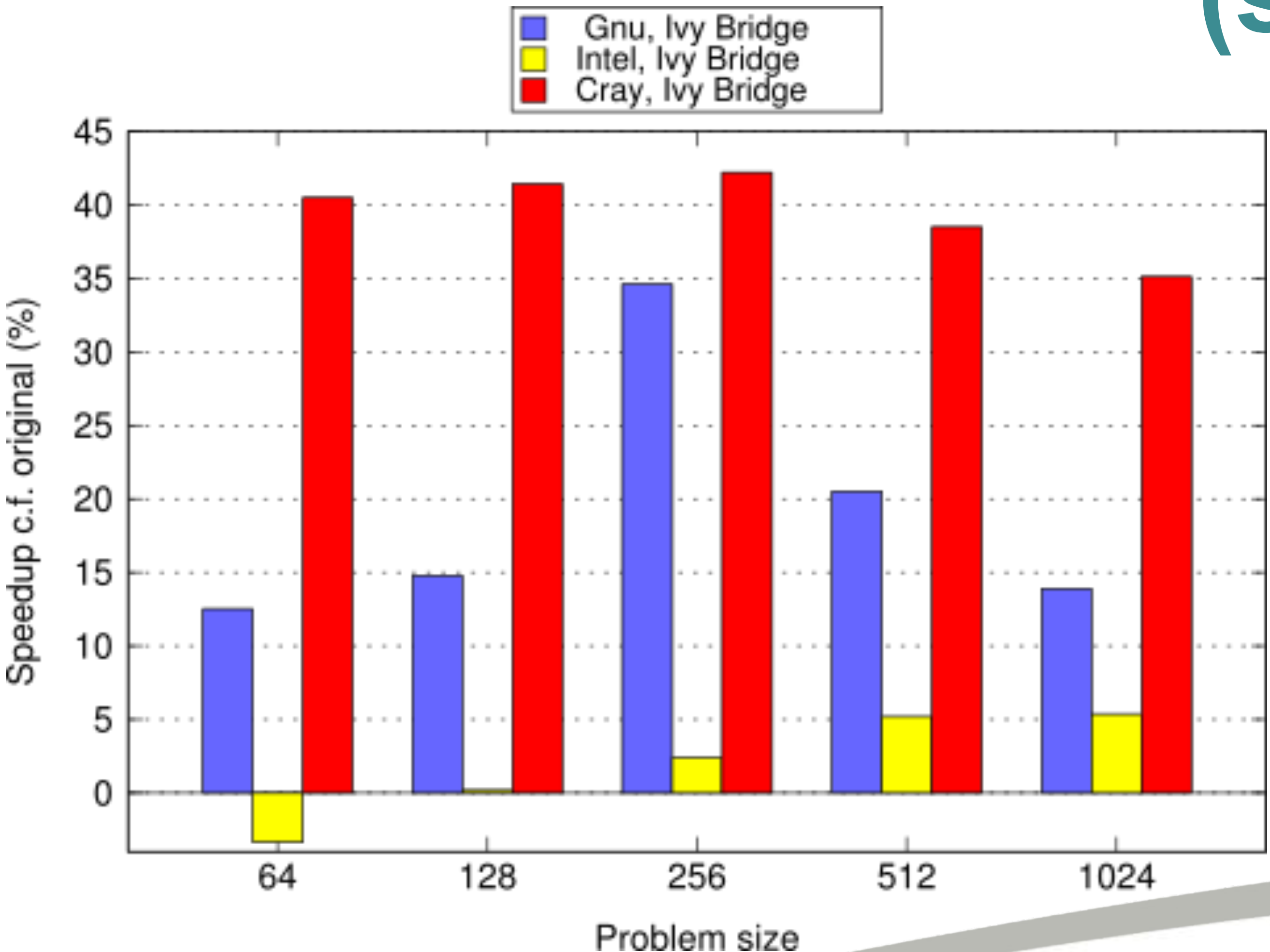


Kernel source in-lining important for all except Cray

Summary for Shallow (serial)



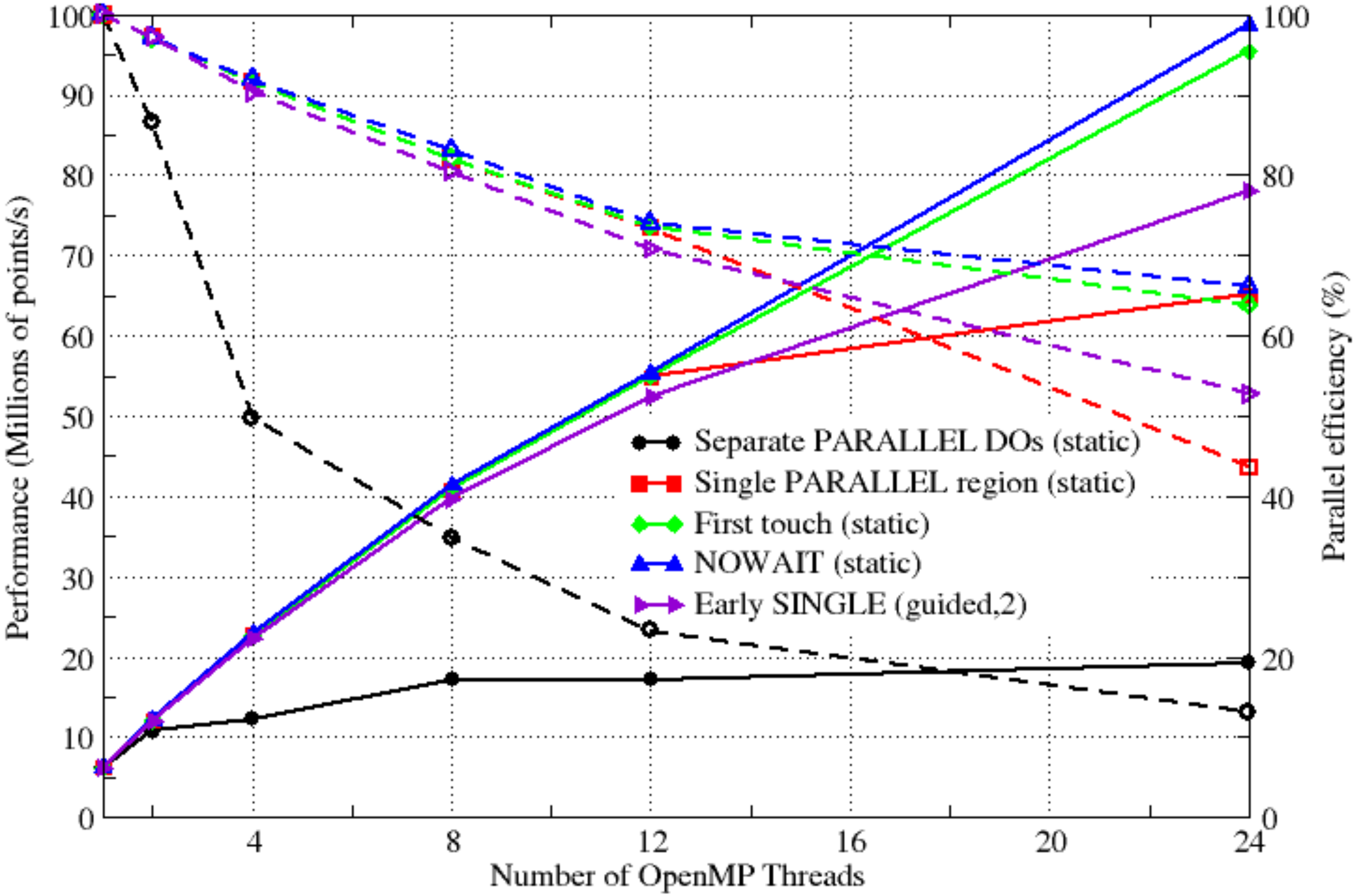
Second benchmark: NEMOLite2D (serial)



Performance with OpenMP

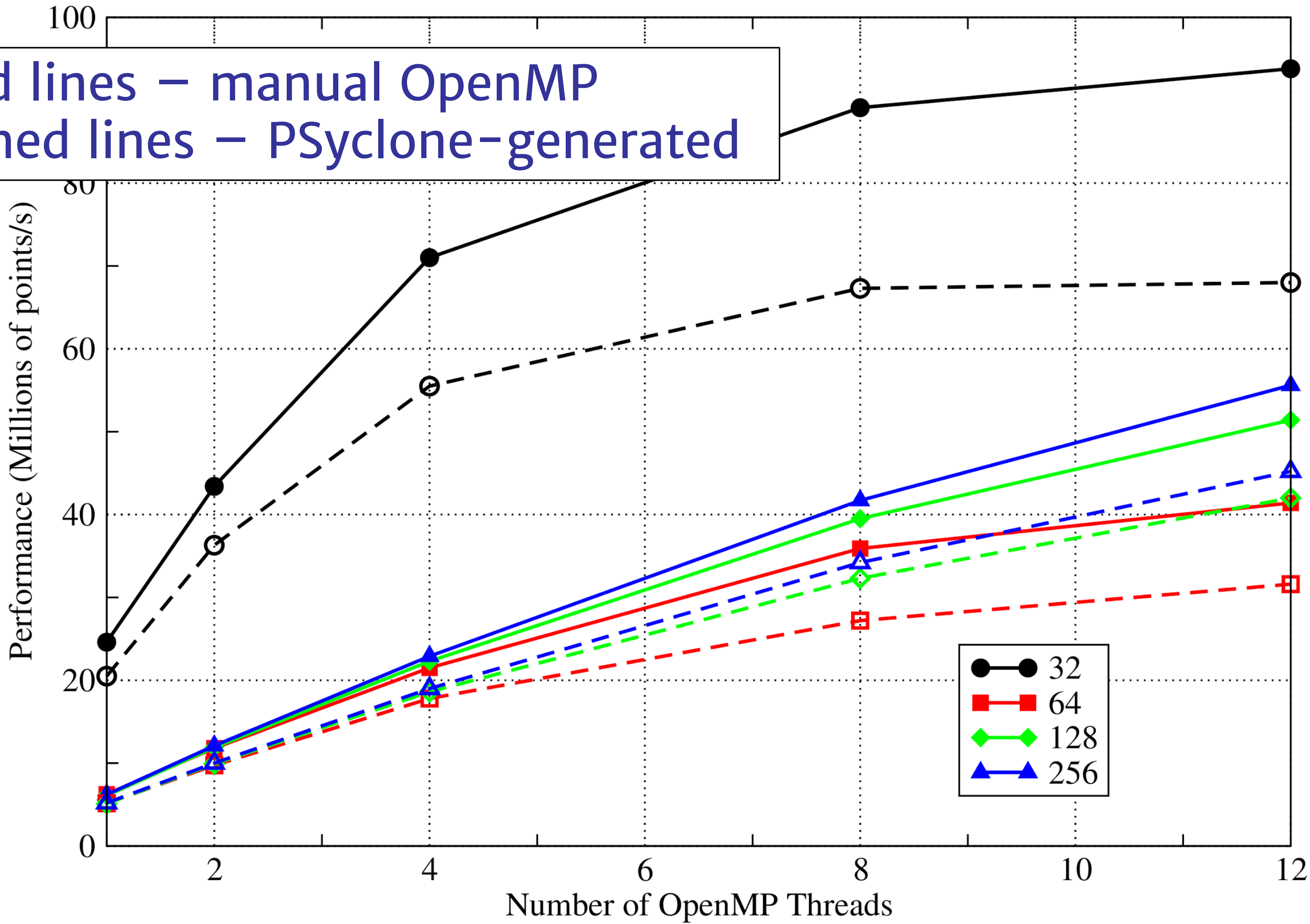
NEMOLite2D, OpenMP optimisation stages

(256 x 256 case, Intel v.14)

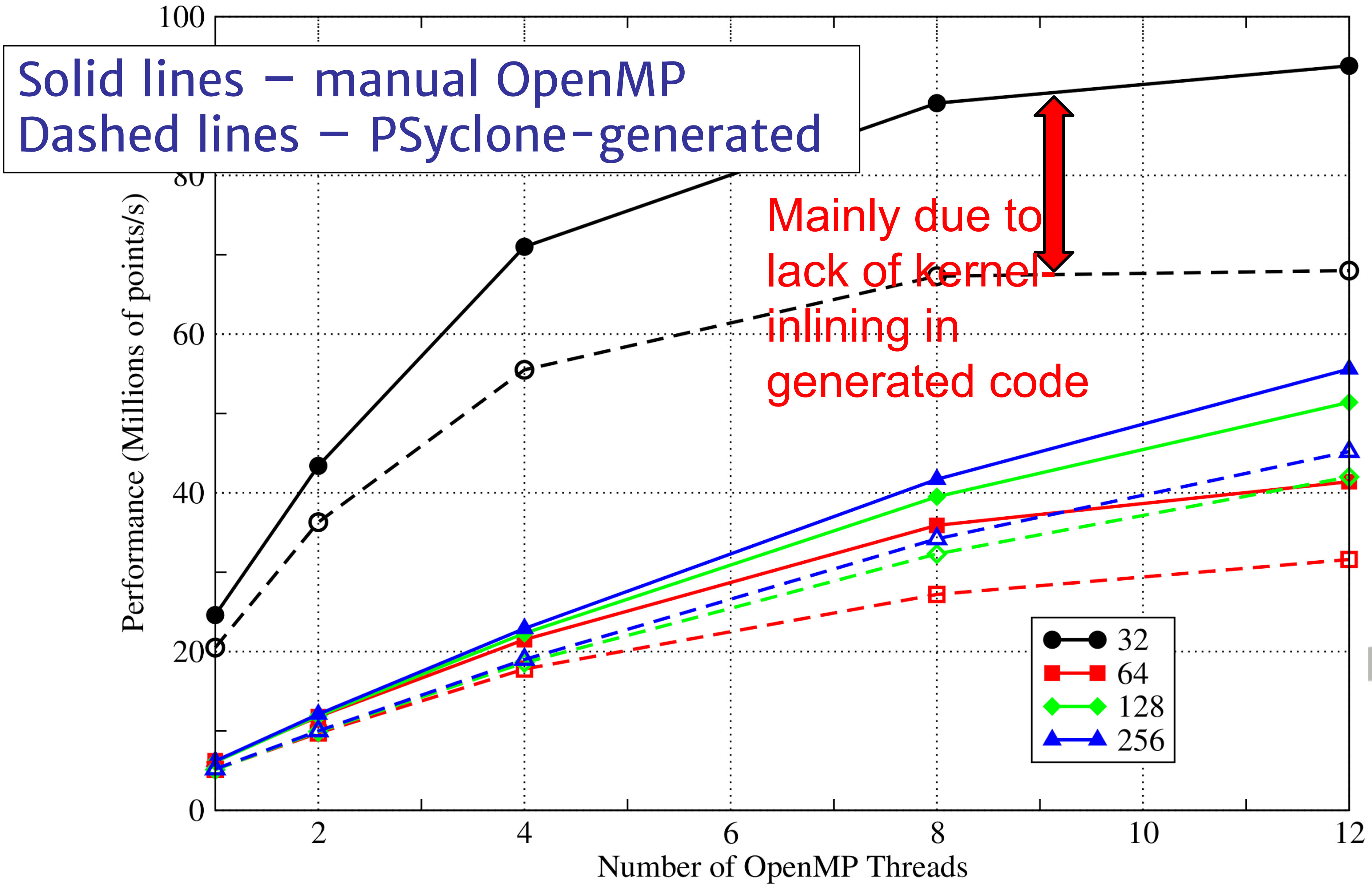


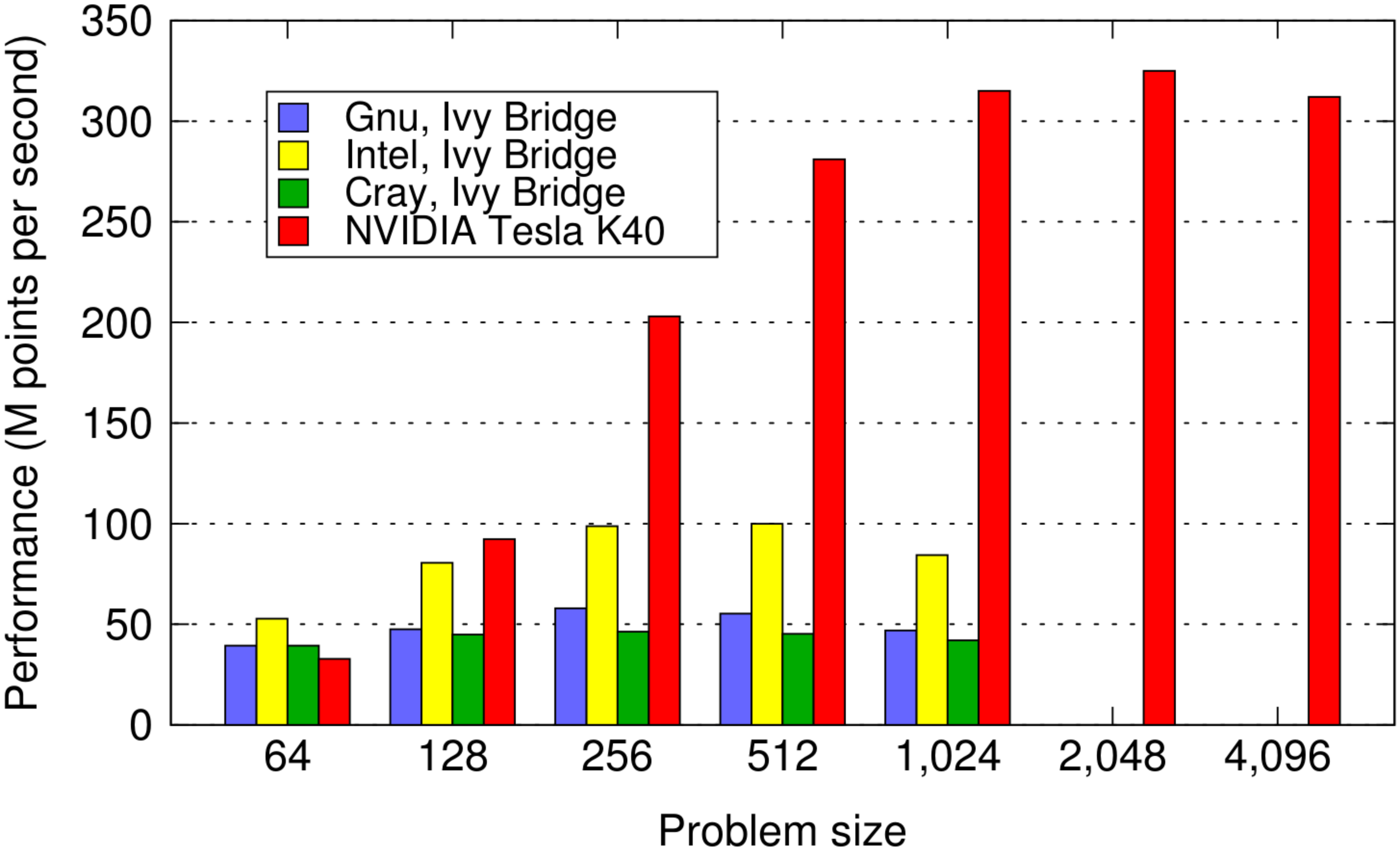
NEMOLite2D, OpenMP

Solid lines – manual OpenMP
Dashed lines – PSyclone-generated



NEMOLite2D, OpenMP





GPU results courtesy of Jeremy Appleyard, NVIDIA

Science & Technology Facilities Council

Next steps…

- Compilers are complex!
  - Recovering performance is not straightforward
  - More transformations required (e.g. in-lining of kernel code for Gnu and Intel)
- Currently not exploring the optimisation space
  - **Only attempting to recover original code structure**
- Three dimensions
  - Current test cases are two-dimensional
  - Full models are a mixture of 2D and 3D…
  - NOC working on introducing some 3D aspects to NEMOLite

Summary I

- Separation of Concerns: Introduces flexibility needed to achieve performance on different architectures
  - potentially enables *e.g*. OpenMP or OpenACC to be used, depending on target hardware
  - No need to modify source code containing the Natural Science (Algorithm and Kernel layers)
- Optimal code structure is both system- and compiler-dependent

Science & Technology Facilities Council

Summary II

- Framework now supports two distinct shallow-water models
- Code generation
  - Support for loop fusion and OpenMP transformations (parallel, parallel do and do)
- Work continuing on PSyclone in the GungHo project
  - More OpenMP support (loop colouring)
  - Distributed memory support (MPI)
  - Support for OpenACC

Science & Technology Facilities Council

Thank you!

Science & Technology Facilities Council

Extras…

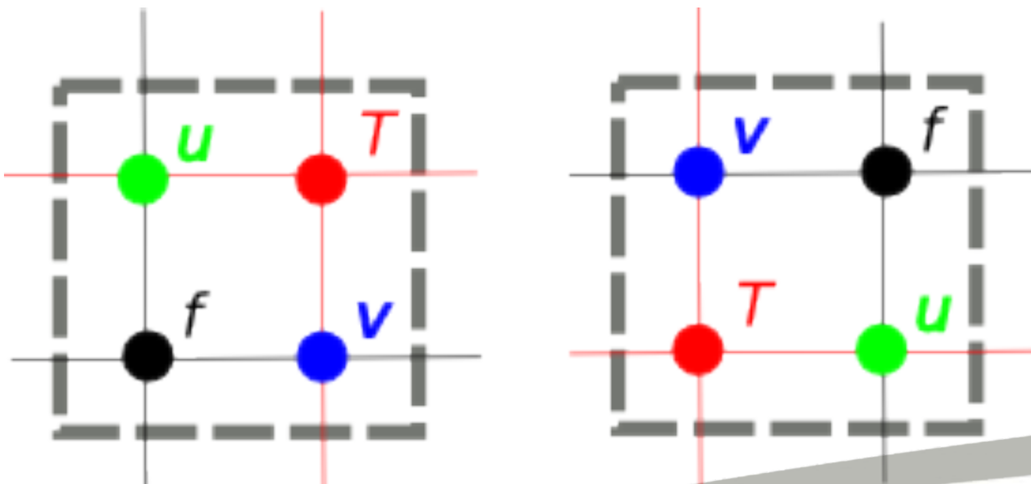
Compilers and CPUs

	Gnu	Intel	Cray	IBM
Intel Haswell (E5-1620 v2)	4.9.3	14.0.0		
Intel Ivy Bridge (E5-2697)	4.9.1	14.0.1.106	8.3.3	
IBM Power 8				15.1.2 (Linux)

Science & Technology Facilities Council

Offset choice

- A *developer* can choose how the different grid-point types are indexed relative to T
- shallow defines {u,v,f} points to the South and West of the T point to have same (i,j) index while NEMO uses those to the North and East:



- We call this choice the 'offset' of the grids
- Specified in kernel meta-data

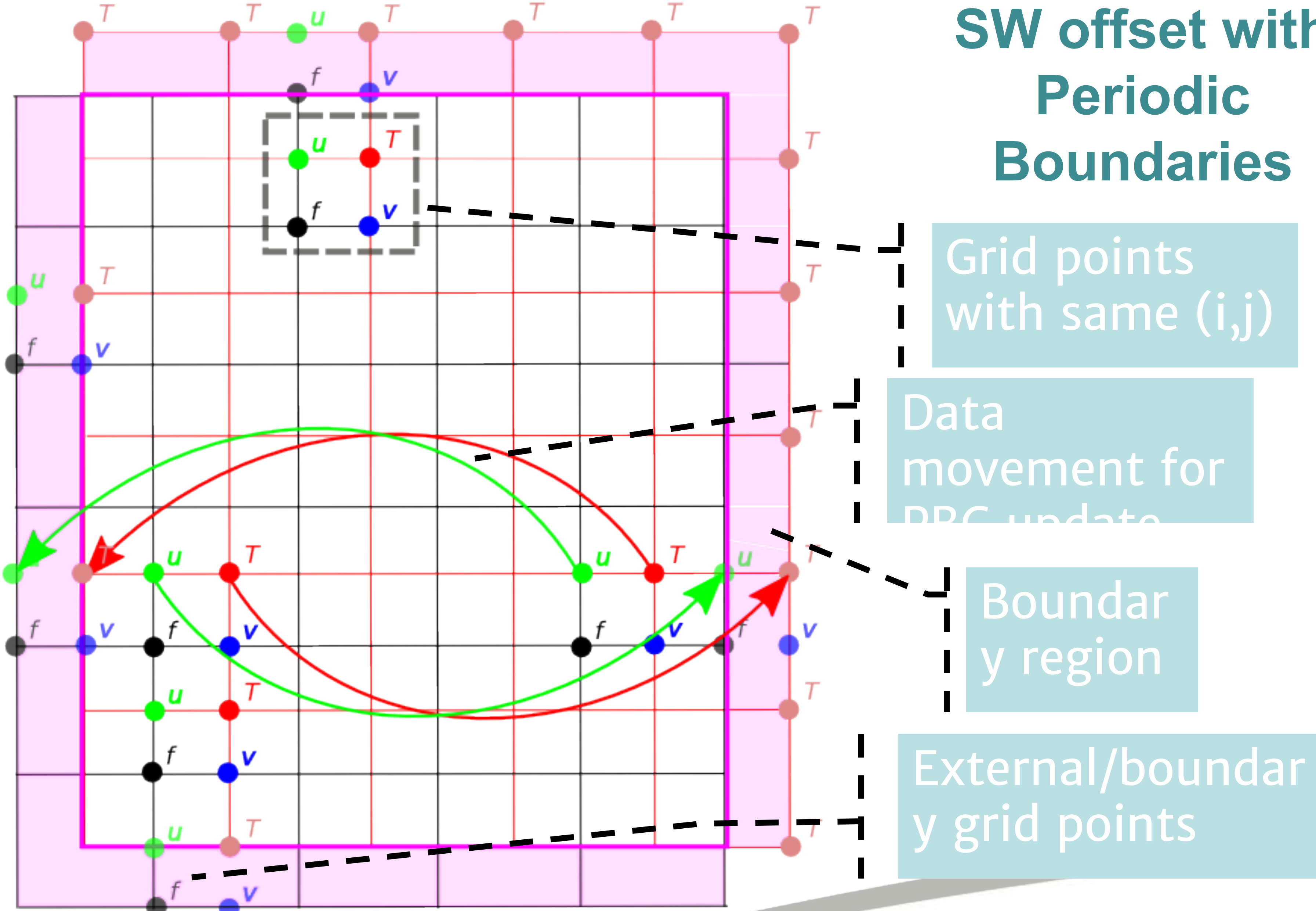
Science & Technology Facilities Council

Middle layer is generated…

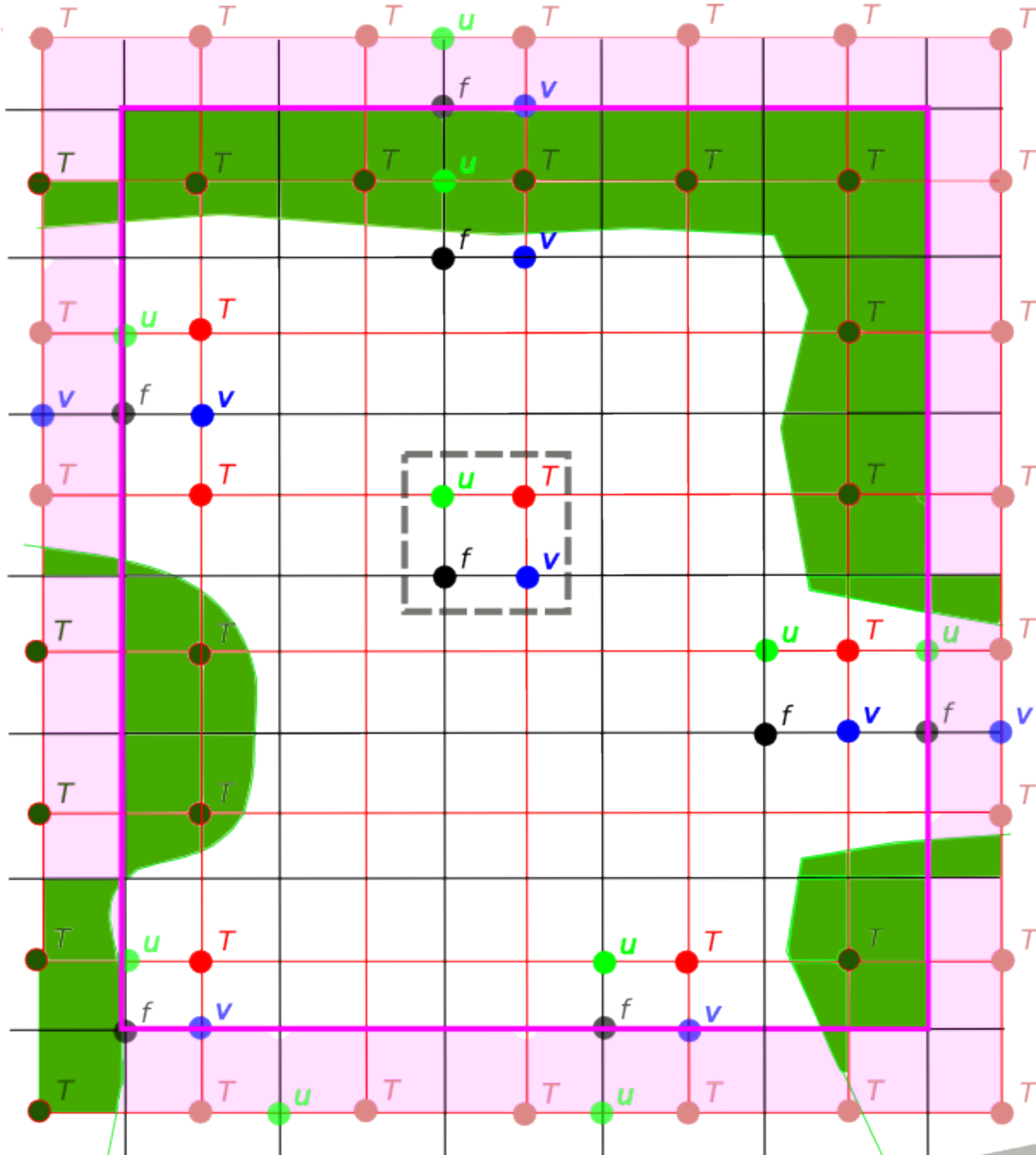
```
!   ** Start of time loop **
DO ncycle=1,itmax

  ! COMPUTE CAPITAL U, CAPITAL V, Z, H
  call invoke(compute_cu_type(CU, P, U),  &
              compute_cv_type(CV, P, V),  &
              compute_z_type(z, P, U, V), &
              compute_h_type(h, P, U, V) )
```

SW offset with Periodic Boundaries



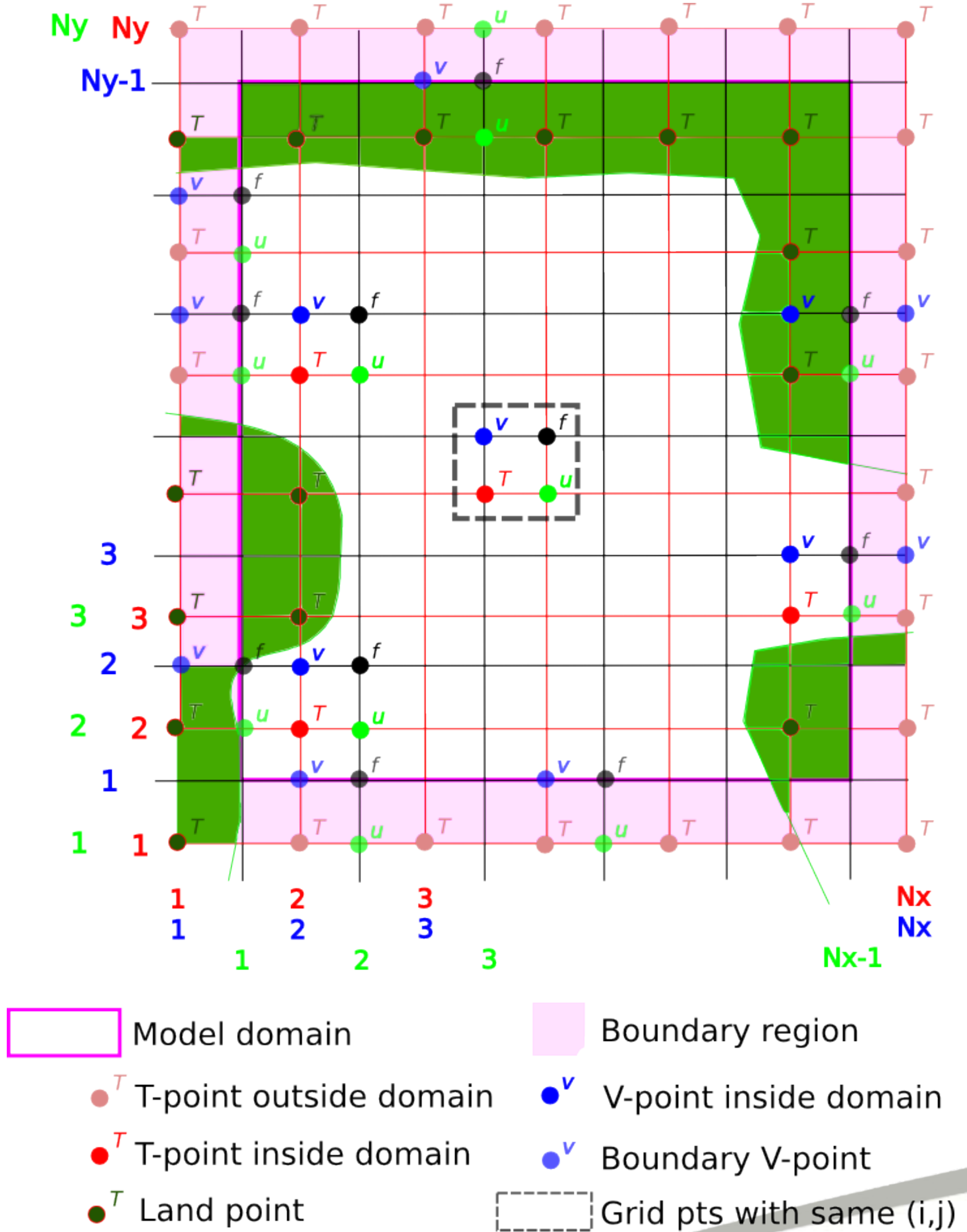
Science & Technology Facilities Council



SW offset with open and closed boundaries

- User defines domain in terms of T points
- Definition *includes* the boundary points
  - Serial implementation doesn't need halos

NE offset with in-place boundary conditions



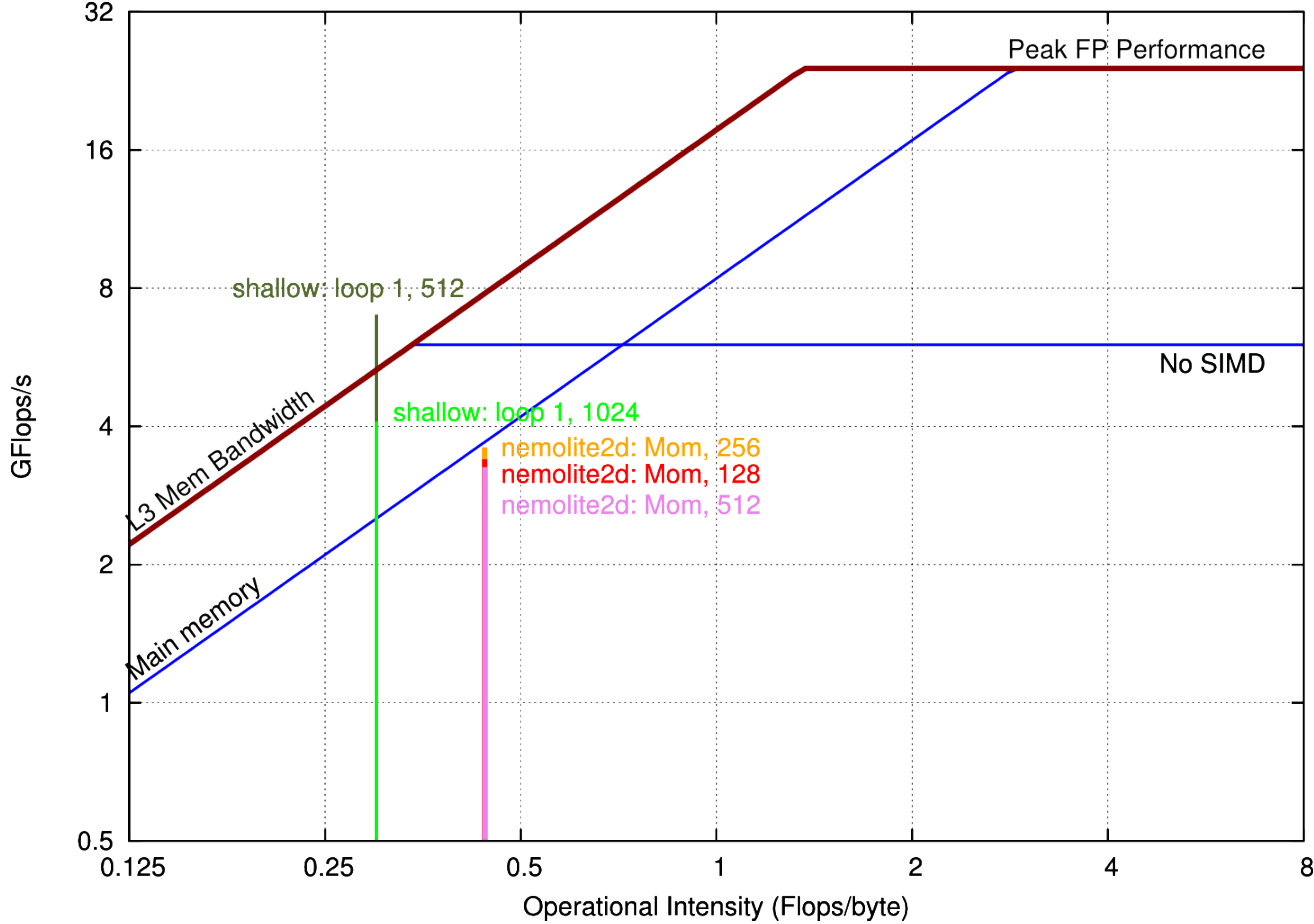
Science & Technology Facilities Council

Code generation to the rescue…

- PSyclone currently has rudimentary support for loop-fusion (and the addition of OpenMP)
- *e.g.* for the time-smoothing section of the code where all 3 loops have same bounds:

```
DO j=1,SIZE(uold,2)
  DO i=1,SIZE(uold,1)
    CALL tsmooth_code(i,j,u…)
DO i=1,SIZE(vold,1)
  DO j=1,SIZE(vold,2)
    CALL tsmooth_code(i,j,v…)
DO i=1,SIZE(pold,1)
  DO j=1,SIZE(pold,2)
    CALL tsmooth_code(i,j,p…)
```

```
DO j=1, SIZE(uold, 2)
  DO i=1, SIZE(uold, 1)
    CALL tsmooth_code(i,j,u…)
    CALL tsmooth_code(i,j,v…)
    CALL tsmooth_code(i,j,p…)
  END DO
END DO
```


Peak FP Performance
No SIMD
L3 Mem Bandwidth
Main memory
shallow: loop 1, 512
shallow: loop 1, 1024
nemolite2d: Mom, 256
nemolite2d: Mom, 128
nemolite2d: Mom, 512
GFlops/s
Operational Intensity (Flops/byte)
32
16
8
4
2
1
0.5
0.125
0.25
0.5
1
2
4
8