Adventures and mis-adventures in hybrid programming

Kevin Stratford (kevin@epcc.ed.ac.uk)

epcc

Introduction

- Distributed and shared memory: hybrid programming
- Ludwig: a lattice Boltzmann code for complex fluids
- Abstraction at the shared memory level
- Shared memory performance
- Hybrid performance
- Capability day results

Hybrid programming

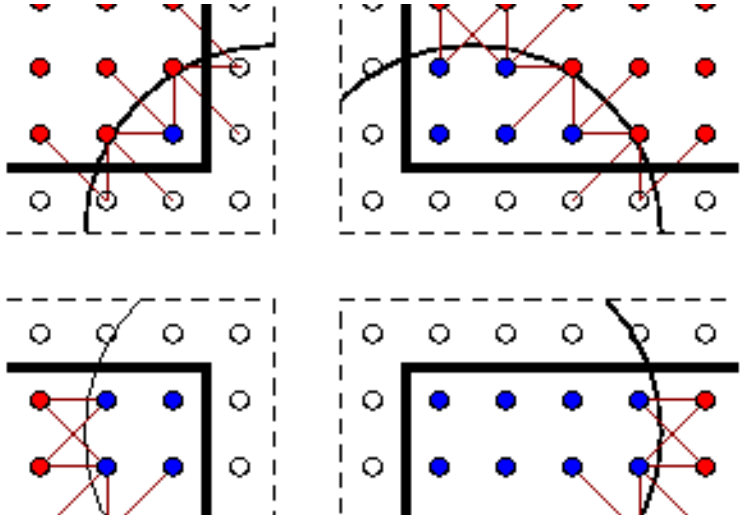
- Taking a broad view...
- Once upon a time ...
  - Single MPI process <=> single core
- Shared memory
  - Single MPI process => many cores
  - OpenMP
- ... to include GPUs ...
  - Single MPI process => single GPU

Ludwig: a lattice Boltzmann code for complex fluids

- Fluid dynamics with a concentration on complex fluids
- Mesoscale problems
- E.g., mixtures, particulate suspensions, liquid crystals, electrokinetics, ...
- Standard ANSI C with MPI for distributed memory
- Abstraction layer for shared memory (CPU, GPU)

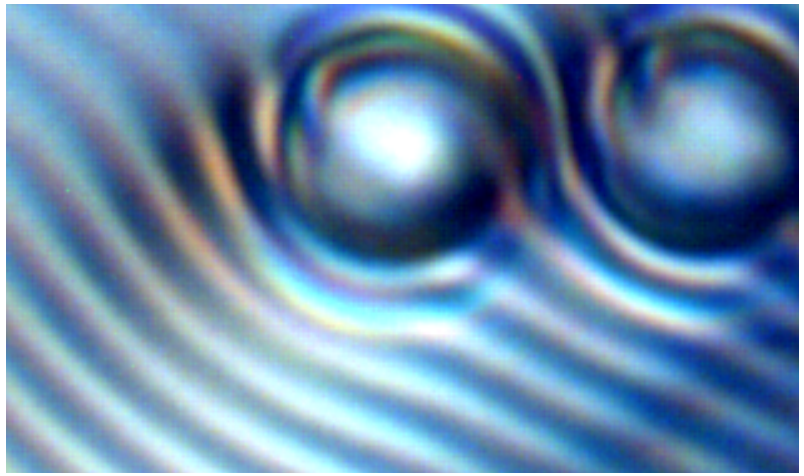
$$\partial_t \rho + \partial_\alpha (\rho u_\alpha) = 0$$

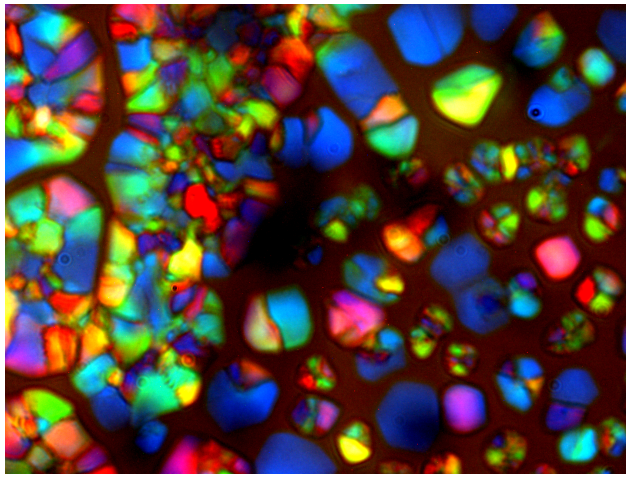
$$\partial_t (\rho u_\alpha) + \partial_\beta (\rho u_\alpha u_\beta) = \partial_\beta P_{\alpha\beta} + \eta \partial_\beta (\partial_\alpha u_\beta + \partial_\beta u_\alpha)$$



Liquid crystal (the standard benchmark)

- Hydrodynamics plus
- Coarse-grained order parameter (5 independent components)
- Coupling between the two





Experimental images: Anne Pawsey

Abstraction

- To address portability, an abstraction is desirable
- Developed a shim layer for CUDA/HIP/OpenMP
- Some simple extensions required to admit OpenMP (1 block, n threads)

```
/* ... host code ... */

tdpLaunchKernel(my_kernel, ...)

/* ... host code ... */


__global__ void my_kernel(...) {

  /* ... kernel code ... */

}
```

Memory

- Explicitly handled

```
/* ... host code ... */

tdpMalloc((void **) &f_d, ndata*sizeof(double));

/* ... host code ... */


__global__ void my_kernel(f_d, ...) {

  /* ... kernel code ... */

}
```

Kernels

- Extra step required for work sharing in OpenMP

```
/* ... host code ... */

tdpLaunchKernel(my_kernel, ...);  /* "Start threads" */

__global__ void my_kernel(...) {

  int index = 0;

  for_simt_parallel(index, niterations, 1) {

    /* .. per thread or work item operations ... */

  }
}
```

Various other features

- E.g., reductions; schematically:

```
__global__ void my_reduction(..., result) {

  __shared__ double rcontrib[N_PAD*MAX_THREADS_PER_BLOCK];

  /* ... accumulate per-thread contributions ... */

  __syncthreads();

  tdpAtomicAdd(&result, ...)
}
```

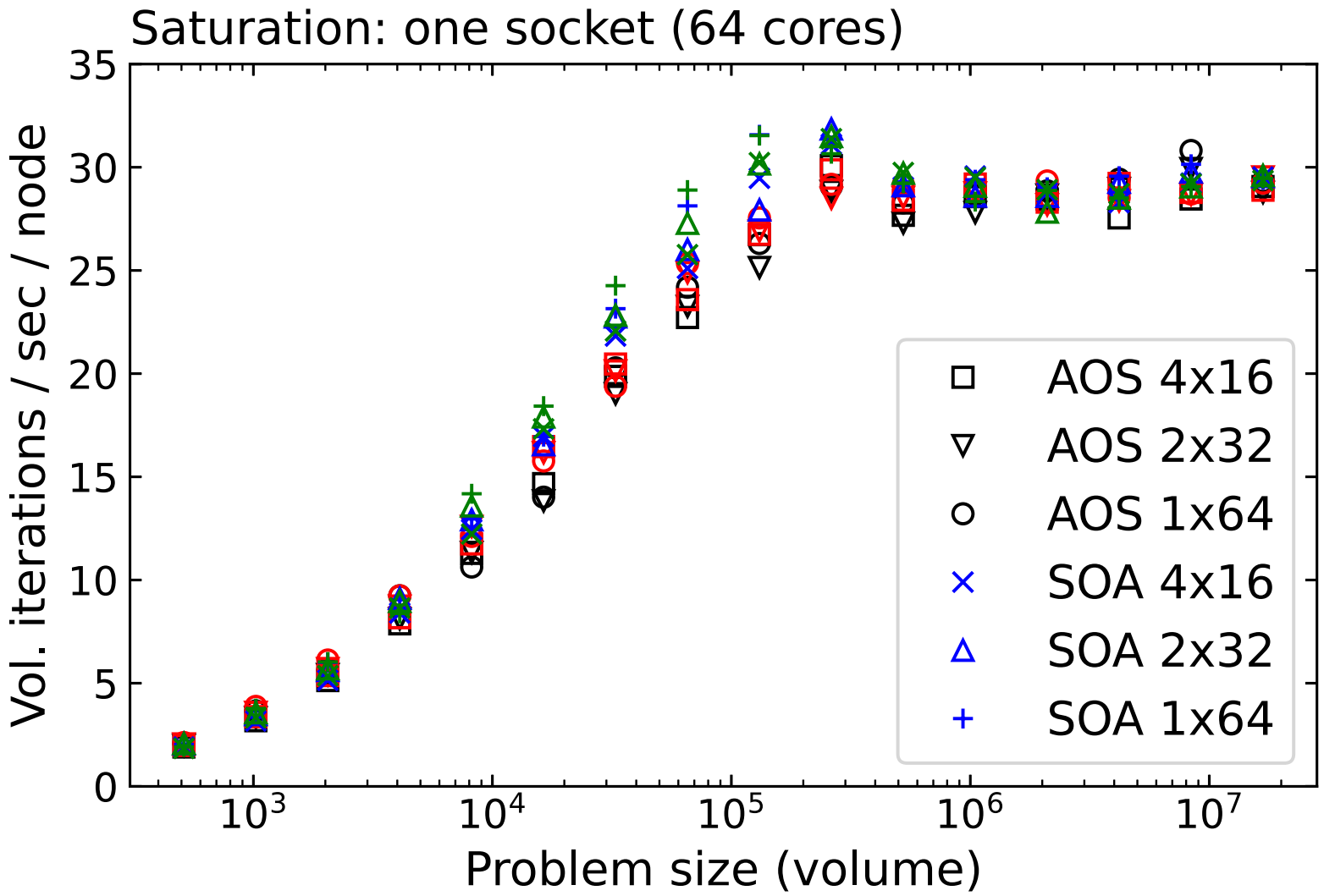
Abstraction

- Abstraction says nothing about memory layout
- A separate mechanism is used to provide either
  - Array of structures (v_x v_y v_z) (v_x v_y v_z) ...
  - Structure of arrays (v_x v_x v_x ...) (v_y v_y v_y ...) (v_z v_z v_z ...)

ARCHER2 runs

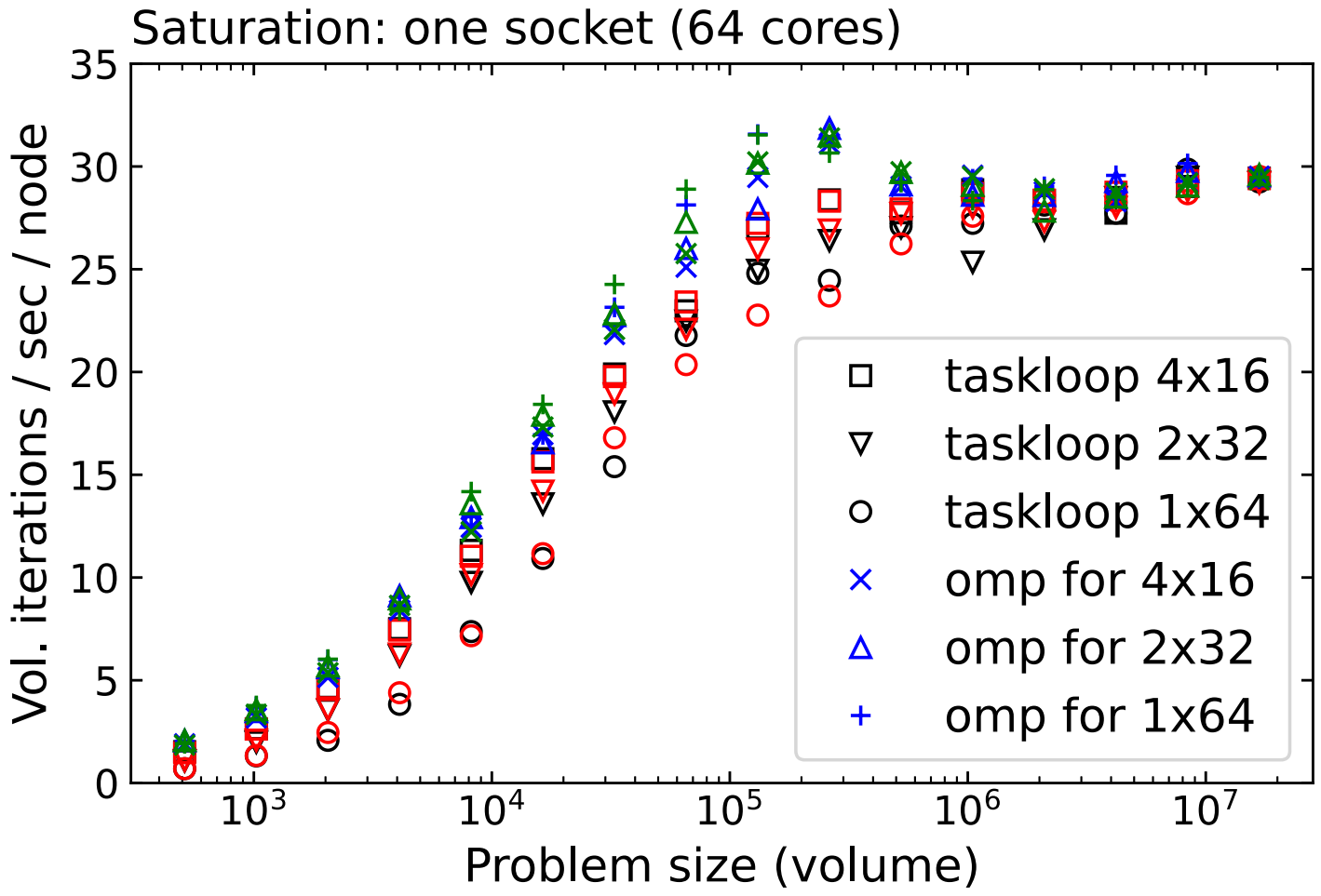
- Usually PrgEnv-aocc
- Run at less 64 cores (1 socket)
- Performance assessed via figure of merit
  - Volume (system size in lattice sites) x no. iterations / second / resource
  - Volume normalised by 128x128x64

ARCHER2 hybrid

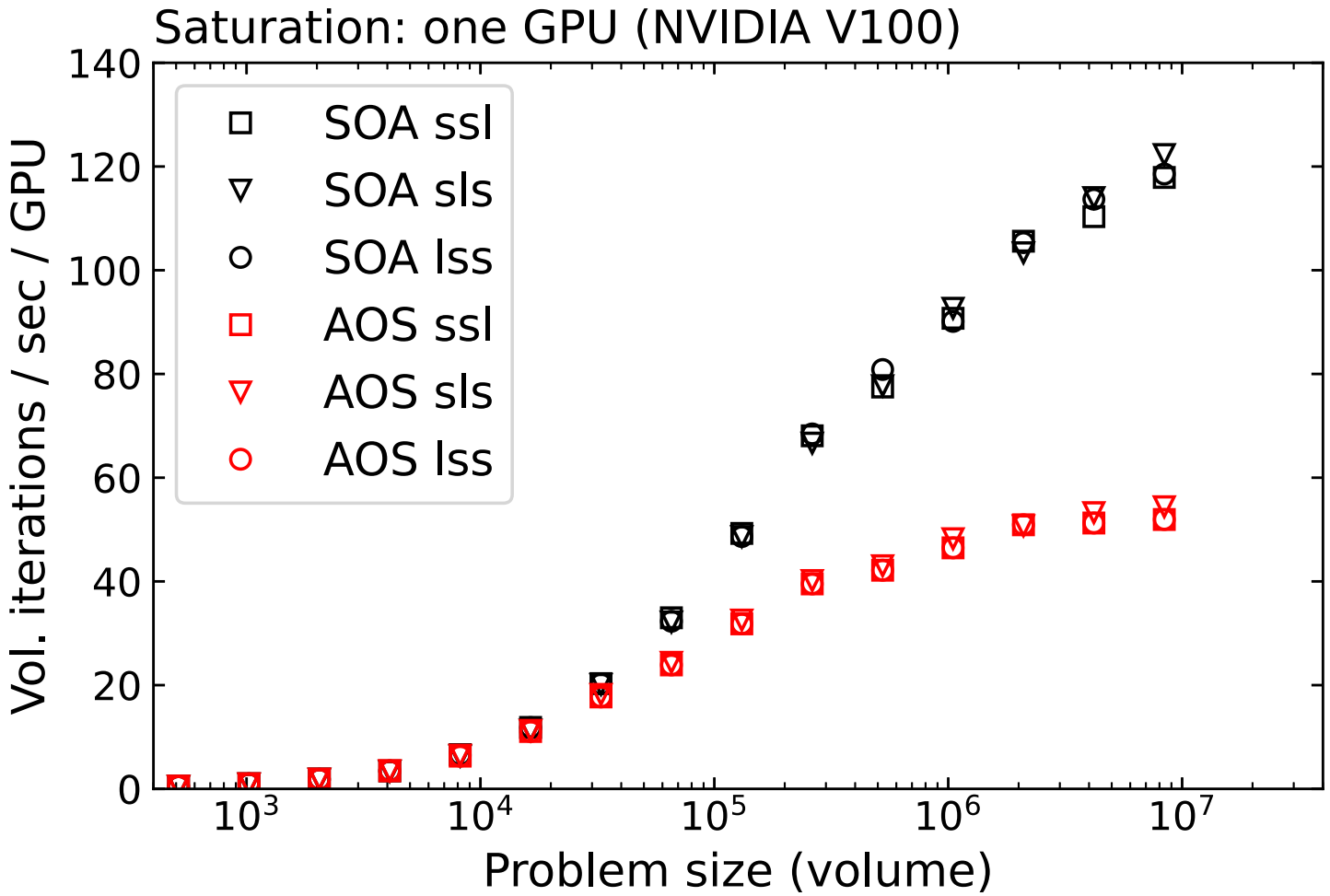


Misadventure

- Try to replace OpenMP for with taskloop



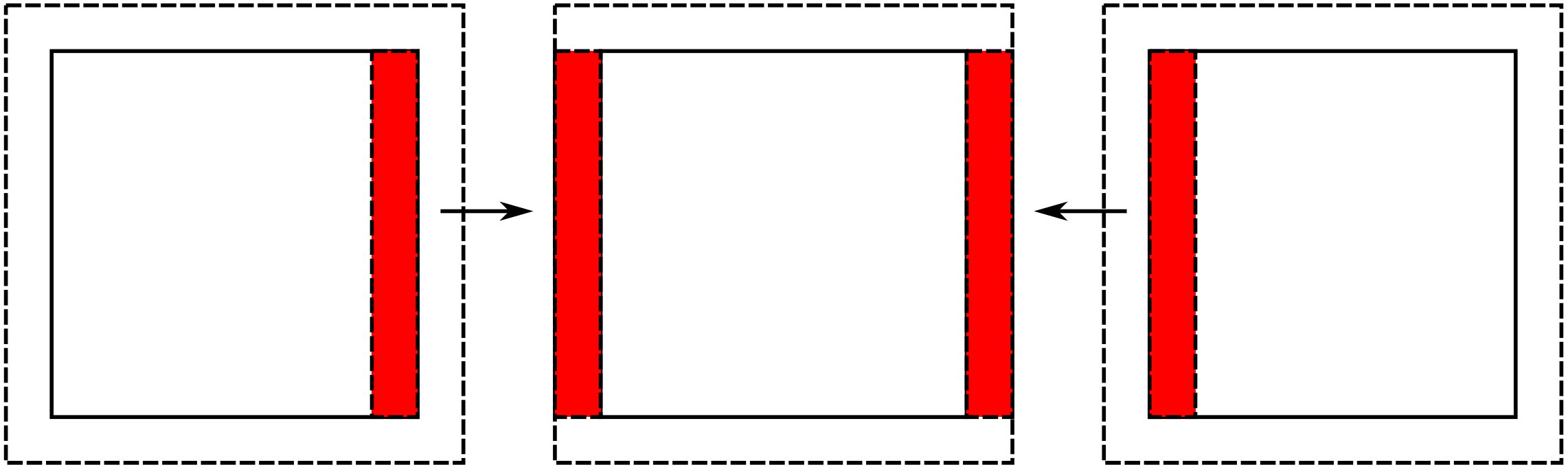
Compare: GPU (NVIDIA V100)



Hybrid again

- Need halo swap
- Problem: serialisation on host at point of halo swap
- MPI Data types?
- OpenMP?
- Thread model?
  - MPI_THREAD_SINGLE
  - MPI_THREAD_FUNNELLED
  - MPI_THREAD_SERIALIZED
  - MPI_THREAD_MULTIPLE

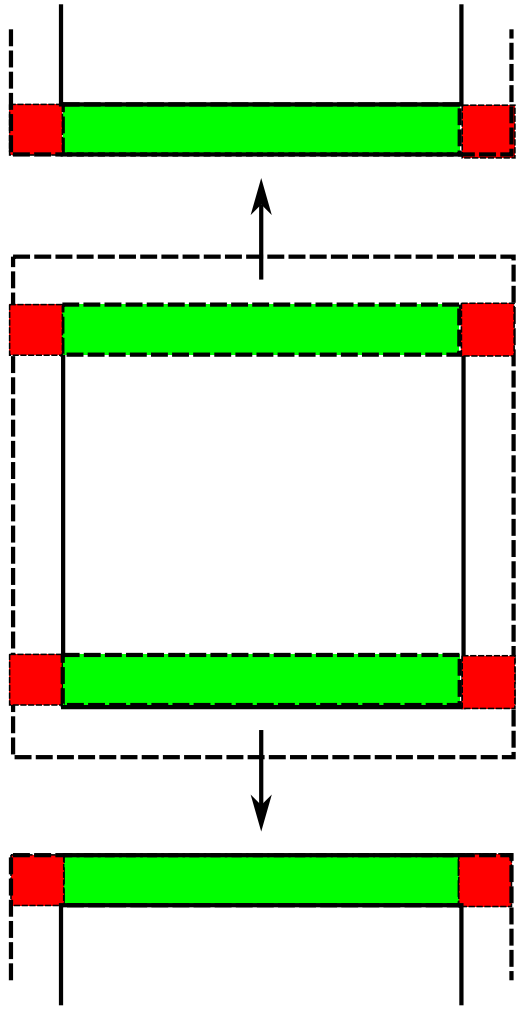
Halo swap: a traditional "trick"



- One send/recv pair per process per direction

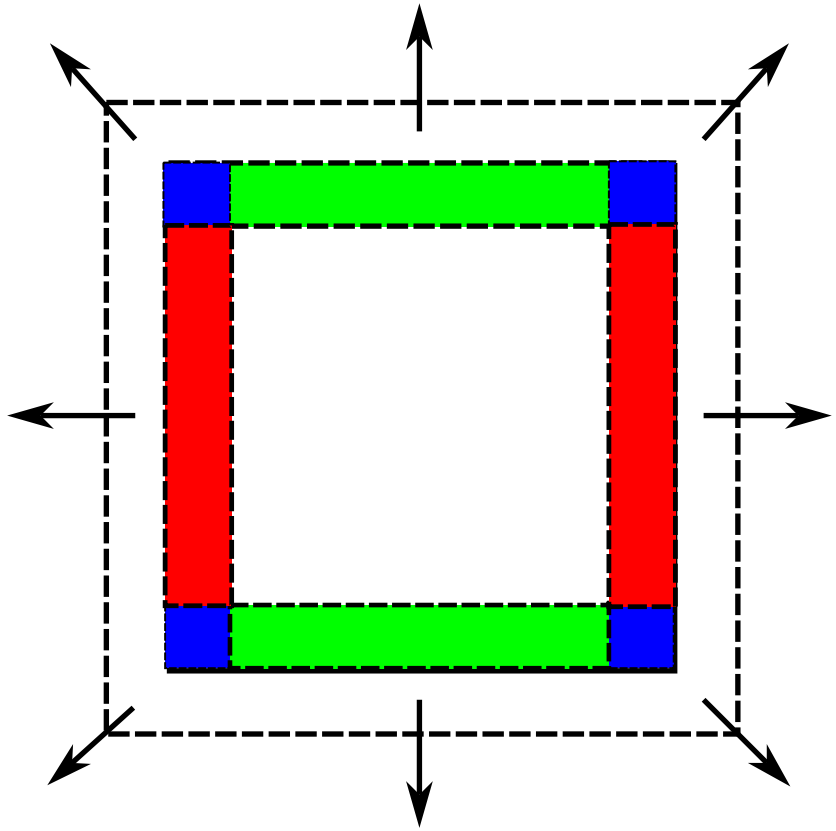
Halo swap: a traditional "trick"

- Corners appear "for free"
- Few messages; straightforward code
- But: must synchronise between messages in different co-ordinate directions



Alternative approach

- Separate message in each direction
- More messages, but no synchronisation



OpenMP

- Halo swap episode (MPI_THREAD_FUNNELLED)

```
/* Post non-blocking recvs for incoming messages... */

#pragma omp parallel
{
   for (int ireq = 0; ireq < nreq; ireq++) {
     load_send_buffer(..., ireq);
   }
}

for (int ireq = 0; ireq < nreq; ireq++) {
  MPI_Isend(...); /* Non-blocking out-going message */
}
```

Halo swap (cont...)

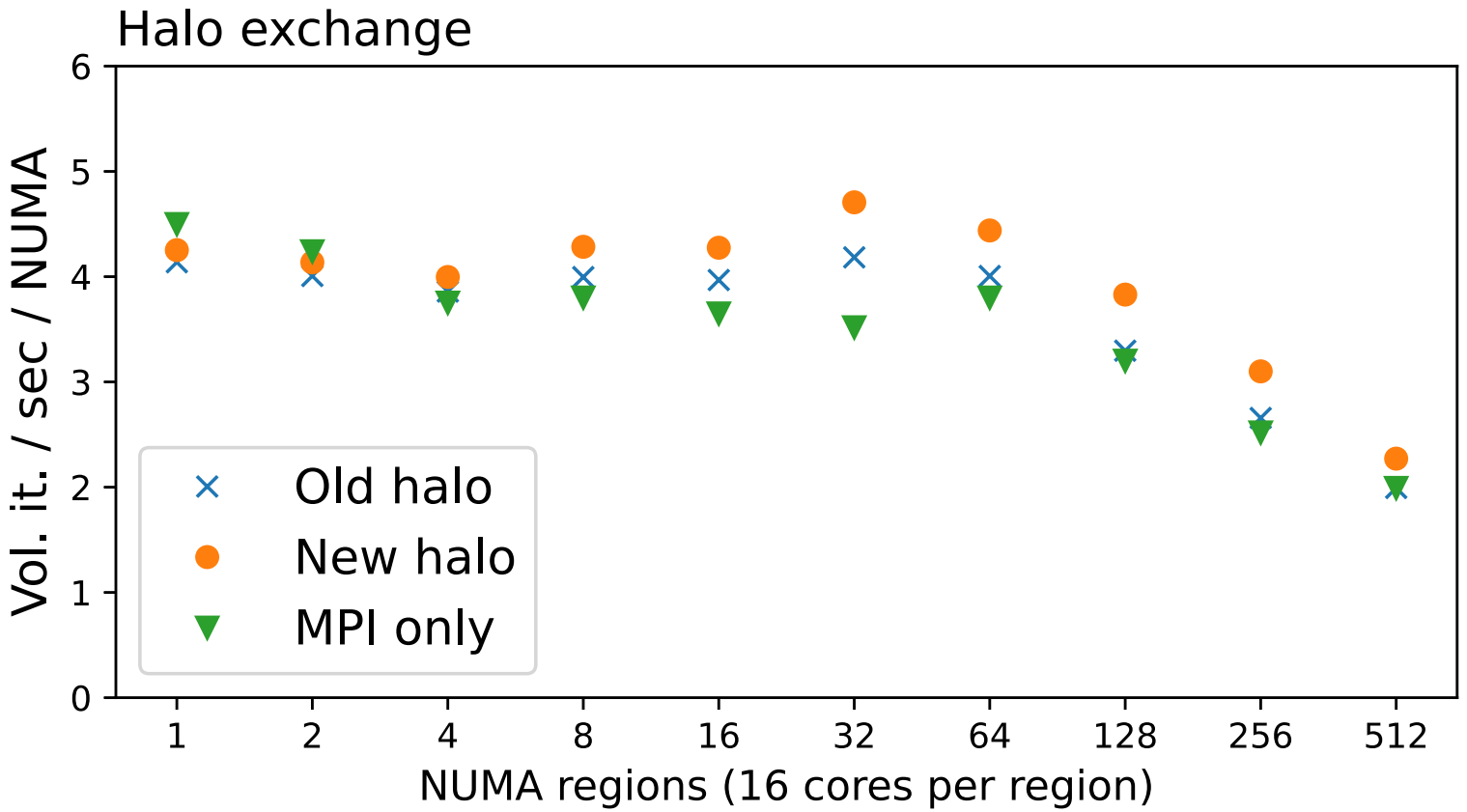
- Wait and unpack the message buffers
- Work sharing over (spatial) extent of the buffer

```
MPI_Waitall(); /* wait for all requests */

#pragma omp parallel
{
   for (int ireq = 0; ireq < nreq; ireq++) {
      unload_recv_buffer(..., ireq);
   }
}

/* ... and finished */
```

Strong scaling



Misadventure (I...)

- With MPI_THREAD_FUNNELLED :

```
for (int ireq = 0; ireq < nreq; ireq++) {

  MPI_Waitany(...);

  #pragma omp parallel
  {
     unload_recv_buffer();
  }
}
```

Misadventure (II...)

- With MPI_THREAD_SERIALIZED

```
#pragma omp parallel
{
    #pragma omp single
    {
       MPI_Waitany(...)
       #pragma omp task
          unload_recv_buffer(...);
    }
}

/* ... and finished */
```

Misadventure (III...)

- With MPI_THREAD_MULTIPLE

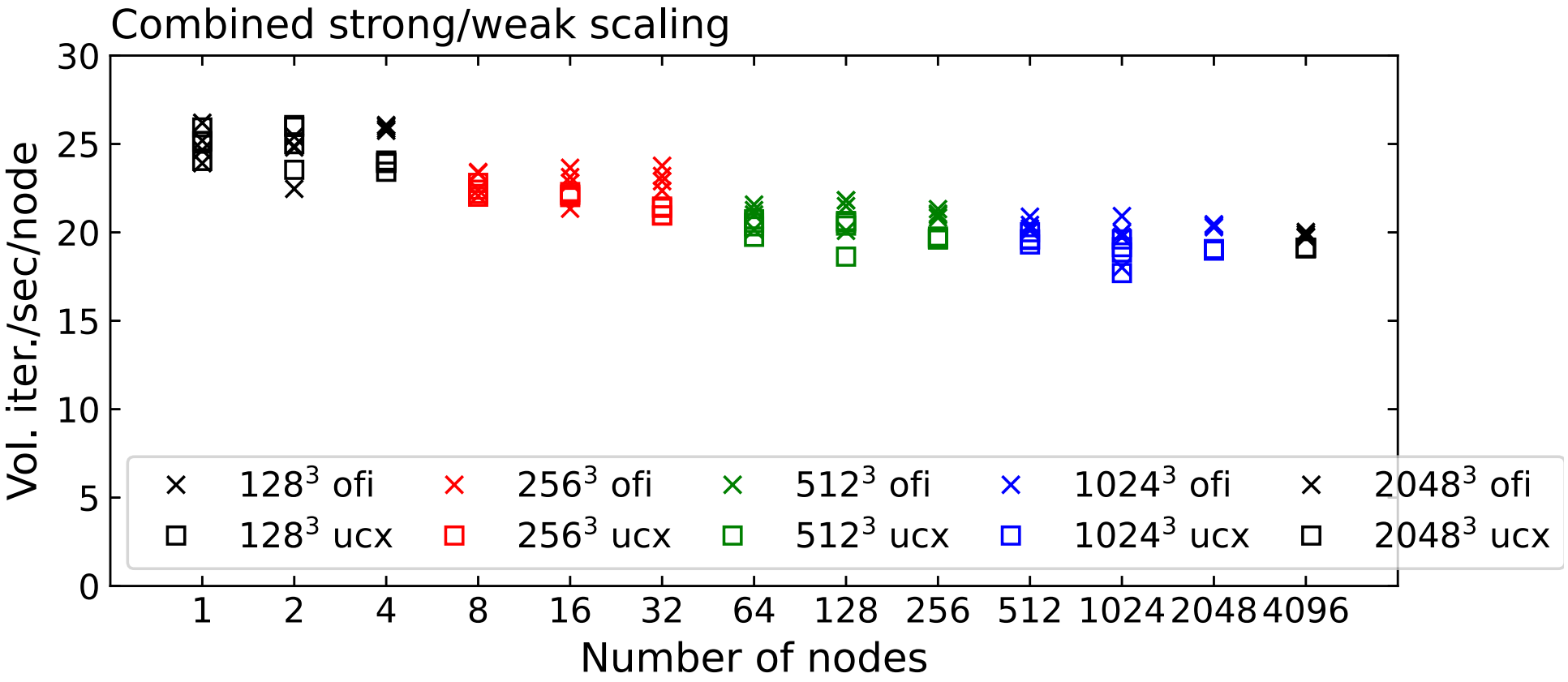
```
#pragma omp parallel
{
   MPI_Waitany(...)
   unload_recv_buffer(...);
}

/* ... and finished */
```

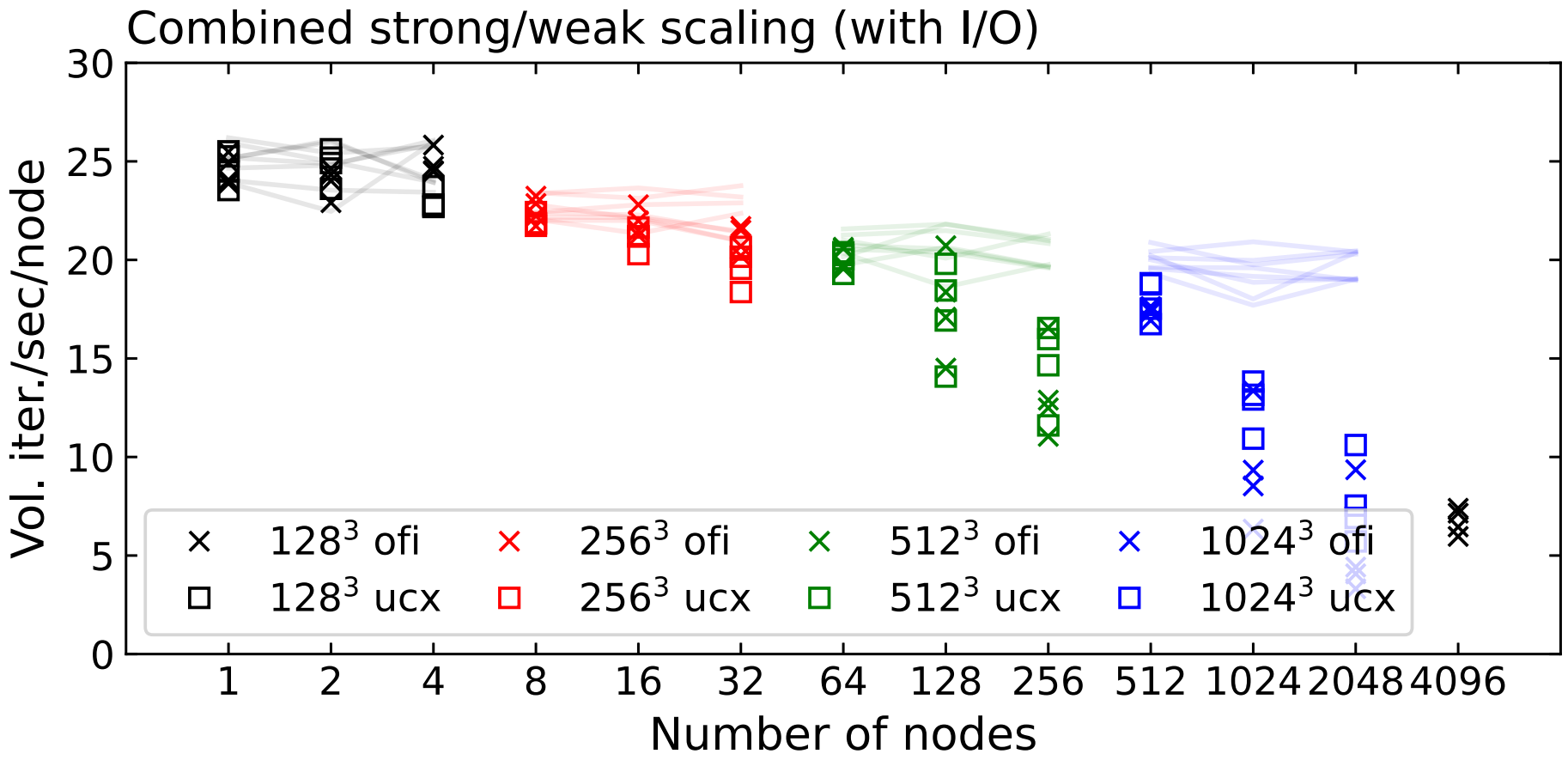
Capability day

- Look at scaling without and then with I/O
- I/O is performed via aggregation in OpenMP to single buffer per MPI
- MPI/IO to single file
- Asynchronous output is possible (not studied)
- Multiple file possible (not shown studied)

Capability day (no I/O)



With MPI/IO to single file



Summary

- Hybrid performance can be effective to the socket level
- OpenMP required in some form at the halo swap
- Saturation on one node provides a useful guide to limit of strong scaling
- Capability day used to investigate upper end of scaling

```
https://github.com/ludwig-cf/ludwig
```

Equations

$$f = \tfrac{1}{2}A_0(1-\tfrac{\gamma}{3})Q_{\alpha\beta}^2 - \tfrac{1}{3}A_0\gamma Q_{\alpha\beta}Q_{\beta\sigma}Q_{\sigma\alpha} + \tfrac{1}{4}A_0\gamma(Q_{\alpha\beta}^2)^2$$

$$+ \tfrac{1}{2}K[(\partial_\beta Q_{\alpha\beta})^2 + (\epsilon_{\alpha\zeta\sigma}\partial_\zeta Q_{\sigma\beta} + 2q_0 Q_{\alpha\beta})^2]$$

$$\partial_t Q_{\alpha\beta} + \partial_\zeta(u_\zeta Q_{\alpha\beta}) - S(\partial_\beta u_\alpha, Q_{\alpha\beta}) = \Gamma H_{\alpha\beta}$$

$$H_{\alpha\beta} = -\frac{\delta\mathcal{F}}{\delta Q_{\alpha\beta}} + \tfrac{1}{3}Tr\left(\frac{\delta\mathcal{F}}{\delta Q_{\alpha\beta}}\right)\delta_{\alpha\beta}$$

I/O settings

```
$ lfs setstripe -c 12 .


export MPICH_MPIIO_HINTS="*:cray_cb_write_lock_mode=2,
                          *:cray_cb_nodes_multiplier=8"
```