ARCHER CHAMPIONS

5TH / 6TH SEPTEMBER 2016

Performance Portability (and TargetDP)

Kevin Stratford, EPCC

Alan Gray, EPCC

Overview

- Future HPC systems will be based around
  - GPU / KNL / “Accelerators”
  - Broadly: threaded models, require vectorisation
  - Increasingly complex memory hierarchy
- Scientific software needs to keep up
  - *Performance Portability*
  - Probably want evolution rather than revolution
  - See also eCSE (tomorrow)

Problems

- Maintaining a scientific application
  - Lifetime of decades
  - Effort for development / upkeep may be episodic
- Performance Portability
  - Run effectively on whatever hardware is available
  - Do not want different versions / programming models
- Effectively:
  - Achieve high (or, at least decent) % of peak flops, or
  - Achieve high % of available memory bandwidth

Considerations (I)

- Memory layout
  - E.g., for a vector field v(r) with components (vx, vy, vz)
- For CPU may want, in memory:
  - `(vx vy vz) (vx vy vz) (vx vy vz) ...`
- For GPU may want:
  - `(vx vx vx ...) (vy vy vy ...) (vz vz vz ...)`
- For vectorisation, may want:
  - `(vx vx vy vy vz vz) (vx vx vy vy vz vz) ...`
- Need to abstract array declaration/indexing

Considerations (II)

- Memory movement
  - Many applications sensitive to memory bandwidth
- Explicit memory transfers
  - More painful
  - May be more efficient / transparent (cf. MPI)
- Important to use specific memory
  - E.g., “constant’ memory on GPU

Considerations (III)

- Execution
  - Based around kernels
  - Single threaded, threaded, (CPU, GPU)
- Abstract
  - At least the thread model
  - Common atomic/reduction operations
- May want task-based model
  - Allow specification and management of tasks
  - Asynchronous execution (dependencies)

Solutions

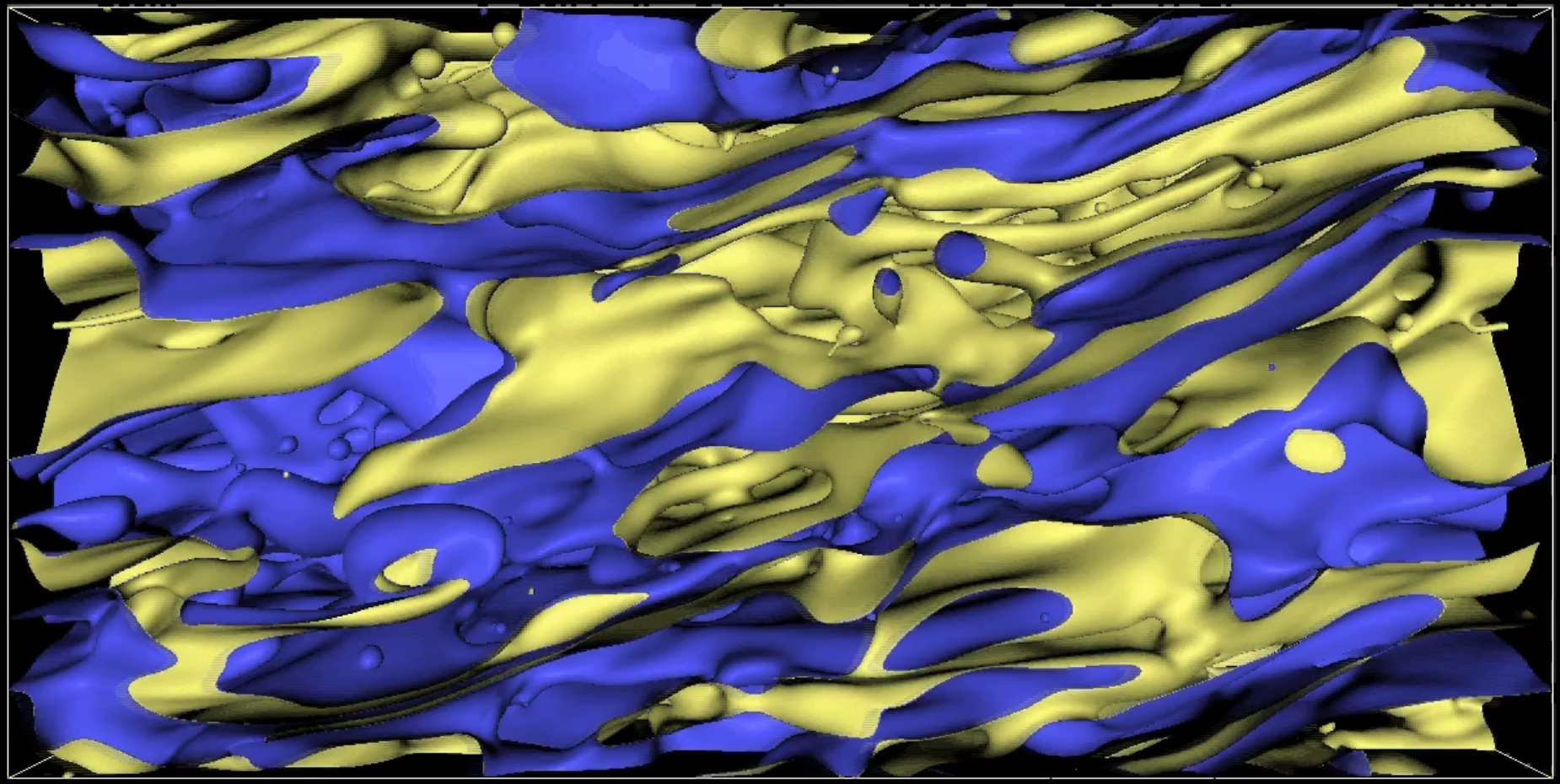
- SYCL
  - C++ abstraction layer for single source
  - Template functions for host / device code
  - Related to OpenCL/SPIR https://www.khronos.org/sycl
  - Not clear whether suited to scientific applications
- Kokkos
  - C++ for performance portable applications
  - Addresses memory access patterns
  - Addresses parallel execution, atomics, memory management…
  - Sandia National Laboratories https://github.com/kokkos/kokkos

Solutions (II)

- OpenMP 4.x
  - C/C++/Fortran compiler directives
  - Addresses thread and task execution (host / target)
  - Addresses memory movement (implicit)
  - Tried and tested in the scientific community
- DIY

Ludwig's Progress

- Ludwig
  - Code developed at Edinburgh for complex fluids
  - Mixtures / suspensions / liquid crystals / active fluids
  - Coarse-grained dynamics + hydrodynamics



Ludwig's Progress

- Ludwig
  - Code developed at Edinburgh for complex fluids
  - Mixtures / suspensions / liquid crystals / active fluids
  - Coarse-grained dynamics + hydrodynamics
- Plain old ANSI C
  - Message passing for domain decomposition
  - Around in various stages of development since late 90s

Ludwig's Progress

- 2009 MSc student Alan Richardson
  - Moved one/two main kernels to CUDA
- 2011-2014 Alan Gray (CRESTA)
  - Developed more wide-ranging CUDA branch
- 2014-2016 Alan Gray (ARCHER ECSE)
  - Introduced abstraction layer "TargetDP"

Non-vectorised kernel

```
__global__ void pth_kernel(kernel_ctxt_t * ktx, pth_t * pth, fe_t * fe) {

  int kindex;
  __shared__ int kiter;

  kiter = kernel_iterations(ktx);

  __target_simt_parallel_for(kindex, kiter, 1) {

    int ic, jc, kc, index;
    double s[3][3];

    ic = kernel_coords_ic(ktx, kindex);
    jc = kernel_coords_jc(ktx, kindex);
    kc = kernel_coords_kc(ktx, kindex);
    index = kernel_coords_index(ktx, ic, jc, kc);

    fe->func->stress(fe, index, s);
    pth_stress_set(pth, index, s);
  }
```

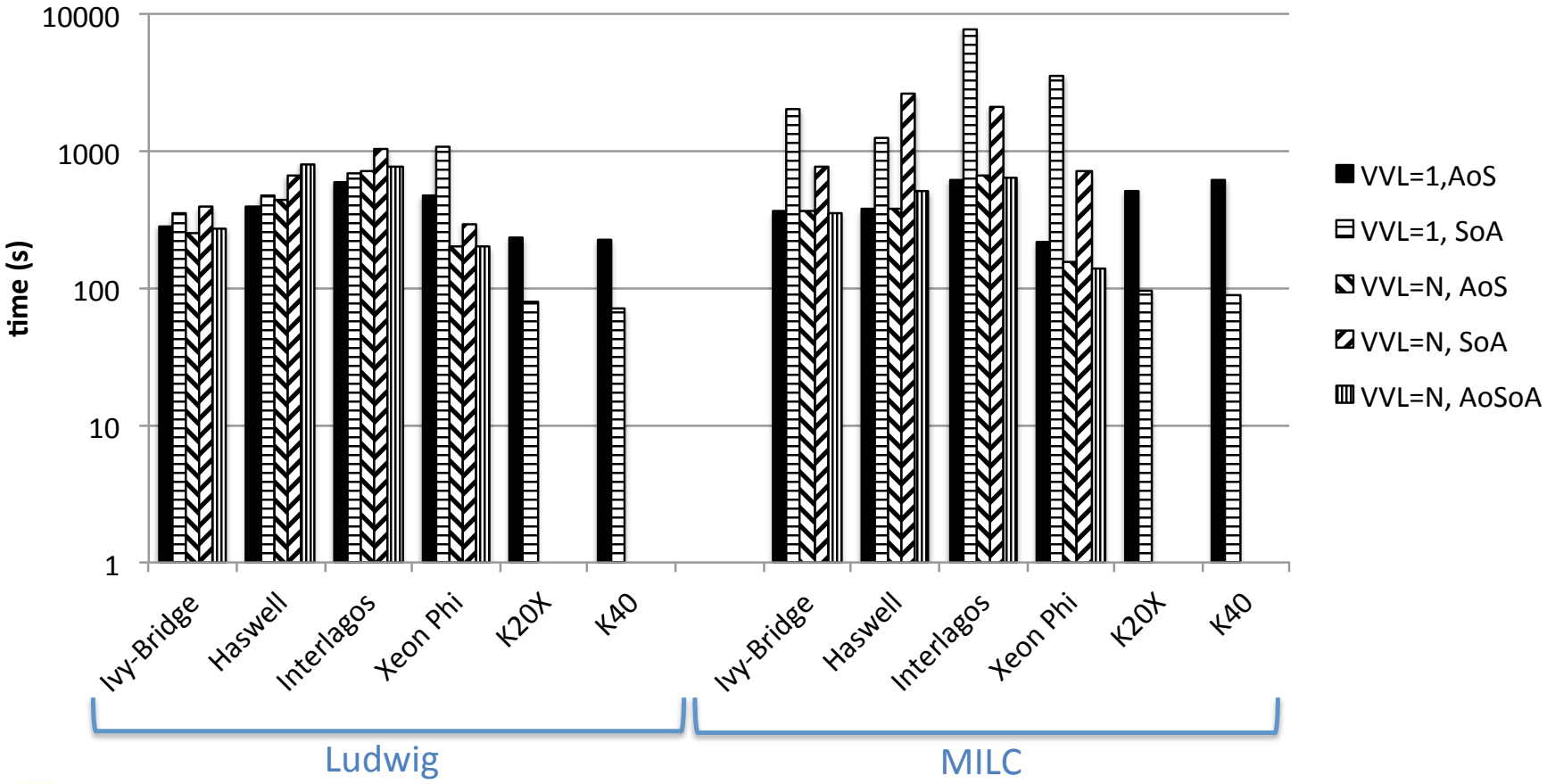
Vectorised kernel

```
kiter = kernel_vector_iterations(ktx);

__target_simt_parallel_for(kindex, kiter, NSIMDVL) {

  int index;
  int ia, ib, iv;
  double s[3][3][NSIMDVL];

  index = kernel_baseindex(ktx, kindex);

  fe->func->stress_v(fe, index, s);

  for (ia = 0; ia < 3; ia++) {
    for (ib = 0; ib < 3; ib++) {
      for (iv = 0; iv < NSIMDVL; iv++) {
        pth->str[addr_rank2(pth->nsites,3,3,index+iv,ia,ib)] = s[ia][ib][iv];
      }
    }
  }
}
```

Memory

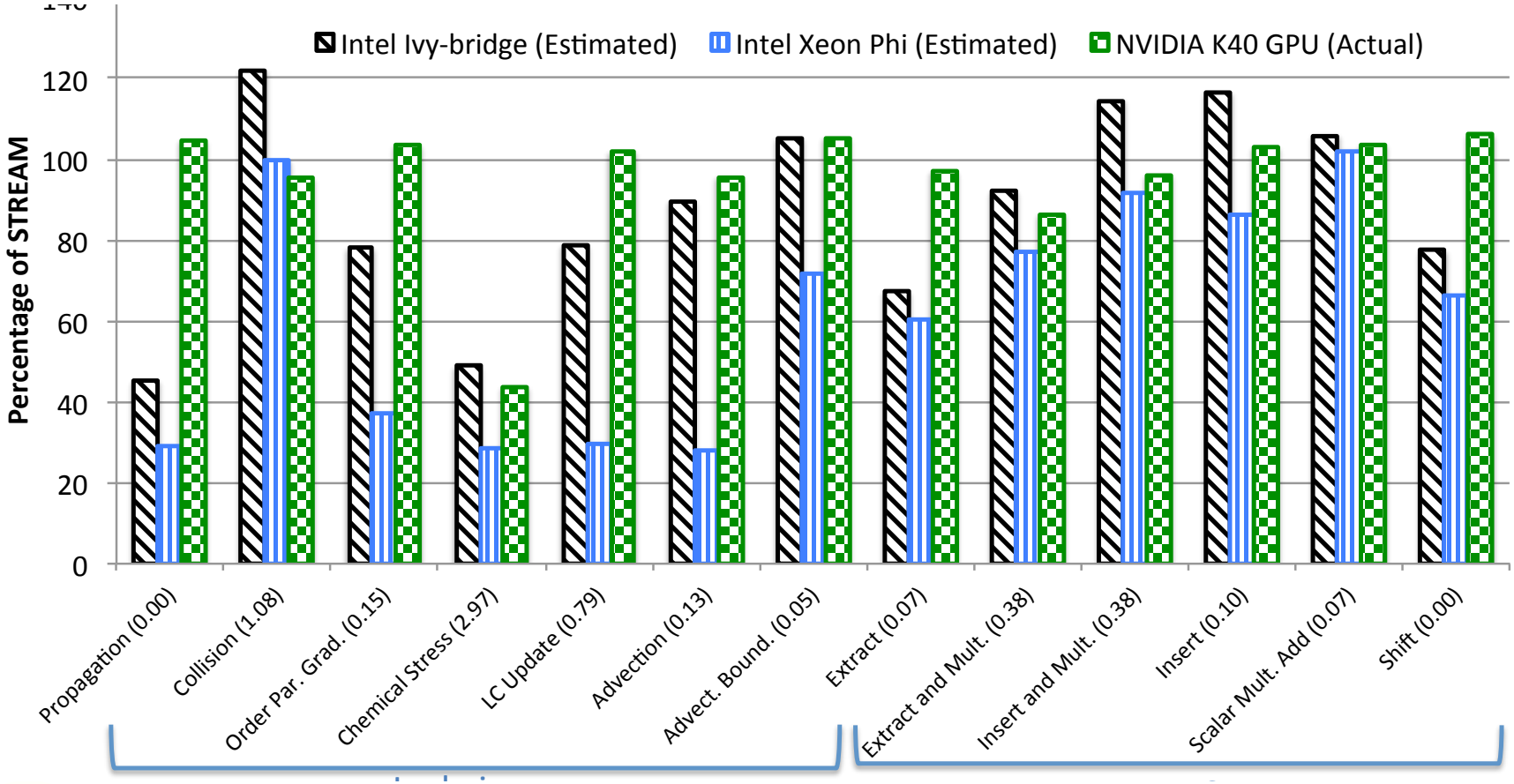
```
struct pth_s {
  ...;                    /* State, data */
  pth_t * target;         /* Target memory */
};

...

if (ndevice == 0) {
  obj->target = obj;
}
else {

  targetCalloc((void **) &obj->target, sizeof(pth_t));
  copyToTarget(&obj->target->nsites, &obj->nsites, sizeof(int));

  targetCalloc((void **) &tmp, 3*3*obj->nsites*sizeof(double));
  copyToTarget(&obj->target->str, &tmp, sizeof(double *));
}
```

Memory arrangement



archer

Memory performance



archer

http://www.archer.ac.uk
support@archer.ac.uk