This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2018-6597C

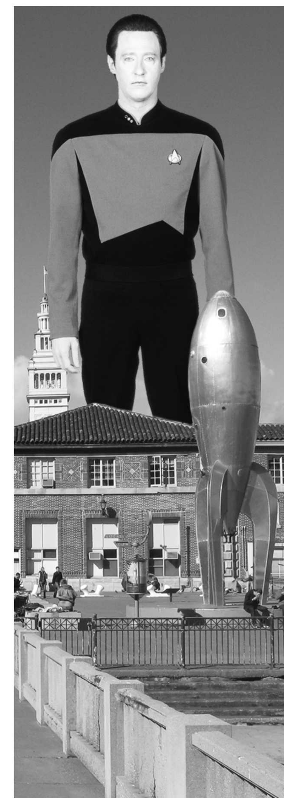
Sandia National Laboratories

You get the pointer, but only on my terms:

Control local data access for better use of modern computer hardware & programming models

Mark Hoemmen

SIAM Annual 2018



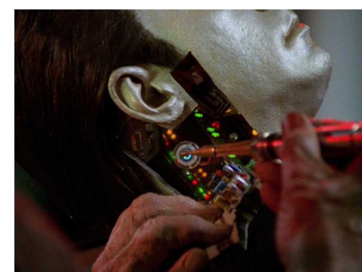
"Big Data" (Josh Lee, *Medium*, 03 Mar 2014)

Many apps have large objects

Examples
- Discretization mesh; mesh degrees of freedom
- Sparse matrix; its factorization or preconditioner
- Particles in space (e.g., PIC)

Large means
- May not fit in a single node's memory
- Contain many things (particles, elements, …)
- Avoid / minimize copying

Access to local data



Access may have side effects

Large objects have local data

Local may mean:
- (Physics) Contiguous, same material, …
- (Discretization) Common finite element type, …
- (Solver) Subdomain, aggregate, ADI line, …
- (Computer) to my MPI process, in cache, …

Access means:
- Read-only, write-only, or read-and-write
- Not all data, just the "local" part (see above)
- Where I want them: e.g., on GPU or CPU (possibly with automatic copy back on write)

Must respect locality for correctness & performance!

Current solutions are brittle

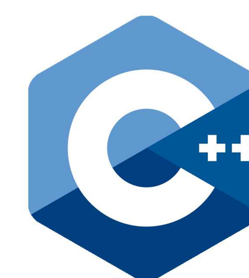
Your code can get noodly

Typical solution: State machine (spaghetti)
- "Can I access the pointer here?"
- e.g., "filling linear system" vs. "solving"

Makes software cluttered & brittle
- Hard to add / remove states
    - Mesh changes? Load rebalancing?
- Bound to programming model; hard to
    - Add GPU support
    - Use distributed-memory task-based models

Solution, Part 1: Custom "smart pointers" to local data

- Pointer lifetime = access lifetime (C++ RAII)
- Declare access intent: {Read,Write}-only or R+W
- Declare access location: CPU, GPU, PGAS, …
- Define ownership model (action on end of life)
    - Nonowning: Do nothing (like T*)
    - Owning: Call deallocator function
- Control access sharing:
    - None (unique_ptr)
    - Peer-to-peer (shared_ptr)
    - 1 owner + slices (Sutter's deferred_ptr)



Custom pointers not enough

- Local data may have unbounded lifetime
- How to prevent incorrect access in 2 different memory spaces concurrently?

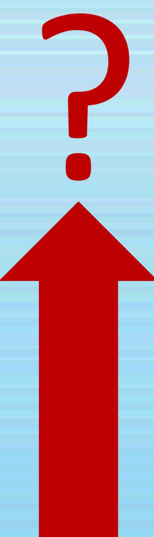
```
{
  auto A_gpu = readOnly(A).on(gpu);
  {
    auto A_host = readWrite(A).on(host);
    modifyOnHost(A_host);
  } // Did A_host's destructor copy back to GPU?
  readOnGpu(A_gpu); // Did this see A_host changes?
}
```

- Fix: Object, not user, must control access scope!

Solution, Part 2: withLocalAccess

- New syntax built on C++ variadic templates
- Local data given as arguments to user-provided "callback" function – not visible outside
- Makes full use of Solution, Part 1

```
// We call user's function with local data.
withLocalAccess(
  [=] (auto field1_lcl,   // const entries
       auto field2_lcl,   // nonconst entries
       auto field3_lcl)   // nonconst entries
  {
    accessReadOnly(field1_lcl);
    accessWriteOnly(field2_lcl);
    accessReadWrite(field3_lcl);
    doStuffWith(field1_lcl, field2_lcl, field3_lcl);
  },
  readOnly(field1).on(Cuda), // imitate C++20 executors
  writeOnly(field2),
  readWrite(field3).on(HostPinned),
);
```



How long is that pointer fresh?

Typical pattern
- Get pointer to object's local data
- Keep it around until ???

Is a pointer to local data still valid if
- Global object (mesh, matrix, …) changes?
- Load rebalance? Restart w/ diff process count?
- I mix calls to GPU-ready & CPU-only code?
- App must free GPU memory to make space?

Should it be valid? e.g., stale references (ref. counting)

Example: Access sparse matrix A

```
{ // Copy/sync to GPU if needed.  Mark modified on GPU.
  auto A_lcl = readWrite(A).on(gpu);
  parallel_for({gpu, {0, getNumRows(A_lcl)}},
    [=] (const int i) {
      auto row = getRow(A_lcl, i); // view of row
      for (auto&& [ind, val] : row) { val=f(ind,val); }
    });
}
{ // A last modified on GPU, so may sync (for GPU kernel)
  // & copy to host (if no UVM).
  auto A_lcl = readOnly(A).on(host);
  auto [ptrs, inds, vals] = getRawCSR(A_lcl);
  readOnlyThirdPartyLibraryFunction(ptrs, inds, vals);
}
{ // PGAS: sync on window & start access epoch
  auto A_lcl = writeOnly(A).on(window);
  writeTo(A_lcl);
} // A_lcl's destructor ends access epoch
```

Use case: Use a Kokkos-like programming model (separate; see github.com/kokkos) to write thread-parallel GPU code that needs read+write access.

Use case: Legacy CPU-only code needs to access sparse matrix that prefers GPU storage.

Use case: MPI 1-sided (PGAS). Optimize for write-only by using MPI_Put, not MPI_Accumulate.

This works best when developers:

- Know at compile time what objects they want to access, & only need a few at a time
    - Expression-based multiphysics codes already use a different approach
- Know how to iterate over all local data efficiently
    - e.g., have an existing mesh iteration abstraction that tiles, vectorizes, etc.

U.S. DEPARTMENT OF ENERGY
NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology &Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
SAND No. ____________

Sandia National Laboratories