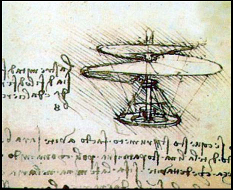
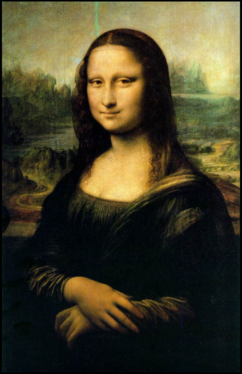


art and the Intensity Frontier

Concurrency Forum Workshop, Ferburary 4-6 2013



Chris Green
SCD-ADSS-SSI

Fermilab
Fermi National Accelerator Laboratory
Office of Science / U.S. Department of Energy
Managed by Fermi Research Alliance, LLC

Outline

- Overview of direction.
- Details.
- Studies.

Overview of direction

- Mostly incremental changes to existing **art** framework.
- Choice of **TBB** for multi-threading framework.
- Event-level and user-level (intra-module) parallelism.
- Initially control-flow, not demand-driven.

Details: choice of parallelism toolkit

- **OpenMP** looked at initially, but rejected:
  - Difficult to use well: not accessible for users.
  - Cannot take advantage of C++-specific knowledge (locks).
  - Nesting intra-module use within tasks very restricted.
- **TBB** chosen for schedule-level tasks and task queues. Intra-module use can be **TBB** high-level constructs (`parallel_for` or `parallel_reduce`) or low-level **TBB** tasks.

Detail: choice of parallelism level.

- Due to generally low module count in **Intensity Frontier** experiments, reward / effort for module-level parallelism for trigger-path modules deemed insufficient, especially since bulk of work combined in just a few modules that must run serially.
- Event-level (multiple schedules) being implemented. First version to process events simultaneously only within a subrun.
- **TBB** use within modules automatically available.
- Module-level parallelism being considered specifically for non-thread-safe analyzers and output modules in lieu of event-level parallelism.

Detail

- Services types: **`LEGACY`**, **`GLOBAL`** and **`PER_SCHEDULE`**. Using **`LEGACY`** services automatically precludes multi-schedule operation. **`GLOBAL`** implies full thread-safety.
- `ServiceHandle` to return correct service regardless of scope without extra arguments using **TBB** task tools.
- Signals / slots for service callbacks: global and per-schedule signals.
- Looking at **HDF5** for parallel I/O for the long term.

Detail

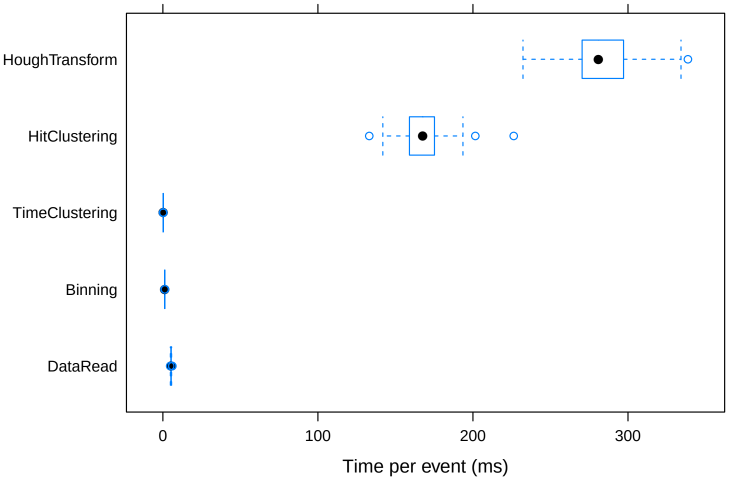
- No initial changes to data model. `RunFragment` and `SubRunFragment` concepts likely to allow event-level parallelism across subrun and run boundaries.
- No changes to ancillary data / metadata.
- Parallel resources managed by configuration option / **TBB**: generally, `services.scheduler.num_threads` >> `services.scheduler.num_schedules`.

Studies: Track-finding — intro.

- **NOνA** simulated far detector data, 5 ms events ($n_{hit} \sim 90$ K).
- 2D Hough-like transform algorithm.
- Results shown for $y - z$ -view only (about 2/3 of hits in each event).
- Six stages:
  1. construct time histogram of hits;
  2. identify time-based clusters;
  3. assemble hit lists for each cluster;
  4. produce (ρ, θ) for hit combinations;
  5. find peaks in transform space;
  6. extract track parameters from peaks.
- Implement stages 1 – 4 in various ways: serial, TBB (with variants).

Studies: Track-finding — serial baseline.

- All results shown for 4× AMD 6128, "Magny Cours," no NUMA control.

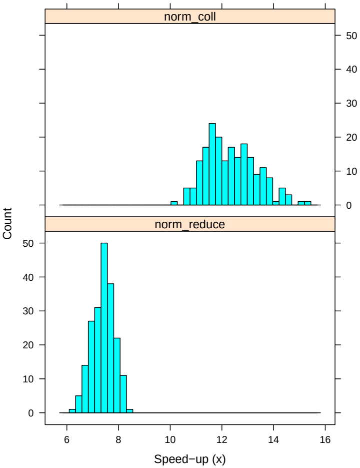


Time for serial algorithm.

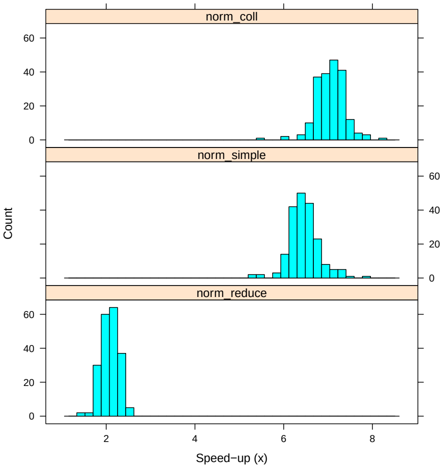
Studies: Track-finding — experiment.

- TBB, `struct-of-arrays` data structures:
  1. `parallel_for`: $\sim$ serial.
  2. `parallel_reduce`, use of `std::set`: $\sim 45\times$ slower than serial.
  3. `parallel_reduce`: $\approx 7\times$ faster than serial.
  4. `parallel_reduce`: $\approx 2\times$ faster than serial;
     `parallel_for` (array offset calculation): $\approx 6.5\times$ faster than serial;
     `task`: no change from `parallel_for`.
- TBB, `array-of-structs` data structures:
  1. Slightly slower than `struct-of-arrays`.
  2. No change from `struct-of-arrays`.
  3. `parallel_for`: $\approx 12\times$ faster than serial.
  4. $\approx 7\times$ faster than serial.

Studies: Track-finding — results.



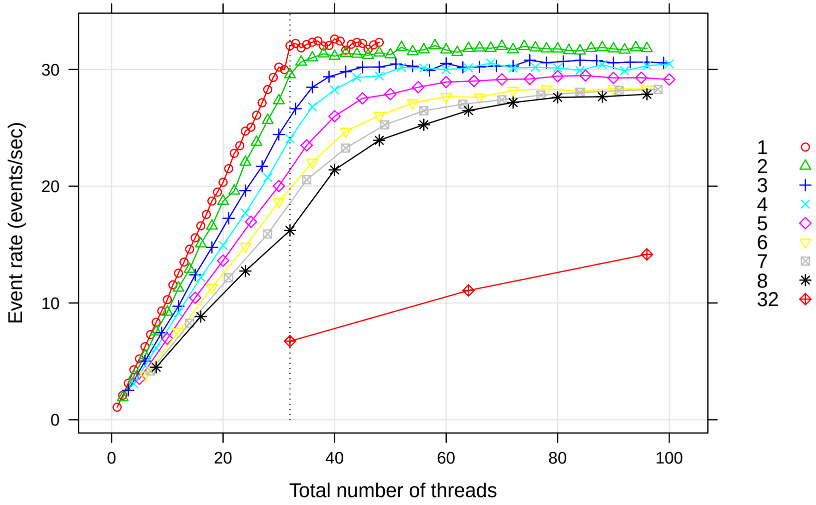
Hit clustering.



Hough transform.

Studies: Track-finding — scaling, interference.

- Scaling: full-machine event rate by total thread count, grouped by threads per process. **TBB**, `struct-of-arrays`.



Studies: Track-finding — lessons learned.

- Not in a regime where computation is so intense that data proximity is more important than algorithm complexity.
- If an algorithm is expressible simply using high-level constructs (*e.g.* `parallel_for`) using `task` is unlikely to help.
- At least with these algorithms, super-linear performance was not achieved. Scaling / interference studies show that single thread processes will achieve about the best per-machine performance (subject to jitter). Moving to 2 threads for the algorithms smooths out the jitter.

Backup slides . . .

Studies: Track-finding.

- Scaling: full-machine event rate by number of processes, grouped by threads per process. **TBB**, **`struct-of-arrays`**.