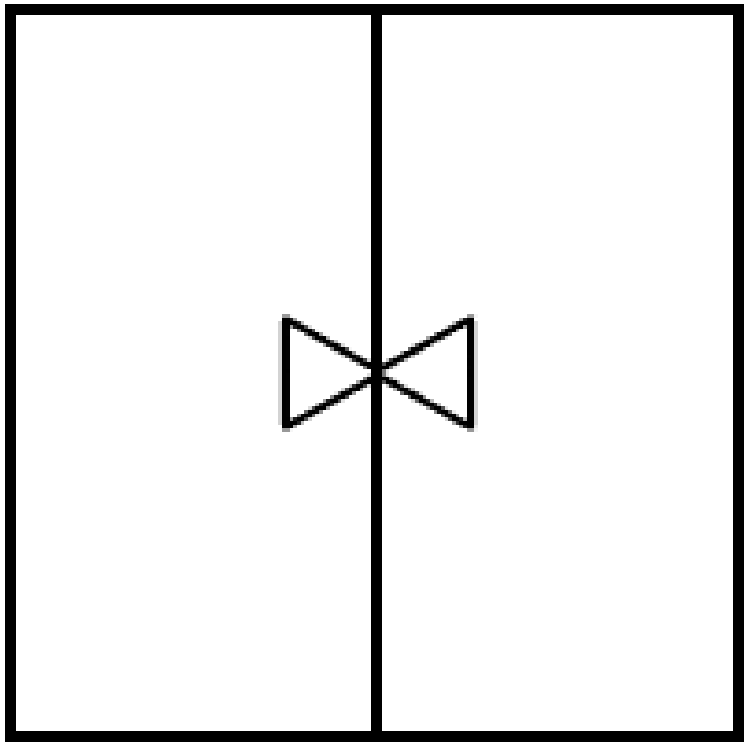
Common software needs and opportunities for HPCs



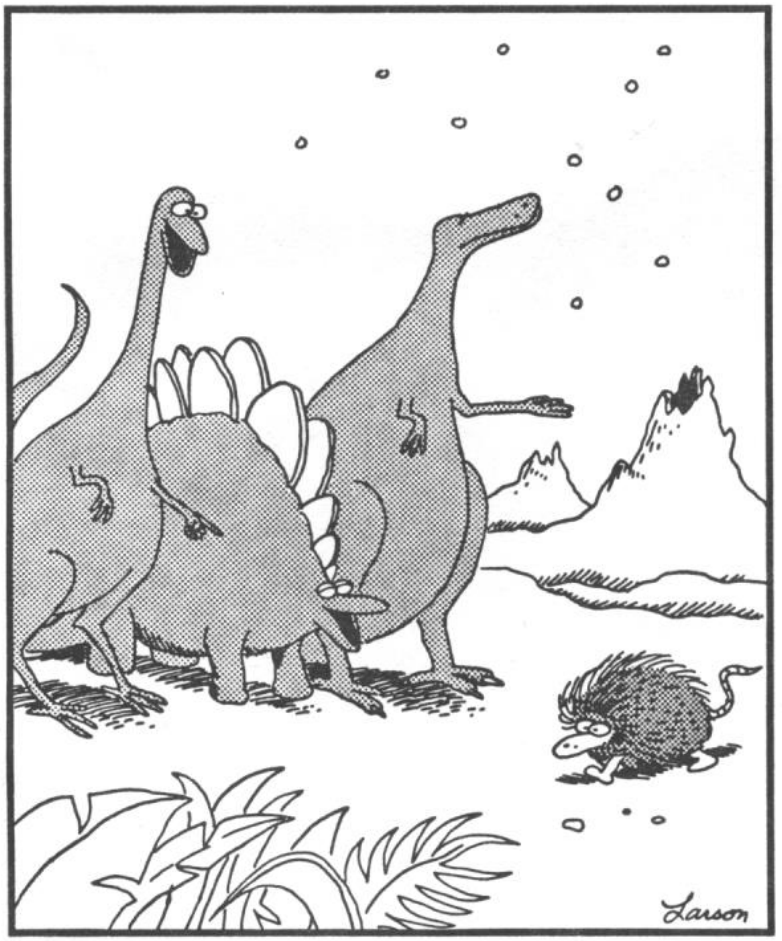
(A man wearing a bow tie giving a slightly too-long ‘elevator speech’)

Tom LeCompte

High Energy Physics Division
Argonne National Laboratory

The Landscape

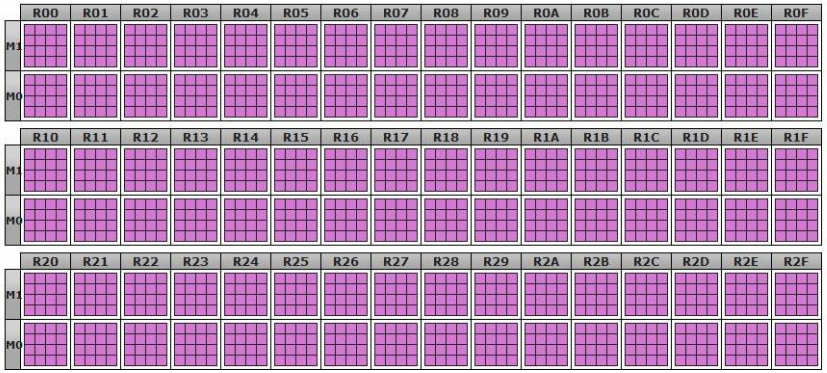
- HEP Code evolved in a world where CPU was expensive and memory was cheap.
- This is changing.
- This is changing whether you are talking about
  - Supercomputers
  - Farms of commodity PCs
  - GPUs
  - Xeon Phis
  - Etc…
- This fact, coupled with the fact that we don't have the resources to rewrite our code from scratch forms **the central problem** for HEP computing.



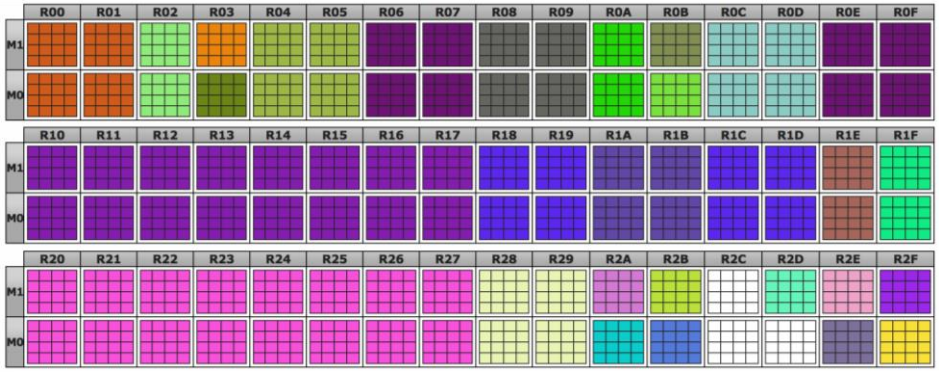
This talk will discuss one (of several) facets that address this.

A Success Story

- This is a picture of Alpgen running on Mira
  - Mira is the #5 supercomputer: a BlueGene/Q with 786,432 cores
- We are using the entire machine here – running 1,572,864 simultaneous instances of Alpgen
  - I don't know exactly the job card we used here:
    - These could be events that the Grid couldn't make (highly filtered Z+5 jets)
    - These could be ordinary events where we are supplementing Grid production
  - Credit: Taylor Childers (ANL/HEP), Tom Uram (ANL/ALCF), Doug Benjamin (Duke)
- While we were running, this was maybe 10x-15x the ATLAS grid



Mira when we were using it



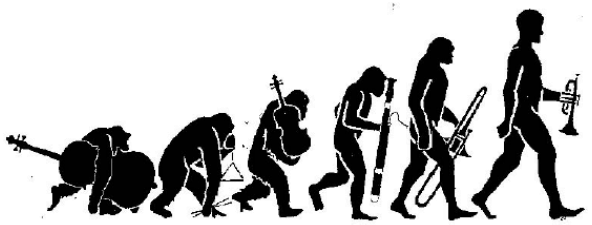
Mira at another time

How Did This Happen?

- Not by accident and not all at once
  - I/O had to be substantially reworked
  - Memory usage had to be reworked – optimization is complex
    - One can use memory to run more processes per node
    - Or one can use memory to buffer data
    - But you can't use the same byte of memory twice
  - A long and ultimately unnecessary detour into the world of random number seeds
    - We use up seeds 1.5 million times faster than a single threaded Alpgen would
  - We are running ~5-ish faster than when we started
    - Not algorithmic improvements – just using the processors more efficiently
    - We have reason to suspect at least another x2 is possible
  - All without breaking (or irreversibly forking) the code base

- I could tell you similar stories about Sherpa and Geant
  - But not in six minutes
  - And those stories are less far along

Evolving Our Code



- Memory/CPU ratio
  - A BlueGene/Q node has 16 GB of memory for 16 CPUs and 64 hardware threads
  - Not too different from a Xeon Phi with 16 GB for 61 single-threaded cores
  - Our biggest payoff on memory optimization is from releasing memory when it is no longer needed
    - Example: once a PDF set is selected, the others no longer need to be kept in memory
    - This is architecture independent
    - This requires understanding of the code – i.e. code authors are in a better position to do this than "HPC optimizers"
- Code quality
  - We found a bug in Alpgen
    - Array out of bounds – crashes on BG/Q, does not crash on Intel
  - Those of you who have been through this before (32 →64 bit, or Vax → Unix) know that changing platforms exposes problems, problems that can now be fixed.

“Without Breaking The Code”

- Rationalizing the I/O
  - Example: using a RAMdisk for output until the very end, where the output is then consolidated.
  - Cordon off the new I/O behind preprocessor directives
  - Obviously, the actual code here is application specific – but the ideas behind it are more general
- Sensible use of PDFs
  - We could use preprocessor directives here as well, but…
  - Nobody needs to switch PDFs mid-run
  - Shouldn’t this be rolled back into the main code base?

In the process of getting code to run efficiently, we are seeing patterns emerge.
Many of these patterns will help with other memory constrained architectures.
Some may even help with today’s single-threaded machines.
How do we best share this knowledge with each other?

Summary

- Widely used HEP code runs on supercomputers
  - And not just x86-based supercomputers with a lot of memory
- These provide us with a platform with many of the challenges that we will face eventually – but with an immediate payoff (delivering computing to our communities) in addition to the long-term (evolving our code)
- Making progress requires collaboration between machine experts and software experts (us). Fortunately, they are really nice guys, so this is not so scary.
- **Certain common patterns are emerging – we shouldn't all rediscover them ourselves**