HPC for Computational Astrophysics: Looking Forward

Ann Almgren

Center for Computational Sciences and Engineering
Lawrence Berkeley National Laboratory
asalmgren@lbl.gov

July 26, 2011

Introduction: Trends in HPC

Power and cost constraints $\longrightarrow$ a significant shift in architectural design for next generation systems

Exascale architectures – likely characteristics

- Much higher concurrency (more cores) in
  - low-power (core may do less – fewer operations, slower clock speed)
  - many-core (cores per chip, cores per node)
  - possibly heterogeneous (not all cores are equal)

  nodes
- Much lower memory per core
- Reduction in relative I/O system performance
- Higher failure rates for components
- Less of a guarantee of reproducibility

Implications

- FLOPS will matter less
- Memory and data movement will matter more
- What does this mean for reliability / reproducibility (aka verification)?

Note that this doesn't apply just to exascale machines – this will extend to your laptop as well!

What Does All This Do For You?

- Today's "Hero" run becomes part of tomorrow's parameter study – use a suite of calculations to understand uncertainties, explore parameter space...

- Tomorrow's "Hero" run has higher resolution, more physics (with more fidelity) – hopefully answers some questions we can't even hope to answer today.

Bad science done faster is still bad science.

But, with any luck, more numbers, faster, leads to more insight and understanding.

Algorithm and Code Changes

Pure MPI – or even MPI + OpenMP – will no longer be standard approach. Efficient codes are going to be harder to write.

Next generation machines will need algorithms that

- run on more cores (more concurrency)
- move data less (more locality with reduced synchronization)
- use less memory / FLOP

Current approaches to algorithm design are based considering how well we can approximate the physics of the problem with a given number of FLOPS.

This is fundamentally at odds with new architectures where FLOPS are (almost) free and the real costs are storage and movement of data. We also have to deal with non-uniform memory access (NUMA) – this will need to be built into the algorithm.

We need to rethink how we model different physical processes within the context of machines that reflect a different character

Machine reliability will also be a problem – algorithms may need to be designed to recover from faults.

Example of Algorithm Changes

Monte Carlo methods for particles are embarassingly parallel.

Put N particles on each processor and let them go.

If the particles need information from an underlying grid, and the entire grid fits on each processor, then the method scales perfectly.

As memory per core gets smaller and the amount of information you want on the grid gets larger, the entire domain will no longer fit on each processor.

This fundamentally alters the parallelization of the algorithm – now particles must be associated with the processor that owns the grid they're in – particles now must move between processors, changing the previous parallelization paradigm.

Programming models

Current programming models are not adequate for the next generation architectures

Example of a programming model: Fortran with MPI.

```
do j = 1, ny
do i = 1, nx

uout(i,j) = uin(i,j) + dt * (flux_x(i,j) - flux_x(i+1,j)) + dt * (flux_y(i,j) - flux_y(i,j+1) )

end do
end do
```

This tells the processor what operations to do.

But this loops says nothing about how to access memory during the operations.

- We express codes in terms of FLOPS and let the compiler figure out the data movement
- Non-uniform memory access is already an issue but programmers can't easily control data layout

We will need new types of programming models that allow us to more explicitly control data motion and data layout.

How Do These Changes Affect the Way you Run Code

- Codes are going to be harder to *run* efficiently - we will need a way to understand what to expect in terms of actual performance on a given architecture in order to understand how to correctly write the algorithm and/or run the code.

- I/O is going to be a bigger issue – no more run now / analyze later.

I/O Issues

We will need to change how we think about I/O – even doing it “right” won’t be enough.

Useful paradigm is that of a laboratory experiment – what is measured and how to measure it is part of the process

We need to decide in advance what we want to measure in a given simulation and build that into the code in advance.

Bad news:

- loss of convenience – no longer store the data and analyze at your leisure
- less ability to use vis/analysis as exploratory tools – you need to know what you’re looking for
- if you forget a diagnostic, you need to reproduce the entire run, not just the vis session

Good news:

- The vis and analytics folks are working hard to make this all work... in situ analysis techniques, new algorithms, new ways of looking at data, etc.

Reproducibilty and Reliability Issues

We have already seen that multiple runs of the same code on the same machine rarely take the same compute time.

Something different happens each time – which cores on which nodes are used, how the messages are routed, etc. Even if the actual lines of code stay fixed in time, the environment in which the code runs does not – changes in the software stack can affect results.

The more nonlinear the phenomenon the more dramatically the results may be affected – imagine a change in one summation causing one more V-cycle causing the temperature to just cross a threshold to cause a star to detonate ...

The idea of “reproducible results”, which is essential to our definition of predictive science, may have to be redefined.