First steps in optimising Cosmos++: A C++ MPI code for simulating black holes

Damon McDougall

TACC, UT Austin, USA

19th September 2017

Overview

1. This list
2. Scientific/ECSS Context
3. Examples of serial performance improvement
4. C++ software design considerations
5. Wrap up and go home

Context

- There's a debate about how to represent viscosity in the modelling of black hole accretion disks.
    - Parameterise stress?
    - Turbulence driven by magneto-rotational instability? (Balbus & Hawley, 1991, 1998).
- Parameterisation is cheap. Full MHD not cheap. The difference between the two is not well understood (Pessah et al., 2008 has a quantification of the difference)
- Only one study that made direct, quantitative comparisons of global viscous and MHD simulations using the same numerical code (Reynolds & Miller, 2009; ONeill et al., 2009).
- The PI's goal is to extend that work by including, for the first time in global accretion disk simulations, a fully relativistic treatment of the Navier-Stokes equations.
- Captures the effects of strong gravity close to the black hole.

The code

The code implements:

- A discretisation of a relativistic treatment of the Navier-Stokes equations (a PDE)
- A time-stepper
- Some finite-element-type discretisation with an adaptive mesh treatment
- A nonlinear optimisation scheme inside the time loop
- Pure MPI and C++

The situation

- The PI had already made an attempt at hybridising their code
- Software doesn't scale too well on `stampede2`
- We recommend a hybrid OpenMP/MPI solution
- 1–2 MPI tasks per node, each owning 32–64 OpenMP threads
- Contiguous stride-1 memory access
- Take advantage of the 512b wide vector registers
- Intel compiler: `-O3 -xMIC-AVX512`
- Use MCDRAM in cache mode (`development` or `normal` queues on `stampede2`)
- TACC advice is here: `https://portal.tacc.utexas.edu/user-guides/stampede2#bestpractices`

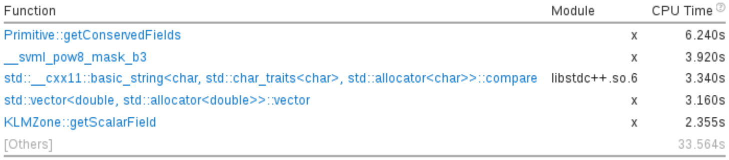
How do I get started?

- I am given a code and I have no idea what it does
- Assessing software quality can be tricky, but these help:
    - Some kind of build system (no pitchforks, please)
    - Some kind of test harness (with tests, as well)
- Now I can profile, and get a feel for where code is spending its time
- We have vtune, which is great for single-node profiling
- Hotspots analysis: why is the PI's initial attempt at OpenMP not scaling?
    - Majority of time spent in `kmp_(fork|join)`? Your loop isn't really doing anything

Back to serial

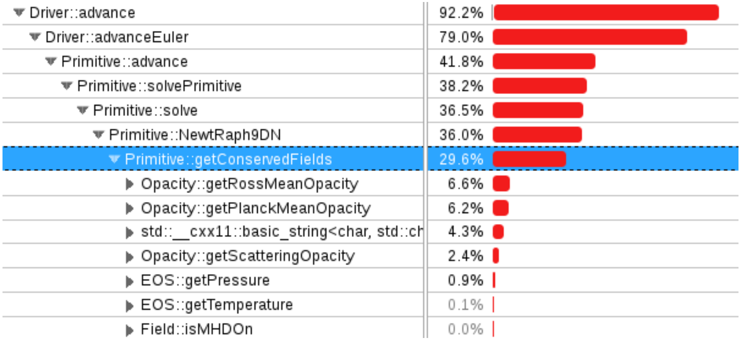
- Before deciding to get good hybrid code execution, how does the serial execution look?

Hotspots summary



Function	Module	CPU Time
Primitive::getConservedFields	x	6.240s
__svml_pow8_mask_b3	x	3.920s
std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::compare	libstdc++.so.6	3.340s
std::vector<double, std::allocator<double>>::vector	x	3.160s
KLMZone::getScalarField	x	2.355s
[Others]		33.564s

Hotspots breakdown



Driver::advance	92.2%
Driver::advanceEuler	79.0%
Primitive::advance	41.8%
Primitive::solvePrimitive	38.2%
Primitive::solve	36.5%
Primitive::NewtRaph9DN	36.0%
Primitive::getConservedFields	29.6%
Opacity::getRossMeanOpacity	6.6%
Opacity::getPlanckMeanOpacity	6.2%
std::__cxx11::basic_string<char, std::ch	4.3%
Opacity::getScatteringOpacity	2.4%
EOS::getPressure	0.9%
EOS::getTemperature	0.1%
Field::isMHDOn	0.0%

The code

```
void OpacPow::getOpacity(double &density,
                         double &temperature,
                         vector<double> &bincenter, // non-const
                         vector<double> &opacn)
{
  // ... some stuff here

  for(int n = 0; n < numBins; n++) {
    double nu = bincenter[n]; // bincenter is only read
    opacn[n] = tmp * pow(nu/mNu0, mNuExp);
  }
}
```

The code

"Use `const` whenever possible" –Scott Meyers

I've gotten a lot of value out of Scott Meyers's books:

- Effective C++
- More Effective C++
- Effective STL
- Effective Modern C++

Making the parameter a `const` reference does two things:

1. It tells me that the function *cannot* change it;
2. It tells the compiler to yell at you if you try.

The code

Now to the caller:

```
void Opacity::getRossMeanOpacity(double &rho,
                                 double &temp,
                                 double &opacn)
{
  vector<double> opactmp(mNumBins, 0.0); // alloc memory...
  mRossMeanAbsorption->getOpacity(rho,temp,mBinCenter,opactmp);
  opacn = opactmp[0]; // ...but we only need the first element
}
```

We can overload `getOpacity` to implement a version that just returns the first element.

The code

```
void PoleAxisBC::applyBC(int numFields,
    vector<vector<vector<double> > > &field, string fieldName) {
  // ... stuff here
  vector<vector<vector<int> > > oppZoneIndex; // alloc...
  oppZoneIndex = mField.getOppZoneIndex(); // ...then copy
  for(int i = 0; i < bclist.size(); i++) {
    if(oppZoneIndex[zoneID][n].size() > 0) { // read oppZoneIndex
      for(int iface = 0; iface < numFaces; iface++) {
        if(iface == 0 || iface == 1)
          for(int ifield = 0; ifield < numFields; ifield++)
            field[ifield][iface][zoneID] =
                 field[ifield][iface][oppZoneIndex[zoneID][n][0]];
                 // read oppZoneIndex
      }
    }
  }
}
```

I had to modify this code to fit it on the slide.

The code

- A noncontiguous `int` datastructure...
  - ...that we copy
  - ...and only read
- It looks as though we only need a reference to `oppZoneIndex`
- Overload `getOppZoneIndex` to return a reference
  - Elides allocation
  - Elides copy
- Re-juggle the `oppZoneIndex` datastructure to be contigous in memory
  - This is *hard*
  - Subsequent `for` loop becomes stride-1 access
- The PI ended up removing this function entirely and implementing a different approach

The code

```
void Opacity::getOpacity(string type, string name,
                         vector<double> &rho,
                         vector<double> &temp,
                         vector<vector<double> > &opacn) {
  if(opacn.size() != mNumBins) {
    for(int n = 0; n < opacn.size(); n++) opacn[n].clear();
    opacn.clear();
    opacn.resize(mNumBins,vector<double>(numZones,0.0));
  } else if(opacn[0].size() != numZones) {
     for(int n = 0; n < mNumBins; n++) {
         opacn[n].clear();
         opacn[n].resize(numZones, 0.0);
     }
  }

  // ...compute stuff here
  // write to opacn
  mAbsorption[iindx]->getOpacity(rho,temp,mBinCenter,opacn);
}
```

The code

- We've already gone over the non-contiguous issue.
- Remember this is being called inside an optimisation loop.
- If we're overwriting `opacn` every time (testing needed), then there's no need to set everything to zero
- Is there an *a priori* bound on the size of the array?
    - ... we might be able to do something better.

The code

```
void Primitive::NewtRaph9D(/* params here */ ) {
  // ... some setup
  vector<double> opaca, opacs;

  if(mOpacity->isRossMeanOn()) {
      mOpacity->getOpacity("absorption", "rossMean", rho, temp,
           opaca);
      if(opaca.size() > 0) kapaR = opaca[0];
      opaca.clear();
  } else {
      kapaR = kapaP;
  }
  mOpacity->getOpacity("scattering", rho, temp, opacs);
  if(opacs.size() > 0) kaps = opacs[0];
  opacs.clear();

  // ... some more stuff
}
```

More words of wisdom from Meyers

"Use objects to manage resources." –Scott Meyers

The code

- Maybe move the management of `opac(a|s)` to the `NewtRaph9D` ctor.
- One-time memory setup, as opposed to managing memory inside a nested function call
- Memory deallocation handled by the dtor
- Use objects to manage resources

One more example

- Last example is an easy one

The code

```cpp
void OpacPow::getOpacity(double &density,
                         double &temperature,
                         vector<double> &bincenter, // non-const
                         vector<double> &opacn)
{
  // ... some stuff here

  for(int n = 0; n < numBins; n++) {
    double nu = bincenter[n]; // bincenter is only read
    opacn[n] = tmp * pow(nu/mNu0, mNuExp);
  }
}
```

The code

- What are the arguments to `pow`?
- In my gdb session, they were numbers like $2^{3.5}$
- Which can be re-written as $2 \times 2 \times 2 \times \sqrt{2}$
- AVX512 has a vectorised version of `sqrt` and `exp`
    - But not `pow` (as far as I'm aware)
    - I'm sure someone in the audience can correct me

The results

- I've thrown a lot of information at you, but what are the results of this?
- 30% speedup on stampede2
- 3x speedup on PI's laptop
- I really didn't have to do too much
    - Explain to the PI what's happening
    - The PI (and their students) learn something
    - They implemented pretty much all of the speedup

The results

- Your mileage may vary with the advice here
- I certainly make assumptions when proposing a re-design
    - Those assumptions may be unsatisfiable
- Communicate those assumptions with the PI
    - The PI may decide a different approach is better
- You won't get anywhere if you don't play
- One of the things I learned is how to communicate at the right level