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?

• 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
Before deciding to get good hybrid code execution, how does the serial execution look?
## Hotspots summary

<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Primitive::getConservedFields</code></td>
<td>x</td>
<td>6.240s</td>
</tr>
<tr>
<td><code>__svml_pow8_mask_b3</code></td>
<td>x</td>
<td>3.920s</td>
</tr>
<tr>
<td><code>std::__cxx11::basic_string&lt;char, std::char_traits&lt;char&gt;, std::allocator&lt;char&gt; &gt;::compare</code></td>
<td>libstdc++.so.6</td>
<td>3.340s</td>
</tr>
<tr>
<td><code>std::vector&lt;double, std::allocator&lt;double&gt; &gt;::vector</code></td>
<td>x</td>
<td>3.160s</td>
</tr>
<tr>
<td><code>KLMZone::getScalarField</code></td>
<td>x</td>
<td>2.355s</td>
</tr>
<tr>
<td>[Others]</td>
<td></td>
<td>33.564s</td>
</tr>
</tbody>
</table>
### Hotspots breakdown

<table>
<thead>
<tr>
<th>Category</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver::advance</td>
<td>92.2%</td>
</tr>
<tr>
<td>Driver::advanceEuler</td>
<td>79.0%</td>
</tr>
<tr>
<td>Primitive::advance</td>
<td>41.8%</td>
</tr>
<tr>
<td>Primitive::solvePrimitive</td>
<td>38.2%</td>
</tr>
<tr>
<td>Primitive::solve</td>
<td>36.5%</td>
</tr>
<tr>
<td>Primitive::NewtRaph9DN</td>
<td>36.0%</td>
</tr>
<tr>
<td>Primitive::getConservedFields</td>
<td>29.6%</td>
</tr>
<tr>
<td>Opaque::getRossMeanOpacity</td>
<td>6.6%</td>
</tr>
<tr>
<td>Opaque::getPlanckMeanOpacity</td>
<td>6.2%</td>
</tr>
<tr>
<td>std::__cxx11::basic_string&lt;char, std::char_traits&lt;char&gt;</td>
<td>4.3%</td>
</tr>
<tr>
<td>Opaque::getScatteringOpacity</td>
<td>2.4%</td>
</tr>
<tr>
<td>EOS::getPressure</td>
<td>0.9%</td>
</tr>
<tr>
<td>EOS::getTemperature</td>
<td>0.1%</td>
</tr>
<tr>
<td>Field::isMHDO</td>
<td>0.0%</td>
</tr>
</tbody>
</table>
The code

```c++
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.
Now to the caller:

```cpp
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.
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

```cpp
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 $\text{opacn}$ every time (testing needed), then there’s no need to set everything to zero.
- Is there an \textit{a priori} bound on the size of the array?
  - ... we might be able to do something better.
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 NewtRaph9Dctor.
- 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);
    }
}
```
• 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