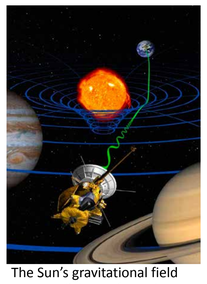
Simulating Black Holes with CUDA

Adam G.M. Lewis[1,2], Abdul H. Mroué[2], and Harald P. Pfeiffer[2]
[1]*Department of Physics, 60 St. George Street, University of Toronto, Toronto, ON M5S 3H8, Canada*
[2]*Canadian Institute for Theoretical Astrophysics, 60 St. George Street, University of Toronto, Toronto, ON M5S 3H8, Canada*

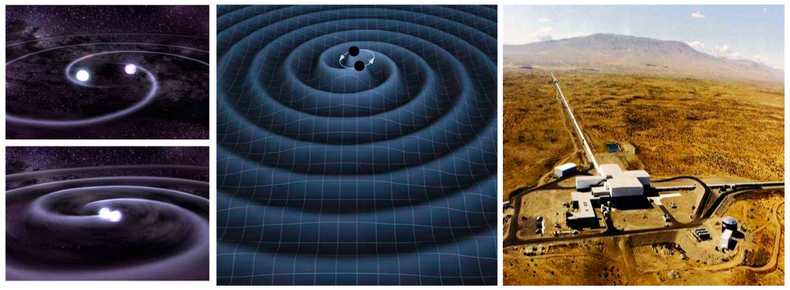
This decade will see the first detections of gravitational waves: ripples in spacetime produced strongly by collisions of dense objects like black holes. It will then be possible to study such events through careful comparison of their gravitational radiation against predictions generated through simulations. These simulations are computationally very expensive, requiring 10,000's of FLOPS per grid point per time step. Using NVIDIA's CUDA framework, we have developed techniques to automatically port our black hole code to GPUs. We have also manually optimized certain key routines, which have sped up by 10-40 times in response.

A new astronomy

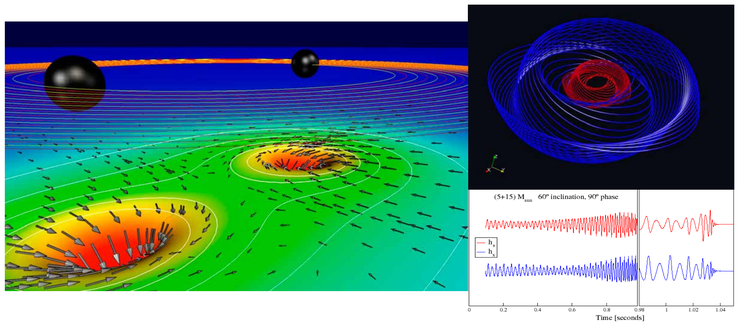
Contemporary physics describes gravity using the general theory of relativity. Mass is thought to bend space and time much as a billiard ball would bend a stretched rubber sheet. These bends curve the masses towards one another.


The Sun's gravitational field curves a light ray from the Cassini science probe [1].

As a direct consequence, orbiting masses create ripples in spacetime which spiral outwards as "gravitational radiation". The first experiment sensitive enough to detect these, Advanced LIGO (aLIGO), should be fully operational within the next 3-5 years.


Left [2], centre [3]: gravitational waves from colliding stars; right [4]: LIGO gravitational wave observatory in Washington State.

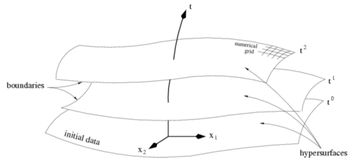
By calculating the shapes of gravitational waves predicted by relativity to issue from physically realistic objects like merging black holes, then comparing these shapes against real observations, we will be able to use aLIGO like a telescope to find where, when, and how mergers happen. This will rapidly advance scientific understanding of compact stars.


Output of numerical relativity simulations. Left: still from simulated black hole merger. Spatial (Ricci) curvature is visualized as colour and height, and paths of infalling particles as arrows; Top right: full trajectories of merging black holes plus Bottom right: predicted gravitational waveform.

Extracting accurate waveforms from general relativity requires a set of 10 coupled, nonlinear partial differential equations called the "Einstein field equations" to be solved with few or no simplifying assumptions. This is computationally very expensive, typically requiring tens of thousands of FLOPS per grid point per time step. But much of the calculation may be written in terms of matrix operations – perfect for acceleration on the GPU.

Spectral Einstein Code (SpEC)

We simulate gravity using Spectral Einstein Code (SpEC) [5]. Taking as input the curvature of space according to some observer at an early time, SpEC makes discrete steps in that observer's time co-ordinate and evolves the slices according to Einstein's equations.



Spatial slices at various timesteps [6].

SpEC:

- Divides space into grids called "domains".
- Performs Fourier-like transforms to map data in each domain to "spectra" with convenient properties.
- This greatly improves accuracy and rate of convergence.

The spectral mapping also allows many operations (esp. differentiation) to be written as matrix multiplications. GPU acceleration is thus easy in principle, but the code's length and complexity (500 000 lines) raise practical challenges.

Many calculations are performed on a class called **DataMesh**, storing one double per grid point. To simplify development, operations on DataMeshes are written as **expression templates** (line (*) in example below).

GPU porting strategy

Streamline memory synchronization

- Extend DataMesh class to copy between host and device only when data is requested.
- MemoryHandler retains memory on device, providing temporary memory *without* cudaMalloc/cudaFree calls.

Automate CUDA translation

- "CodeWriter" outputs equivalent CUDA kernel for each expression template.

Hand-code key routines

- Runtime dominated by 10 feasibly-hand-coded processes.

```
class DataMesh;

void RaiseIndex(std::vector<DataMesh>& W,
                const std::vector<DataMesh>& V,
                const std::vector<std::vector<DataMesh> >& M,
                const int dim) {
    for (int k=0; k<dim; ++k) {
        W[k]=0.;
        for (int i=0; i<dim; ++i)
            W[k] += g[k][i]*V[i]; (*)
    }
}
```

```
#DataMesh.hpp

DataMesh& DataMesh::operator+=(template_type_representing_AB& op) {
    f_0000(this->DataOnDevice(), this->Size(),
    op.LHS.DataOnDevice(), op.RHS.DataOnDevice));

#DataMesh.cu

__global__ f_0000(double* out, const in sz,
                  const double* A, cont double* B) {
    int idx=...;
    if(idx<sx) {
        out[idx] += A[idx] * B[idx];
    }
}
```

Preliminary results

Overall 1.63x speedup obtained for simple test cases.

Hand-coded ports have yielded more dramatic results:

FirstDeriv:

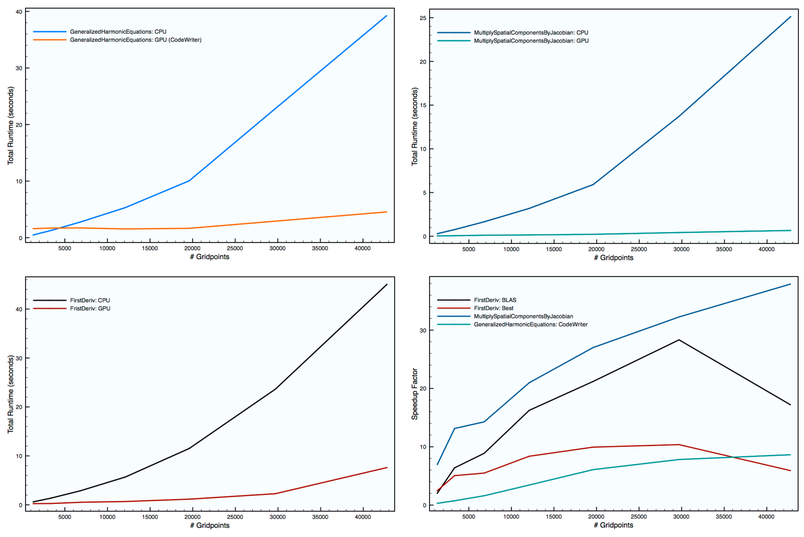
- Compute first derivative of a tensor w.r.t. given full set of co-ordinates.

MultiplySpatialComponentsByJacobian:

- Perform co-ordinate transformation on spatial part of a given tensor.

GeneralizedHarmonicEquations:

- Solve Einstein equations at a time step.



Total time consumed during simulation of a single black hole on CPU (Intel Xeon 2.0 GHz) and GPU (NVIDIA Tesla M2090) as a function of grid size by Top left: GeneralizedHarmonicEquations; Top right: MultiplySpatialComponentsByJacobian; and Bottom left: FirstDeriv. Bottom right: GPU vs. CPU speedup factor as a function of grid size.

References and Image Credits

1. NASA, JPL Caltech.
2. NASA/Dana Berry, Sky Works Digital.
3. K. Thorne (Caltech) and T. Carnahan (NASA GSFC).
4. LIGO Scientific Collaboration, http://www.ligo.org/multimedia/gallery/lho.php.
5. http://www.black-holes.org/SpEC.html
6. H. Dimmelmeier, http://www.mpa-garching.mpg.de/rel_hydro/core_collapse_in_cfc_plus/index.shtml