# Binary Black Hole Grand Challenge

**Black hole collisions have 2 major interest to numerical relativist**

1. Black hole collisions should be strong generators of gravitational radiation. LIGO(Laser Interferometric Gravitational Wave Observatory) should be able to detect this radiation by the end of the century.
2. This is an extremely difficult problem for numerical relativist. We will make use of the most powerful computers and the sophistication of AMR. It should provide us with the first "general" solution of Einstein Equations.

Groups involved

- Physics
  - **Austin**, **Cornell**, **Illinois**, **North Carolina**, **Northwestern**, **Penn State**, **Pittsburgh**,
- Computer Science

– **Austin**, **Illinois Syracuse**

# 4 dimensional spacetime into 3+1

- The four dimensional spacetime is split into 3 dimensional hypersurfaces which evolve in time.
- Breaking spacetime up this way gives us 10 non-linear PDE's to solve
- These equations are divided into two classes.
  - 4 Initial Value Equations (Elliptic)
  - 6 Coupled Hyperbolic equations (Hyperbolic)

One of the most important components in the solution to the evolution equation is the choice of gauges.

- The choice of gauge is the choice of coordinate systems.
- Some gauges can allow all the computational grids points to be "sucked" inside a black hole, where as other gauge choices can allow the hole to move on the computational domain.

**NOTE:** In Classical Physics evolutions are dictated by momentum variables and position variables. Numerical Relativity uses the extrinsic curvature, $K_{ij}$ to act like the momentum, and the metric $g_{ij}$ to act as the position.

# Initial Value Problem for 2 black holes

- The 4 equations break down into two sets of equations
  - The Momentum Constraint Equation ( vector equation)

$$\bar{D}_j \bar{K}^{ij} = 0. \tag{1}$$

  - The Hamiltonian Constraint Equation (scalar equation)

$$\bar{\nabla}^2 \psi = -\frac{1}{8} \psi^{-7} \bar{K}_{ij} \bar{K}^{ij} \tag{2}$$

The momentum equation can be solved with semi-analytical techniques for 2 black holes. The Hamiltonian Constraint equation solver solves this constraint equation along with the boundary conditions:

The boundary condition at infinity follows directly from asymptotic and conformal flatness:

$$\frac{\partial\psi}{\partial r} = \frac{1-\psi}{r}, \tag{3}$$

The boundary condition around the holes is, $(\alpha = 1, 2)$

$$n^i_\alpha \bar{D}_i \psi\Big|_{B_\alpha} = -\frac{\psi}{2r_\alpha}\Big|_{B_\alpha}, \tag{4}$$

where $r_\alpha = |\vec{x} - \vec{C}_\alpha|$ and $\vec{n}_\alpha = (\vec{x} - \vec{C}_\alpha)/r_\alpha$ (these are Euclidean operators in the flat background space).

The first 3D solutions were found by three separate groups

1. G. Cook (Cornell University)
2. S. Klasky, M. Choptuik, R. Matzner (University Texas at Austin)

3. M. Dubal, S. Oliveira, R. Matzner (University Texas at Austin)

**This problem has been solved**

# Methods to try to solve the Evolution Equations

- There are many methods to try to solve the evolution equations.
  - ADM formalisms
  - Hyperbolic formalisms

In order to solve these equations we must use AMR. This is because the problem varies over many length scales.

Most difficult part of the solution will be the boundary conditions.

The outer boundary problem will be solved by using a matching condition between a Cauchy(evolution) code, and a characteristic, or perturbation code.

The Cauchy code can "handle" the strongly nonlinear regime, and the Characteristic code can "handle" only the weakly nonlinear regime. The boundary condition is only known at infinity, thus we need two codes.

## Methods to try to solve the Evolution Equations

The inner boundary is a problem for several reasons:

1. We will excise the points inside the apparent horizon.
2. We would like to excise the points inside the event horizon. The problem is that the event horizon is only known at $t = \infty$. Thus, we need a way to exclude the singularity of the hole.
3. The apparent horizon is found by looking for *Trapped surfaces*. This breaks down into solving a very difficult elliptic equation.
4. When the hole moves in the computational domain, coordinate points that were previously excised will now appear inside the computational domain.

Many groups have explored special ways to difference around the holes (causal differencing). A new formalism, J. York, the "hyperbolic" formalism was developed such that relativist would not have to *worry* about the gauge freedom.

# The Texas, Syracuse T2 approach

Here we describe one such ADM method `T2` that Texas and Syracuse are working on.

The evolution equations that the `T2` solves are as follows:

$$\partial_t g_{ij} = -2\alpha K_{ij} + D_i\beta_j + D_j\beta_i \tag{5}$$

$$\partial_t K^i_j = \mathcal{L}_\beta K^i_j - D^i D_j \alpha + \alpha\left(R^i_j + K K^i_j\right) \tag{6}$$

where $g_{ij}$ is the 3-metric,

$K^i_j$ is the extrinsic curvature, $D_i$ is the covariant derivative computational with $g_{ij}$,

$\alpha$ is the lapse function, $\beta^i$ is the shift vector

and $\mathcal{L}_\beta$ is the Lie derivative along the vector field $\beta$.

$\mathcal{L}_\beta K^i_j$ is given by the following equation:

$$\mathcal{L}_\beta K^i_j = \beta^k D_k K^i_j - K^k_j D_k \beta^i + K^i_k D_j \beta^k \tag{7}$$

## Variable Storage

| Grid Function | Number of independent components |
|---|---|
| $g_{ij}$ | $6 \times 2$ independent components(two = n+1 timestep) |
| $g^{ij}$ | 6 independent components(required for various stages) |
| $K^i_j$ | $9 \times 2$ components (multiply by two for n+1 timestep) |
| $\Gamma^i_{jk}$ | 18 components (stored instead of $g_{ij,k}$) |
| $R_{ij}$ | 6 components (Raised when needed in eqn. (6)) |
| $\alpha$ | scalar function |
| $\beta^i$ | 3 components ($\beta_i$ can be obtained using $g_{ij}$) |

Hence the lower limit on required variables for the solution to the evolution is 64 grid variables. Plans are in the work for writing the equations using $K_{ij}$ thus eliminating 6 variables.

## Modules for the connection coefficients

The calculation of the connection coefficients can be broken up into several stages. This obviates the necessity for storing the first derivatives of the 3-metric which saves us 18 grid functions.

The connection coefficients are given by

$$\Gamma^i_{jk} = \frac{1}{2} g^{il} \left( g_{lj,k} + g_{lk,j} - g_{jk,l} \right) \tag{8}$$

The calculation can be broken up into three separate sets using 9 temporary variables.

There are 3 steps:

## 3 Steps for Connection coefficients

For the first term in eqn. (8)

$$\begin{aligned} \Gamma^i_{jk} &= \frac{1}{2} g^{il} g_{lj,k} \\ &= \frac{1}{2}\left(g^{i1} g_{1j,k} + g^{i2} g_{2j,k} + g^{i3} g_{3j,k}\right) \end{aligned} \quad (9)\ (10)$$

We call these parts C1-C3. For the second term in eqn. (8)

$$\begin{aligned} \Gamma^i_{jk} &= \Gamma^i_{jk} + \frac{1}{2} g^{il} g_{lk,j} \\ &= \Gamma^i_{jk} + \frac{1}{2}\left(g^{i1} g_{1k,j} + g^{i2} g_{2k,j} + g^{i3} g_{3k,j}\right) \end{aligned} \quad (11)\ (12)$$

We call these parts C4-C6. For the third term in eqn. (8)

$$\begin{aligned} \Gamma^i_{jk} &= \Gamma^i_{jk} - \frac{1}{2} g^{il} g_{jk,l} \\ &= \Gamma^i_{jk} - \frac{1}{2}\left(g^{i1} g_{jk,1} + g^{i2} g_{jk,2} + g^{i3} g_{jk,3}\right) \end{aligned} \quad (13)$$

We call these parts C7-C9.

## Ricci tensor

The calculation of the Ricci tensor

can be similarly broken up into several segments with the use of 9 temporary grid functions once more. The Ricci tensor is given by

$$R_{ij} = \Gamma^a_{ij,a} - \Gamma^a_{ia,j} + \Gamma^a_{ab}\Gamma^b_{ij} - \Gamma^a_{bj}\Gamma^b_{ia} \tag{14}$$

The evaluation of the Ricci tensor is broken as follows:

$$\begin{aligned} R_{ij} &= \Gamma^a_{ij,a} \\ &= \Gamma^1_{ij,1} + \Gamma^2_{ij,2} + \Gamma^3_{ij,3} \end{aligned} \tag{15}$$

$$\begin{aligned} R_{ij} &= R_{ij} - \Gamma^a_{ia,j} \\ &= R_{ij} - \left(\Gamma^1_{i1,j} + \Gamma^2_{i2,j} + \Gamma^3_{i3,j}\right) \end{aligned} \tag{16}$$

$$R_{ij} = R_{ij} + \Gamma^a_{ab}\Gamma^b_{ij} - \Gamma^a_{bj}\Gamma^b_{ia} \tag{17}$$

where in eqns. (14,15) terms like $\Gamma^1_{ij,1}$ and $\Gamma^1_{i1,j}$ are calculated using temporary arrays.

# Evolution equation for the 3-metric

$$\partial_t g_{ij} \;=\; -2\alpha K_{ij} + D_i\beta_j + D_j\beta_i. \tag{18}$$

The evaluation of the second and third terms is carried out by once again using 9 temporary grid functions. Recall that $\beta^i$, the contravariant form of the shift, is stored. Since $g_{ij}$ is compatible with $D_i$ we can rewrite $D_i\beta_j = g_{jk}D_i\beta^k$. Hence we are left with calculating

$$D_i\beta_j = g_{jk}\left(\partial_i\beta^k + \Gamma^k_{il}\beta^l\right). \tag{19}$$

Here once again we use a 9 grid functions to store the components of $\partial_i\beta^k$ first and then add $\Gamma^k_{il}\beta^l$ to form eqn. (19). Then evaluating eqn. (**??**) becomes a matter of using the temporary variables for $D_i\beta_j$.

Use second order finite differencing for time derivative. 5 Modules for these routines

```
C in evolve_g1
      call diff(nx,ny,nz,h,beta1,t11,'x',ml,dl,nl)
      call diff(nx,ny,nz,h,beta1,t12,'y',ml,dl,nl)
      call diff(nx,ny,nz,h,beta1,t13,'z',ml,dl,nl)
      call diff(nx,ny,nz,h,beta2,t21,'x',ml,dl,nl)
      call diff(nx,ny,nz,h,beta2,t22,'y',ml,dl,nl)
      call diff(nx,ny,nz,h,beta2,t23,'z',ml,dl,nl)
      call diff(nx,ny,nz,h,beta3,t31,'x',ml,dl,nl)
      call diff(nx,ny,nz,h,beta3,t32,'y',ml,dl,nl)
      call diff(nx,ny,nz,h,beta3,t33,'z',ml,dl,nl)
C in evolve_g2
      FORALL (i=1:3,j=1:3)
         t(i,j) = t(i,j)+c(1,i,j)*beta(1)+c(2,i,j)*beta(2)
     .            +c(3,i,j)*beta(3)
      END FORALL
      FORALL (i=1:3,j=1:3)
         t(i,j) = t(i,1) * g(1,j) + t(i,2) * g(2,j) + t(i,3) * g(3,j)
      END FORALL
C in evolve_g3
      FORALL (j=1:3,i=1:3,i<=j)
```

```
         t(i,j) = t(i,j) + t(j,i)
         t(j,i) = t(i,j)
      END FORALL
C in evolve_g4
      FORALL (i=1:3,j=1:3)
         t(i,j) = t(i,j) - 2.0d0*alpha(1) * ( g(1,i) * k(1,j) +
     .           g(2,i) * k(2,j) + g(3,i) * k(3,j)  )
      END FORALL
C in evolve_g5
      FORALL (j=1:3,i=1:3,i<=j)
         g(i,j,1) = 2.0d0*dt*t(i,j) + g(i,j,-1)
      END FORALL
```

## $\boxed{\text{Evolution equation for } K^i_j}$

$$\partial_t K^i_j \;=\; \mathcal{L}_\beta K^i_j - D^i D_j \alpha + \alpha \left( R^i_j + K K^i_j \right)$$

We break this equation up into 5 parts(5 modules)

```
C evolve_k1
      call diff(nx,ny,nz,h,alpha,temp1,'x',ml,dl,nl)
      call diff(nx,ny,nz,h,alpha,temp2,'y',ml,dl,nl)
      call diff(nx,ny,nz,h,alpha,temp3,'z',ml,dl,nl)
C evolve_k2
      call diff(nx,ny,nz,h,temp1,t11,'x',ml,dl,nl)
      call diff(nx,ny,nz,h,temp1,t12,'y',ml,dl,nl)
      call diff(nx,ny,nz,h,temp1,t13,'z',ml,dl,nl)
      call diff(nx,ny,nz,h,temp2,t22,'y',ml,dl,nl)
      call diff(nx,ny,nz,h,temp2,t23,'z',ml,dl,nl)
      call diff(nx,ny,nz,h,temp3,t33,'z',ml,dl,nl)
      t21 = t12
      t31 = t13
      t32 = t23
      FORALL (j=1:3,i=1:3,i<=j)
         t(i,j) = t(i,j) - c(1,i,j) *dalpha(1) -
     .              c(2,i,j)*dalpha(2) - c(3,i,j) * dalpha(3)
         t(j,i) = t(i,j)
      END FORALL
      call rais02i1()
```

```
FORALL (i=1:3,j=1:3)
   t(i,j) = -t(i,j)
END FORALL
```

Evolution equation for $K^i_j$ continued

```
! Evolve K3
*     Lie_beta(K^1_1)
      call diff(nx,ny,nz,h,k11,temp1,'x',ml,dl,nl)
      call diff(nx,ny,nz,h,k11,temp2,'y',ml,dl,nl)
      call diff(nx,ny,nz,h,k11,temp3,'z',ml,dl,nl)
      t11 = t11 + beta1 * temp1 + beta2*temp2+beta3*temp3

      call diff(nx,ny,nz,h,beta1,temp1,'x',ml,dl,nl)
      call diff(nx,ny,nz,h,beta1,temp2,'y',ml,dl,nl)
      call diff(nx,ny,nz,h,beta1,temp3,'z',ml,dl,nl)
      t11 = t11 - k11 * temp1 - k21 * temp2 - k31 * temp3

      call diff(nx,ny,nz,h,beta1,temp1,'y',ml,dl,nl)
      call diff(nx,ny,nz,h,beta2,temp2,'y',ml,dl,nl)
      call diff(nx,ny,nz,h,beta3,temp3,'y',ml,dl,nl)
      t11 = t11 + k11 * temp1 + k12 * temp2 + k13 * temp3
C Evolve K4
      trace_k(1) = k11 + k22 + k33
      FORALL (i=1:3,j=1:3)
         t(i,j) = t(i,j) + alpha(1) *(
```

```
     .              gup(i,1) * r(1,j) + gup(i,2) * r(2,j) +
     .              gup(i,3) * r(3,j) + trace_k(1) * k(i,j) )
      END FORALL
C Evolve K5
      FORALL (i=1:3,j=1:3)
         k(i,j,1) = 2.0d0*dt*t(i,j) + k(i,j,-1)
      END FORALL
```

# Derivative routines, Vertex Centered

Spatial derivative routines

contain routines for Inner boundary, Outer boundary and interior. Time derivatives, $O(h^2)$ use a 3 level leap frog scheme. All derivative routines are second order for the first derivative in $x, y, z$ directions.

To handle the hole, 2 non-distributed lists are created, `m(nl),d(nl)` where `m(l)` contains the i,j,k location of the "hole" points and `d(l)` contains either (1,-1,0) indicating either forward, backwards or centered.

The derivatives around the hole are determined to second order. Second derivatives, *for example* $f_{xx}$*, are only first order accurate.* The interior second derivatives are second order accurate. Routine looks as follows:

```
      subroutine diff(h,f,df,dir,m,d,nl)
      use params
      implicit none
      real*8 h
      real*8, dimension(nx,ny,nz) :: f,df
      real*8, dimension(3*nl)     :: d
      integer, dimension(3*nl)    :: m
      character dir
!HPF$ PROCESSORS PROC(nproc)
!HPF$ DISTRIBUTE f *(BLOCK,*,*) ONTO *PROC
!HPF$ ALIGN(:,:,:) WITH *f(:,:,:)         :: df
      call diff_int(h,f,df,dir)
      call diff_ob(h,f,df,dir)
      if (nl.gt.1) call diff_ib(h,f,df,dir,m,d,nl)
      return
      end
```

```fortran
      subroutine diffx_ib(h,f,df,m,d,nl)
      use params
      implicit none
      integer nl
      real*8, dimension(nx,ny,nz) :: f,df
      real*8, dimension(3*nl)     :: d
      integer, dimension(3*nl)    :: m
      real*8 h,h2
      integer i,j,k,l

      h2 = 0.5d0 / h

      do l = 1 , nl
         i  = m(l)
         j  = m(nl+l)
         k  = m(2*nl+l)
         d1 = d(l)
         d2 = d(nl+l)
         d3 = d(2*nl+l)
         da = abs(d1)
         df(i,j,k) = h2*(d1*(-3.0d0*f(i,j,k)+4.0d0*f(i+int(d1),j,k)
     $                        -      f(i+2*int(d1),j,k)) +
```

```
$                (1.0d0-da)*(f(i+1,j,k)-f(i-1,j,k) ) )
 end do
 return
 end
```

```
      subroutine diffx_ob(h,f,df)
      use params
      implicit none
      real*8, dimension(nx,ny,nz) :: f,df
      real*8  h,h2,hi
      integer i,j,k
!HPF$ PROCESSORS PROC(nproc)
!HPF$ DISTRIBUTE f *(BLOCK,*,*) ONTO *PROC
!HPF$ ALIGN(:,:,:) WITH *f(:,:,:)         :: df
      hi = 1.0d0/h
      h2 = 0.5d0 / h
      FORALL (k=1:nz,j=1:ny)
         df(1,j,k) = h2 * ( -3.0d0*f(1,j,k)
     .             + 4.0d0*f(1+1,j,k) - f(1+2,j,k) )
         df(nx,j,k) = h2 * (  3.0d0*f(nx,j,k)
     .              - 4.0d0*f(nx-1,j,k) + f(nx-2,j,k) )
      END FORALL
      FORALL (i=2:nx-1,k=1:nz)
            df(i,1,k) = h2 *(f(i+1,1,k)-f(i-1,1,k) )
            df(i,ny,k) = h2 *(f(i+1,ny,k)-f(i-1,ny,k) )
      END FORALL
      FORALL (i=2:nx-1,j=1:ny)
```

```
      df(i,j,1) = h2 *(f(i+1,j,1)-f(i-1,j,1) )
      df(i,j,nz) = h2 *(f(i+1,j,nz)-f(i-1,j,nz) )
END FORALL

return
end
```

# Future Plans

- Moving the hole on the domain, use some sort of Alcubierre and Schutz method(implicit scheme using ADI). ( Huq & Matzner)
- Apparent Horizon solver written in a Data Parallel language. (Huq, Klasky & Matzner)
- Causal differencing around the inner boundary.
- Stability of methods around the inner boundary is being looked at by
- AMR in `DAGH` and `HPF` : Haupt,Huq,Klasky,Parashar Correll