Goals of the Alliance

- Physics Goals
  - Provide gravitational waveforms for LIGO.
  - Evolve Einstein's equations for two holes through the coalescence.
- CS Goals
  - Develop a problem solving environment for Einstein's equations
    - Includes environment on Parallel Computers.
    - Includes Adaptive Mesh Refinements.
  - Collaborative visualization techniques for AMR.
- Products
  - A Parallel Adaptive Computational Infrastructure (DAGH), HPF.
  - Initial Value Solvers.
  - Outer Boundary Solvers.
  - 3D evolution codes.
  - Physics to understand the coordinate conditions.

Computational Infrastructure.

- Computational resources requirements
  - Current uni-grid, uni-processor versions of the code take 10 Cray T-90 seconds per iteration (for a 65^3) run. Implementing all other physics with this code will make the code take approximately 120 seconds/iteration.
  - Typical runs to obtain physics requires approximately 2000/iterations at this resolution, but we must work with at least twice this resolution for accurate waveform extraction, i.e. total T90 time estimate = (120 sec)(8 twice the resolution)(4 CFL condition)(2000 iterations) = 88.9 T90 days, (or about 22 years on a Pentium Pro 200).

Memory requirements

- Memory Requirement for uni-grid codes
  - To get accurate waveform extraction, for stable evolution codes, our memory requirement is (unigrid codes):

    $8(100)((1/32)^{-1}(2)(32))^3 = 6871$ Gigabytes, where (8=bytes/word, 100=total number of 3d grid functions, 1/32 = mesh spacing, +,- 32=outer boundary placement relative to the center of the domain(0,0,0).

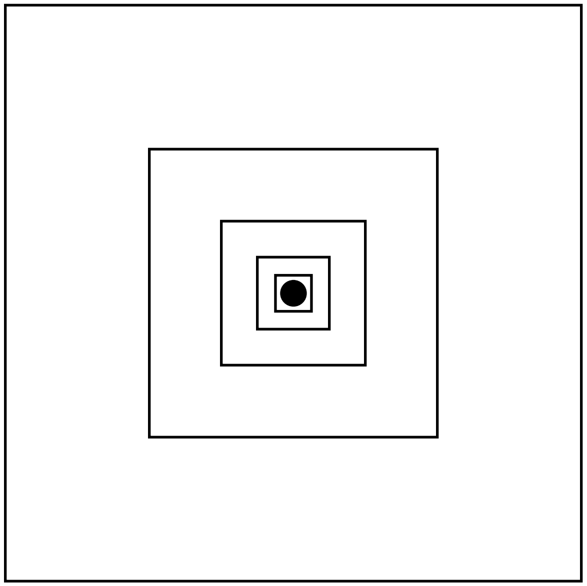
Memory requirements (continued)

•Memory Requirement for AMR

- •If we use Adaptive Mesh Refinement, our memory requirements can be greatly reduced.

  $$8(100)(5)(128)^3 = 8\ \text{Gigabytes},$$

  where we can use 5 grids with 128^3 points per grid, as shown on the figure to the left. The grids will dynamically adapt to the equations, using a truncation error estimate.



ADM Equations

3+1 Splitting of Einstein's equations

- We decompose the four-dimensional covariant version of Einstein's equations so that it is possible to pose them as a Cauchy evolution problem.

$$
\begin{aligned}
G_{00} &= 8\pi T_{00} \\
G_{0i} &= 8\pi T_{0i} \\
G_{ij} &= 8\pi T_{ij}
\end{aligned}
$$

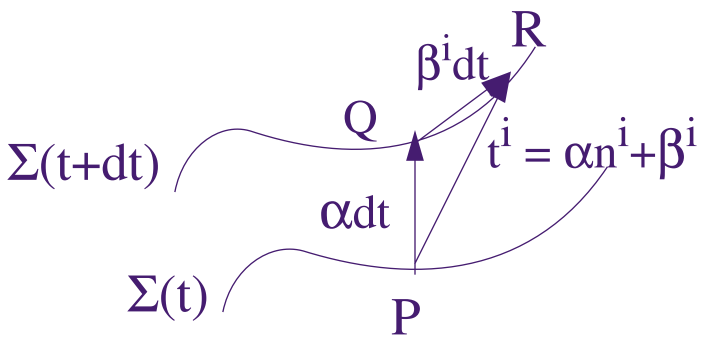
- A relationship between the components of the metric of spacetime and those of the spatial 3-metric must now be defined.
- We can look at the Arnowitt-Desser-Misner (ADM) form of the metric to see this.

$$ds^2 = -\alpha^2 dt^2 + \gamma_{ij}(dx^i + \beta^i dt)(dx^j + \beta^j dt)$$

where the functions α and β^i are called the lapse and shift functions, and γ_{ij} is the 3-metric.

ADM Equations (continued)

- α determines the lapse in proper time and β^i describes the shift in the spatial coordinates, and t^i are the tangents to a coordinate observer who in moving through the spacetime moves in a timelike direction by αdt, and in a spacelike direction by β^i.



- The extrinsic curvature is a spacelike object which determines the contraction and shear of the normals to $\Sigma(t)$.
- The evolution equations **we solve** are

$$\partial_t g_{ij} - \beta^k \partial_k g_{ij} = -2\alpha K_{ij} + g_{lj}\partial_i \beta^l + g_{il}\partial_j \beta^l$$

$$\partial_t K_{ij} - \beta^k \partial_k K_{ij} = -\partial_i \partial_j \alpha + \Gamma^k_{ij}\partial_k \alpha + \alpha(R_{ij} + K K_{ij} - 2K^l_i K_{lj}) + K_{lj}\partial_i \beta^l + K_{il}\partial_j \beta^l$$

ADM Equations (continued)

- We have the ability to use either a 3-level non-staggered leapfrog scheme or a 2-level semi-iterative Crank Nicholson scheme to evolve the equations forward in time.
- In order to evolve the equations in a stable manner, we make use of causal differencing.

Difference Equations

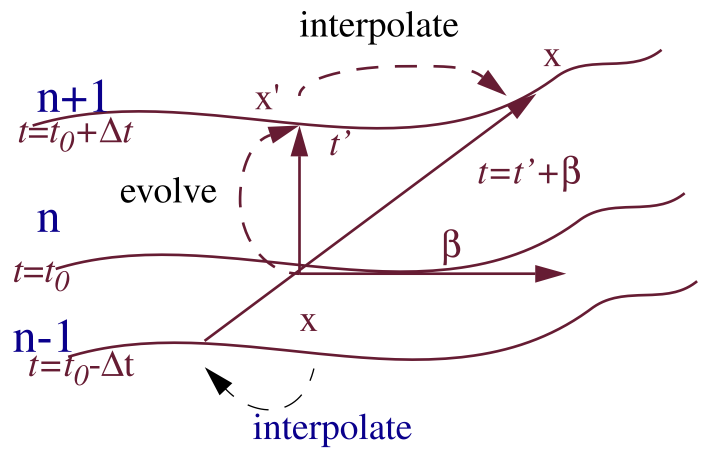
- Vertex centered derivatives on a uniform mesh.
- 3-level non-staggered leapfrog scheme.
- 2-level semi-iterative Crank Nicholson scheme.
- We iterate until the residuals are small.

Causal Differencing

- Causal Differencing handles the $\beta^k\partial_k$ terms in the evolution equations (the advective terms).
- β^k moves the world lines of particles in the coordinates and changes the coordinate basis vectors, i.e. the shift "shifts" the coordinates.

Our implementation (for a 3 level system) is as follows (t=proper time, t' is time along unit normal)

- Evaluate the RHS ${}^{(g)}\mathfrak{R}^{(n)}_{ij}$, ${}^{(K)}\mathfrak{R}^{(n)}_{ij}$ at time level n $(t=t_0)$.
- Compute the shifted coordinates at levels $n\text{-}1(t=t_0-\Delta t)$ and $n(t=t_0)$.



Causal Differencing (continued)

- **Interpolate** RHS's to the new shifted coordinates *n*.
- **Interpolate** g_{ij} and K_{ij} at the shifted coordinates on the *n-1* time level.
- Carry out a leapfrog update.
- **To Summarize**, we use causal differencing to obtain stable evolutions. Causal differencing involves interpolating grid functions everywhere in the domain. The complication arises when we must interpolate around regions which are cut out of the computational domain, *i.e.* around holes.

Apparent Horizon boundary Conditions

- To achieve long-term stability we must cut out the singularity from the computational domain. (Cosmic censorship!)
- When we do not cut the hole out of the computational domain, we must use a special shift vector so the code does not crash do to the singularity of the black hole.
  - This was the "traditional method" until the BBH GC. (Smarr etc.)
  - Grids points get "sucked" into the singularity, causing all of these codes to crash.
- Problems when we don't cut out the hole.
- We can't find the event horizon, so we find the apparent horizon (the innermost "trapped" surface, found by solving a elliptic equation.
  - We must know the future in order to determine the location of the event horizon.

Computational components/modules

- Modular structure of the code
  - It allows to mix C++ and Fortran (driving routines in C++ and computational modules in Fortran)
- Many of Fortran90 features makes implementation and maintenance of the ADM code easier:
  - Array syntax and Fortran90 intrisics
  - Derived types to implement tensor operations:

```
type GridFunction
real*8, dimension(:,:,:), pointer :: d
end type GridFunction
type(GridFunction), dimension(3,3) :: g,k

do i=1,3
 do j=1,3
   g(i,j)%d=funct(k(i,j)%d)
 enddo
enddo
```

HPF and HPF2

- HPF
  - HPF addresses data parallel paradigm (explicitly expressed using Fortran 90 syntax, FORALL and compiler directives, e.g., INDEPENDENT).
  - Data mapping is specified using compiler directives (BLOCK, CYCLIC).
  - Global name space: compiler detects need for interprocessor communications, and actually inserts calls to the runtime support.
  - In general this model is not sufficient: EXTRINSIC procedures.
- HPF2 extensions
  - Limited support for task parallelism.
  - Generalized BLOCK and user defined data distributions.
  - Better support for distribution of user defined (“derived”) data types.

DAGH

- Data distribution optimized for AMR
  - Data are organized into a hierarchy of grid functions defined on the computational domain.
  - Grid functions are distributed over available processors in blocks of a variable size; the blocks of data are referred to as "grid components".
  - Grid functions corresponding to fine level of refinements are aligned with the base grid so that restrictions and prolongations never require any interprocessor communication.
  - Optimal load balance is achieved by adjusting the size of grid components assigned to processors; the algorithm used takes into account the actual structure of the grid hierarchy after each refinement and is based on the concept of a self-similar space-filling curves.

DAGH (continued)

- Data parallel
  - Expressed explicitly by the user using DAGH “forall” macro. Unlike HPF, DAGH forall means: “for all grid components”, that is, if possible, carry out computations on all grid components concurrently.
- Local name space: user develops node code
  - User is responsible for interprocessor communication.
  - The user is responsible for global-to-local array indices translation.
  - No access to data from other grid components (augmented with ghost regions) without explicit calls to the DAGH communication library or MPI; note: the DAGH communication library is not accessible from Fortran.

DAGH (continued)

- DAGH communication library
  - Optimized for the specific DAGH data distribution.
  - Supports prefetching of ghost (“shadow”) regions necessary for any stencil based computations.
  - Supports many of Fortran90/HPF features, such as reductions or moving subregions defined by bounding boxes.
  - In general it is not sufficient: user can add custom developed routines with calls to a generic MPI library.

DAGH (continued)

- DAGH system provides a set of high level abstractions that can be used to implement an existing sequential unigrid PDE solver in the AMR regime on multiprocessor systems.
  - User defines a Cartesian computational domain (“base grid”), and specifies parameters for creating a grid hierarchy. The list of parameters includes maximum levels of refinement, refinement ratio, switch between cell-centered and vertex-centered grids, and more.
  - User provides routines to update the grid and boundary conditions, and routines that provide data for testing regridding criteria (typically truncation error estimate).
  - DAGH provides a library of routines for intergrid data movement (prolongation and restriction), and the user is allowed to replace it by his/her own that better matches the problem in hand.
  - For parallel execution, the user specifies the size of ghost regions.
- DAGH is implemented in C++, and the user routines can be written in F90 (particularly important when porting an existing code into DAGH).

Conversion of an existing Fortran 90 implementation to DAGH

- The main program and drivers must be rewritten in C++.
  - Data structures are defined in C++.
  - Communication library is implemented in C++ and accepts C++ data structures.
- Fortran computational modules may stay (almost) without modifications.
  - When running in parallel mode any fortran routine is executed independently for each grid component; that is, the Fortran routines implements the node code: no access to data outside the grid component, no global indices.

F90 to DAGH: communication

- Requests for off-processor data (calls to the DAGH communication library) must be inserted to C++ drivers.
  - Updates of the ghost regions.
  - Reductions.
  - Operations on whole grids (thus not a single grid component).
  - If necessary, MPI based routines must be developed (equivalent of HPF extrinsics).

F90 to DAGH: Passing data

- Passing data from C++ to Fortran.
  - DAGH provides two macros that simplify passing arrays to Fortran: FBA which extracts the actual shape of the grid component and its lower and upper bounds; and FDA which extracts the grid function data from the grid hierarchy.
  - The DAGH grid hierarchy must be “flattened”: since Fortran routine operates on a single grid component, one level of refinement at a time, there is no need to preserve DAGH data structure when passing data to Fortran. A set of “plain” arrays is passed instead.
- Derived types used in the ADM code.
  - Fortran 90 routines expect tensors as the input; therefore a “interface” routines has to be added. These routines accept array arguments from C++ drivers, define the tensor derived types, initialize them (pointer associations) and call the original fortran computational routines.

Automatic translation

- The burden of generating the calling sequences is removed by using custom developed `perl` scripts for this purpose.
  - Annotations in the form of structured comments, similar in concept to the HPF directives, makes automatic translation to DAGH easier. For example, requests for updates of the ghost regions (foreign in the Fortran sequential code) can be persistently marked that way.

IMPLEMENTATION

1=easy 2=medium 3=difficult to implement

Difficulties inherent to the BBH problem	**HPF**	**HPF2**	**DAGH**	**Comments**
1. Linear wave problem. No “inner boundaries”	1	1	1	lapse=1, shift=0, no holes.
2. Apparent horizon finder	1	1	1	Expand spherical harmonics.

Difficulties inherent to the BBH problem	HPF	HPF2	DAGH	Comments
3a. Spherical “inner boundary” in Cartesian coordinates.	2	1	2	Introduces load imbalance. Best performance when boundary points are organized into a list (1-dimensional array). One sided differencing require larger stencil leading to more expensive communications.
3b. Moving “inner boundary”.	2	1	2	If any techniques for improvement of the load balance are used, the data must be remapped every time step. Additional computations (extrapolation) add to load imbalance.

Difficulties inherent to the BBH problem	HPF	HPF2	DAGH	Comments
4. Outer boundary - wave extraction	1	1	3	Requires coexistence of 2d and 3d meshes and data movement (interpolations) between them.
5. Adaptive Mesh Refinement.	3	2	1	**HPF**: Support for derived types needed in HPF. Impossible to achieve the data distributions in HPF as good as in DAGH. **DAGH:** Outer boundary must be implemented as "DAGH extrinsics".

1. Linear waves

Linear waves in the BBH GC code have been run in both DAGH and HPF. Both implementations allow for very good parallel speedup, since all calculations are regular.

SP2	33	65	129	T3d	33	65
1	170	-	-		208	-
2	104	541			113	
4	65	303			63	
8	55	220			44	220
16	38	113	607		38	138
36		75	286			
64			222			

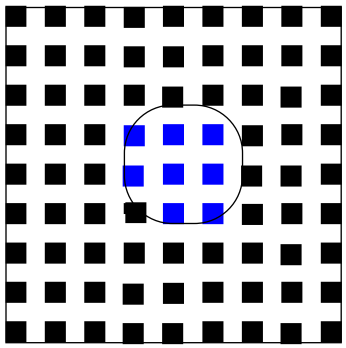
Timing of HPF run with pghpf (version 3.02) using `(*,*,Block)` distribution for small problems, and `(*,Block,Block)` for larger problems(n=129).

2. Finding the Apparent Horizon

- We have two methods to solve this elliptic problem.
  - One method expands the terms of the equation of trapped surfaces into spherical harmonics.
  - The other method solves the elliptic problem using finite-differencing with a Newton shooting method and a conjugate gradient method.
- During the evolution, we must solve for the apparent horizon at each time step.
- NOTE: During a 2 black hole evolution, there will be two apparent horizons, up until they start to coalesce, and then there will be one apparent horizon. This shift from 2 to 1 apparent horizons can cause complications with the grid distribution on parallel computers.

3a. Black Holes (Inner Boundary Condition)

- Region inside the apparent horizon must be cut-off and special forward and backward differencing is used to determine the derivatives of the grid functions at the “boundary” lattice points.



- For a static hole, the inner boundary points are generated from placing a spherical boundary onto Cartesian coordinates.
- For a moving hole, the boundary points can be anywhere in the computational domain.
- For interacting holes, the shape of the apparent horizon cannot be approximated by a sphere.

Inner Boundary condition (continued)

- The simplest data parallel approach: introduce 3d mask to define the boundary surface and employ WHERE statement.
  - Very inefficient:
    - Additional storage.
    - Necessity to scan the 3d volume to identify points on a 2d surface.
    - In general, we obtain a very poor load balance.

Inner Boundary Conditions (continued)

•Fortran90 implementation:

•Remedy: use 1-dimensional list of points on the boundary:

```
do p = 1, nl  !loop on all of the points on the 1d list
     df(m(p),m(nl+p),m(2*nl+p))=
     h2*(d(p)*(3.0d0*f(m(p),m(nl+p),m(2*nl+p))+      &
     4.0d0*f(m(p)+int(d(p)),m(nl+p),m(2*nl+p))       &
     -f(m(p)+2*int(d(p)),m(nl+p),m(2*nl+p))) +       &
     (1.0d0-abs(d(p)))*(f(m(p)+1,m(nl+p),m(2*nl+p))- &
     f(m(p)-1,m(nl+p),m(2*nl+p)) ) )
end do
nl - # boundary points (=length of the list)
df - a grid function (3D array)
m  - list of indices of the boundary points (m(p)=i,
 m(nl+p)=j, m(2*nl+p)=k -> i,j,k indices
d  - defines stencil direction (+1 = forward diffenc-
 ing, -1 = backward differencing, 0 = centered differ-
 encing
```

•Very efficient on vector machines (Cray).

Inner Boundary Conditions (continued)

- DAGH implementation
    - Create a separate list for each grid component (locality!)
    - Define array of a size equal to number of grid components.
    - Each element of the array is a 1d array itself that comprise a list of boundary points within the grid component.
  - Advantages
    - The list arrays aligned with grid components: no interprocesor communication (once ghost regions are updated).
  - Disadvantages
    - Load balance problem not solved.

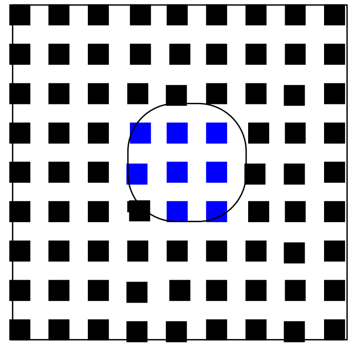
Inner Boundary Conditions (continued)

- HPF implementation
  - The idea of the distributed 1d list introduced in our DAGH implementation is difficult to implement directly in HPF, since it violates the global name space of HPF. However, it can be easily implemented as an extrinsic procedure.
  - Other solutions, such as declaring a global 1d array (the list of points) may lead to additional communications, since the block size of the grid functions and the block size of the list owned by a processor (assuming both are distributed blockwise) in general do not match.
  - The load balance can be improved using general block or indirect distribution introduced in HPF-2.

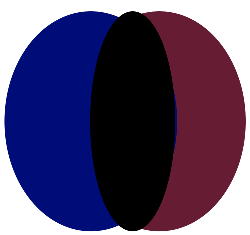
Inner Boundary Condition: Variable stencil size

- We use a fourth order operator to evaluate the derivatives around the hole, and these are forward and backwards differenced. Thus, the stencil size around the hole can be as large as 4.
    - the HPF-2 directive SHADOW allows to define a variable size of the ghost regions so that the very expensive communications needed for computations that involve stencil of size 4 can be easily restricted to the routines that impose the boundary conditions (a feature not available in DAGH).

3b. Moving Inner Boundary (A Moving Hole)



During the evolution, the black holes will move on the computational domain. Since the computations around the holes are more expensive, the distribution of the data must be redistributed in order to achieve “better” load balancing.



As the hole moves to the right (from the blue hole to the red hole), the blue portion of the hole is now in the computational domain (Points inside the hole are not in the computational domain). Thus, we must extrapolate to determine the grid functions at these points.

Moving Hole (Continued)

- DAGH implementation
  - We are ignoring the load balancing problems around the hole, although DAGH does provide a weight function for grid points, and this feature may be used.
  - DAGH does not have any special techniques to handle holes, so on processors that have many boundary points, a large slowdown is expected.
  - For non-moving holes, we can place the data so that communication cost will not be large around the holes, i.e. we can move the center of the hole such that one processor contains all of the hole points.

Moving Hole (continued)

- HPF implementation
  - When using 3d mask and WHERE construct, there is no difference for a static and moving black hole. The efficiency of the code (poor load balance) remains constant as the evolution of the hole proceeds.
  - Using extrinsic procedures in HPF allows it to reproduce DAGH performance.
  - As in case of the static hole, the load balance can be significantly improved in HPF-2 using generalized block or more general indirect distribution at the cost of additional computation to determine the optimal block size for each processor, and remapping of data (next neighbor communications).

4. Linear wave problem with matching outer boundary condition for gravitational wave extraction.

- Interior and exterior evolution
  - The ADM code evolves Einstein's equations on the 3d lattice (Cauchy problem).
  - Outer boundary lattice points from this evolution are interpolated to the points on the 2D (characteristic evolution) or 1D (perturbative approach) lattice.
  - Exterior evolution (2D or 1D) is used for the wave extraction
  - Values resulting from the exterior evolution are then used as the outer boundary values for the interior evolution.

Matching interior and exterior evolution

- DAGH implementation
  - Currently there is no built-in support for transferring data between 3D to 2D/1D data structures. Routines for that must be custom developed using MPI directly (“DAGH extrinsic procedures”).
- HPF implementation
  - HPF is a general purpose language, so it can handle this.
  - Extracting the boundary points from the 3D grids necessarily leads to the interprocessor communication.
  - Optimal data distribution for the outer evolution that minimize cost of data remapping (twice per iteration) remains to be determined.

5. Support for (parallel) AMR

Requires dynamical memory management; while a serial code can be implemented in Fortran77, Fortran90 and C++ offer much more elegant syntax.

- Implementation in HPF is very difficult. Two main reasons:
  - Lack of support for distributed components of derived types - needed to implement the dynamical data structures
  - Lack of support for irregular data distributions - necessary for load balance.
- HPF-2 extensions address these issue
  - No compilers available at this time.
- Dedicated system (DAGH) in C++ with MPI
  - Dedicated data structures.
  - Dedicated load balancer.
  - Can this experience help with developing HPF-2 code and/or compilers?