INSTITUTE OF PHYSICS PUBLISHING CLASSICAL AND QUANTUM GRAVITY

Class. Quantum Grav. **21** (2004) 743–766 PII: S0264-9381(04)66175-2

A fast apparent horizon finder for three-dimensional Cartesian grids in numerical relativity*

Jonathan Thornburg

Max-Planck-Institut für Gravitationsphysik, Albert-Einstein-Institut, Am Mühlenberg 1, D-14476 Golm, Germany

E-mail: jthorn@aei.mpg.de

Received 16 July 2003
Published 29 December 2003
Online at stacks.iop.org/CQG/21/743 (DOI: 10.1088/0264-9381/21/2/026)

Abstract

In $3+1$ numerical simulations of dynamic black-hole spacetimes, it is useful to be able to find the apparent horizon(s) (AH) in each slice of a time evolution. A number of AH finders are available, but they often take many minutes to run, so they are too slow to be practically usable at each time step. Here I present a new AH finder, AHFINDERDIRECT, which is very fast and accurate: at typical resolutions it takes only a few seconds to find an AH to $\sim 10^{-5}m$ accuracy on a GHz-class processor. I assume that an AH to be searched for is a Strahlkörper ('star-shaped region') with respect to some local origin, and so parametrize the AH shape by $r = h\,(\text{angle})$ for some single-valued function $h : S^2 \to \Re^+$. The AH equation then becomes a nonlinear elliptic PDE in h on S^2, whose coefficients are algebraic functions of g_{ij}, K_{ij}, and the Cartesian-coordinate spatial derivatives of g_{ij}. I discretize S^2 using six angular patches (one each in the neighbourhood of the $\pm x$, $\pm y$, and $\pm z$ axes) to avoid coordinate singularities, and finite difference the AH equation in the angular coordinates using fourth-order finite differencing. I solve the resulting system of nonlinear algebraic equations (for h at the angular grid points) by Newton's method, using a 'symbolic differentiation' technique to compute the Jacobian matrix. AHFINDERDIRECT is implemented as a thorn in the CACTUS computational toolkit, and is freely available by anonymous CVS checkout.

PACS numbers: 04.25.Dm, 02.70.Bf, 02.60.Cb

M This article features online multimedia enhancements

(Some figures in this article are in colour only in the electronic version)

* Appendix B on 'multiprocessor and parallelization issues' and appendix C on 'searching for the critical parameter of a 1-parameter initial data sequence' also appear in the preprint-archive version of this paper (gr-qc/0306056).

1. Introduction

In 3 + 1 numerical relativity, it is often useful to know the positions and shapes of any black holes in each slice. These are both key physics diagnostics, and potentially valuable in choosing the coordinate conditions in a numerical evolution. Moreover, black holes inevitably contain singularities, which may need to be excised from the computational domain [6, 44][1]. Since the event horizon can be determined only once the entire future development of the slice is known[2], i.e., only after a numerical evolution is done, in practice one usually uses the apparent horizon(s) as a working approximation which can be computed slice-by-slice while a numerical evolution is still ongoing. (Recall that an apparent horizon is always contained inside an event horizon, and they coincide if the spacetime is stationary [26].) Apparent horizons are also interesting due to their close relationship to isolated horizons, which have many useful properties (see [7, 22] and references therein).

There has thus been long-standing interest in algorithms and codes to find apparent horizons in numerically computed spacetimes (slices). Here I focus on the case where there are no continuous symmetries such as axisymmetry, and where the spatial grid is Cartesian. Many researchers have developed apparent horizon finding algorithms and codes for this case, for example [5, 10, 25, 27, 28, 30, 37, 41, 42, 45, 51]. However, with the exception of [41, 42][3], the existing numerical codes for apparent horizon finding are generally very slow, often taking several minutes to find each apparent horizon even on modern computers. This is a serious problem, since we would ideally like to find apparent horizons at each time step of a numerical evolution, and there may be tens of thousands of such time steps.

In this paper I describe a new numerical apparent horizon algorithm and code (based on a generalization of the algorithm and code I described previously for polar-spherical grids [48]) which is very fast: for typical resolutions it takes only a few seconds to find an AH, so it is practical to run it at every time step of a numerical evolution. This apparent horizon finder is also very accurate, typically finding apparent horizons to within a few tens of parts per million in coordinate position, with similar accuracies for derived quantities such as the apparent horizon area, irreducible mass, coordinate centroid, etc. This apparent horizon finder is implemented as a module ('thorn') AHFinderDirect in the Cactus computational toolkit [24][4], and is freely available (GNU GPL licensed) by anonymous CVS checkout.

In the main body of this paper I give a relatively high-level description of the algorithms used in AHFinderDirect; in the appendices I discuss various technical issues in more detail.

1.1. Notation

I generally follow the sign and notation conventions of [53]. In particular, I use the Penrose abstract-index notation, with indices *i–m* running over the (Cartesian) spatial coordinates $x^i \equiv (x, y, z)$ in a (spacelike) 3 + 1 slice. g_{ij} is the 3-metric in the slice, with associated covariant derivative operator ∇_i. K_{ij} is the extrinsic curvature of the slice (I use the sign convention of [54], not that of [53]) and $K \equiv K_i{}^i$ is its trace. η_{ij} is the flat 3-metric. Indices uvw run over generic angular coordinates $y^u \equiv (\rho, \sigma)$ on the apparent horizon surface. 'N-D' abbreviates 'N-dimensional'. In cases where the distinction is important, I use a prefix $^{(3)}$ to denote quantities defined on a 3D (three-dimensional) neighbourhood of the apparent horizon surface.

[1] For more recent work on this topic, see (for example) [2, 3, 11, 15].
[2] For numerical purposes the usual approximate development to a nearly stationary state suffices [4, 13, 21, 32].
[3] Schnetter [41, 42] has developed an apparent horizon finding algorithm and code quite similar to mine. His work and mine were done independently; neither of us learned of the others' work until our own was mostly complete.
[4] `http://www.cactuscode.org`.

Small-capital indices IJK label angular grid points on the apparent horizon surface, and $h[\mathrm{I}]$ is the evaluation of a grid function h at the grid point I. $\mathtt{D}_u$ and $\mathtt{D}_{uv}$ are finite difference molecules discretely approximating the angular partial derivatives ∂_u and ∂_{uv} respectively. If $\mathtt{m}$ is an index into a finite difference molecule $\mathtt{M}$, then $\mathtt{M}[\mathtt{m}]$ is an individual molecule coefficient, and $\mathtt{m} \in \mathtt{M}$ means that this coefficient is non-zero. Molecule indices may be obtained by subtracting grid point indices ($\mathtt{m} = \mathrm{J} - \mathrm{I}$), or correspondingly the sum of a grid point index and a molecule index gives a grid point index ($\mathrm{J} = \mathrm{I} + \mathtt{m}$).

2. Apparent horizons

Given a (spacelike) 3 + 1 slice, a 'marginally trapped surface' (MTS) is defined as a closed spacelike 2-surface in the slice, whose future-pointing outgoing null geodesics have zero expansion Θ. In terms of the usual 3 + 1 variables this condition becomes [55]

$$\Theta \equiv \nabla_i n^i + K_{ij} n^i n^j - K = 0 \tag{1}$$

where n^i is the outward-pointing unit normal to the surface.

An 'apparent horizon' (AH) is then defined as an outermost MTS in a slice (there may be multiple MTSs nested inside each other). In this paper I actually describe an algorithm and code for locating MTSs, but since the primary application will be the location of AHs, for convenience of exposition I refer to the MTSs as AHs.

As is common in AH finding, I parametrize the AH surface by first choosing a local coordinate origin x_0^i inside the AH, then assuming that the horizon is a 'Strahlkörper' ('ray body' or more commonly 'star-shaped region') about this point. A Strahlkörper is defined by Minkowski [43, p 108] as

> a region in n-D Euclidean space containing the origin and whose surface, as seen from the origin, exhibits only one point in any direction.

I take $y^u \equiv (\rho, \sigma)$ to be generic angular coordinates on the AH surface (or equivalently, on the unit 2-sphere S^2). Given these, I then define the AH shape by $r = h(\rho, \sigma)$, where $r \equiv \left[\sum_i \left(x^i - x_0^i\right)^2\right]^{1/2}$ is a radial coordinate around the local coordinate origin[5], and the 'AH shape function' $h : S^2 \to \Re^+$ is a single-valued function giving the radius of the AH surface as a function of angular position about the local coordinate origin.

3. Computing the expansion Θ (continuum)

To write the expansion Θ (and thus the AH equation (1)) explicitly in terms of this parametrization, i.e. in terms of h and its first and second angular derivatives, I first define a scalar function which vanishes on the AH surface and increases outwards, ${}^{(3)}F \equiv r - h(\rho, \sigma)$. I then define a (non-unit) outward-pointing normal covector to the AH surface as the gradient of this scalar function,

$$s_i \equiv {}^{(3)}s_i \equiv \nabla_i {}^{(3)}F \tag{2}$$

$$= \partial_i {}^{(3)}F \quad \text{since } F \text{ is a scalar} \tag{3}$$

$$= \partial_i r - \partial_i h \tag{4}$$

$$= \frac{x^i}{r} - X_i^u \partial_u h, \tag{5}$$

[5] Note that I define r to be the *flat-space* distance from x_0^i to x^i—there is no use of the 3-metric here.

where I define the coefficients $X^u_i \equiv \partial y^u / \partial x^i$. It is then straightforward to show that

$$\partial_i s_j = \frac{T_{ij}}{r^3} - X^u_{ij} \frac{\partial h}{\partial y^u} - X^u_i X^v_j \frac{\partial^2 h}{\partial y^u \partial y^v}, \tag{6a}$$

where

$$T_{ij} = \begin{cases} \sum_{k \neq i} (x^k)^2 & \text{if} \quad i = j \\ -x^i x^j & \text{if} \quad i \neq j, \end{cases} \tag{6b}$$

and where I also define the coefficients $X^u_{ij} \equiv \partial^2 y^u / \partial x^i \partial x^j$.

The outward-pointing unit normal to the AH surface is then

$$n^i = \frac{s^i}{\|s^k\|} = \frac{g^{ij} s_j}{(g^{k\ell} s_k s_\ell)^{1/2}}, \tag{7}$$

so the expansion Θ is given by

$$\Theta \equiv \nabla_i n^i + K_{ij} n^i n^j - K \tag{8}$$

$$= \partial_i n^i + (\partial_i \ln \sqrt{g}) n^i + K_{ij} n^i n^j - K \tag{9}$$

$$= \partial_i \frac{g^{ij} s_j}{(g^{k\ell} s_k s_\ell)^{1/2}} + (\partial_i \ln \sqrt{g}) \frac{g^{ij} s_j}{(g^{k\ell} s_k s_\ell)^{1/2}} + \frac{K^{ij} s_i s_j}{g^{k\ell} s_k s_\ell} - K \tag{10}$$

$$= \frac{A}{D^{3/2}} + \frac{B}{D^{1/2}} + \frac{C}{D} - K, \tag{11}$$

where

$$A = -(g^{ik} s_k)(g^{j\ell} s_\ell) \partial_i s_j - \tfrac{1}{2} (g^{ij} s_j)[(\partial_i g^{k\ell}) s_k s_\ell] \tag{12a}$$

$$B = (\partial_i g^{ij}) s_j + g^{ij} \partial_i s_j + (\partial_i \ln \sqrt{g})(g^{ij} s_j) \tag{12b}$$

$$C = K^{ij} s_i s_j \tag{12c}$$

$$D = g^{ij} s_i s_j. \tag{12d}$$

Setting $r = h$ in the definitions (5) and (6) and substituting into (11) and (12) gives Θ explicitly in terms of h and its first and second angular derivatives, so the AH equation (1) takes the form

$$\Theta \equiv \Theta(h, \partial_u h, \partial_{uv} h; g_{ij}, K_{ij}, \partial_k g_{ij}) = 0 \tag{13}$$

where the dependence on g_{ij}, K_{ij} and $\partial_k g_{ij}$ is implicit through their position dependence (this is discussed in detail in section 6.1).

4. Solving the apparent horizon equation

I view the AH equation (13) as an elliptic PDE for h on S^2, and discretize it using standard finite differencing methods: I introduce a total of N_{ang} angular grid points $\{(\rho_\text{I}, \sigma_\text{I})\}$ on S^2, and represent h and Θ by their values $\{h_\text{I}\}$ and $\{\Theta_\text{I}\}$ at these points. Approximating the angular derivatives $\partial_u h$ and $\partial_{uv} h$ by finite differencing, (13) then becomes a set of N_{ang} nonlinear algebraic equations $\{\Theta_\text{I} = 0\}$ for the N_{ang} $\{h_\text{I}\}$ values.

I solve these equations by Newton's method in N_{ang} dimensions. This in turn has several subparts:

- The actual Newton-method iteration algorithm.
- Computing the (discrete) expansion $\{\Theta_I\}$ given a (discrete) trial AH shape $\{h_I\}$.
- Computing the Jacobian matrix $\mathbf{J}_{IJ} \equiv d\Theta_I/dh_J$ given a (discrete) trial AH shape $\{h_I\}$.
- Solving the Newton-method updating equations $\mathbf{J} \cdot \delta h = -\Theta$.

I describe these in detail in the following sections.

5. Newton's method

The basic multidimensional Newton-method algorithm is well known (see, for example, [47, section 5.3]), but several refinements are necessary for a practical AH finder.

To make Newton's method converge more robustly if the initial guess is poor, and to limit divergence if the iteration does not converge, AHFinderDirect limits any single Newton step to have an ∞-norm over the angular grid which is no more than a specified maximum fraction (10% by default) of the mean horizon radius.

Much more sophisticated 'modified Newton' algorithms could be used to achieve faster or more robust convergence (e.g., [8, 9, 29, 36, 38–40]), but in practice this has not been necessary[6]. In particular, the high-spatial-frequency convergence problems I have previously described for Newton-method apparent horizon finding [48], do not seem to occur often in practice.

If the slice does not contain an AH (or if either the 3D Cartesian grid or the S^2 angular grid has insufficient resolution), then the Newton iteration will probably fail to converge. In practice, AHFinderDirect detects this by limiting the Newton iteration to a maximum number of iterations. It is useful to distinguish between two subcases here:

- If we are searching for an AH or AHs at each time step of a numerical evolution, and we found this AH at the previous time step, then that AH shape probably provides an excellent initial guess for Newton iteration of this step, so a relatively low maximum-iterations limit is appropriate. AHFinderDirect uses a default of 10 iterations for this case.
- Otherwise (if we do *not* have a previous-time-step AH as an initial guess), in practice the initial guess is likely to be rather inaccurate, so a higher maximum-iterations limit is appropriate. AHFinderDirect uses a default of 20 iterations for this case.

In addition to the maximum-iterations limit, AHFinderDirect also aborts the finding of an AH if any trial horizon shape $\{h_I\}$ is outside the 3D Cartesian grid. Otherwise, AHFinderDirect considers an AH to have been found if and only if the ∞-norm of the $\{\Theta_I\}$ values over the angular grid is below a specified threshold (10^{-8} by default).

For better efficiency, in a multiprocessor environment AHFinderDirect finds multiple AHs in parallel across multiple processors. I describe the algorithm for doing this in appendix B.

6. Computing the expansion Θ (discrete)

Given a trial AH shape $\{h_I\}$, I compute the expansion $\{\Theta_I\}$ using (13), approximating the angular derivatives $\partial_u h$ and $\partial_{uv} h$ by the usual centred fourth-order finite difference molecules $\mathtt{D}_u$ and $\mathtt{D}_{uv}$ respectively. However, there are several complications in this process, which I discuss in the following subsections.

[6] Additionally, due to the way AHFinderDirect finds multiple AHs in parallel across multiple processors (discussed in detail in appendix B), it would be difficult to use many of the uniprocessor modified-Newton software packages such as [36, 38, 39].

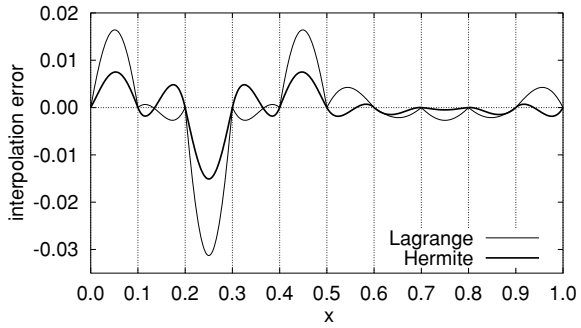

Figure 1. This figure shows the errors for cubic Lagrange and Hermite interpolation of the function $f(x) = \exp[\sin(2\pi x)]$ with grid spacing $\Delta x = 0.1$. Note that the Lagrange error (and hence the Langrange interpolant itself) is non-differentiable at the grid points, whereas the Hermite error (and interpolant) is differentiable everywhere.

6.1. Geometry interpolation

As shown in section 3, the expansion Θ implicitly depends on the geometry variables g_{ij}, K_{ij} and $\partial_k g_{ij}$ at the AH surface. In practice the geometry variables are only known on a (3D) Cartesian grid, so they must be interpolated to the AH surface.

Instead of computing the 3-metric derivatives $\partial_k g_{ij}$ on the full 3D grid and then interpolating these values to the AH surface, it is much more efficient to do the differentiation only at the AH-surface points, inside the interpolator: an interpolator generally works by (conceptually) locally fitting a fitting function (usually a low-degree polynomial) to the data points in a neighbourhood of the interpolation point, then evaluating the fitting function at the interpolation point. By evaluating the *derivative* of the fitting function, the $\partial_k g_{ij}$ values can be interpolated very cheaply, using only the 3D input values which are used anyway for interpolating the g_{ij}.

Even for C^∞ g_{ij} and K_{ij}, the usual Lagrange polynomial interpolators give results which are continuous, but *not differentiable* (the interpolated $\partial_k g_{ij}$ generically has a jump discontinuity) at each position where the interpolator switches the set of input 3D grid points it uses. (The non-smoothness of interpolation errors is discussed in more detail in [49, appendix F].) Unfortunately, this lack of smoothness propagates into the AH equation (13), sometimes causing Newton's method to fail to converge. To avoid this problem, I use a (cubic) Hermite interpolator, which guarantees that the interpolated g_{ij} and K_{ij} remain differentiable, and that the interpolated $\partial_k g_{ij}$ remains continuous, even when the interpolator switches input-grid-point sets. Figure 1 shows an example of the smoothness properties of Lagrange and Hermite interpolation, for a simple 1D (one-dimensional) model problem.

While the resulting (C^0) smoothness of $\Theta(h)$ is not quite ideal for Newton's method, in practice it seems to be sufficient not to impair convergence, and attaining a higher degree of smoothness would require a significantly more complicated and expensive interpolator.

6.2. Multiple grid patches

To avoid z-axis coordinate singularities in the angular computations, I use multiple grid patches to cover S^2. Figure 2 shows an example of this. In general there are six patches, covering neighbourhoods of the $\pm x$, $\pm y$, and $\pm z$ axes respectively.

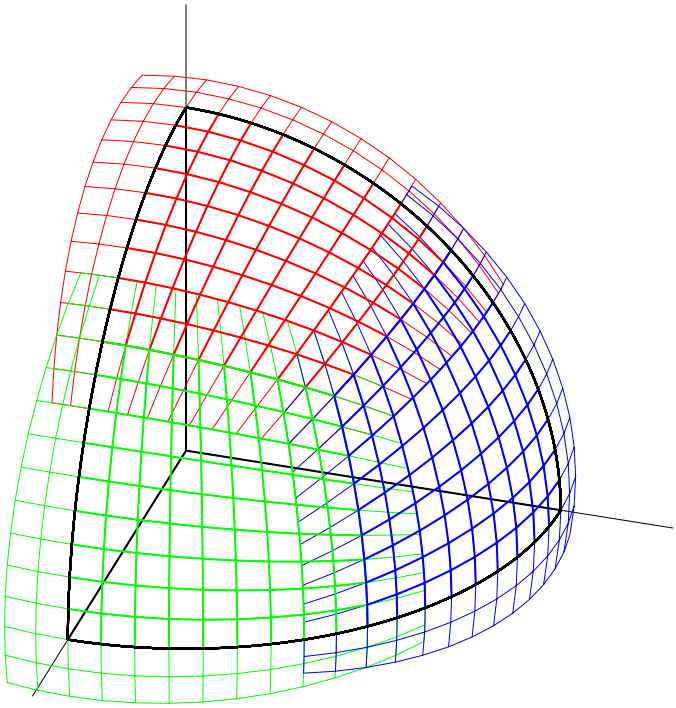

Figure 2. This figure shows a multiple-grid-patch system covering the (+, +, +) octant of S^2 with three patches, at an angular resolution of 5°. The $+z$, $+x$, and $+y$ patches are shown in red, green and blue respectively. The nominal grid of each patch is shown in thick lines; the ghost zones are shown in thin lines.

The nominal grid of each patch (shown in thick lines in figure 2) is surrounded by a 'ghost zone' (shown in thin lines in figure 2). Once the h values in the ghost zones are filled in by symmetry operations and/or interpatch interpolation, the finite differencing code can ignore the patch boundaries in computing Θ. To keep the interpatch interpolation errors (more precisely, their numerical second derivatives) from dominating those of the fourth-order patch-interior angular finite differencing, I use fifth-order Lagrange polynomial interpolation. The patch coordinates (ρ, σ) are defined such that adjacent patches always share a common angular coordinate, so only 1D interpolation is required here. I describe the multiple-patch scheme in detail in appendix A.

The Jacobian matrix $\mathbf{J}_{\mathrm{IJ}}$ must also take into account the ghost-zone symmetry operations and interpatch interpolations. This is conceptually simple, but does require explicitly knowing the Jacobian (i.e., the interpolation coefficients) of the interpatch interpolation. The details are somewhat complicated, and are described in appendix A.3.

The multiple-patch scheme works well, but requires a lot of subtle coding, particularly in handling the ghost-zone updates near patch corners. The overall patch infrastructure is currently about 12k (7k non-blank non-comment) lines of C++ code, out of a total of about 25k (15k) lines of C++ and 2.5k (1.5k) of Maple in AHFinderDirect. In hindsight, a much simpler scheme might well have sufficed to avoid z-axis problems. Notably, [41, 42] reports excellent results using a simple latitude–longitude grid on S^2, with the grid points staggered across the north/south poles. Another possibility [23] would be to have two patches meeting at the equator, each using stereographic coordinates.

7. Computing the Jacobian matrix $\mathbf{J}_{\mathrm{IJ}}$

If there are N_{ang} angular grid points, then the Jacobian matrix $\mathbf{J}_{\mathrm{IJ}} \equiv \mathrm{d}\Theta_{\mathrm{I}}/\mathrm{d}h_{\mathrm{J}}$ is an $N_{\mathrm{ang}} \times N_{\mathrm{ang}}$ matrix; $\mathbf{J}$ is sparse due to the locality of the angular finite differencing. The obvious way to compute $\mathbf{J}$ is by numerical perturbation: perturb h at a single angular grid point J, then re-evaluate[7] Θ and determine the Jth column of $\mathbf{J}$ from the changes in Θ. However, for typical N_{ang} values of 300–3000, this is very slow (though its relative simplicity makes it useful for debugging purposes).

Instead of numerical perturbation, AHFinderDirect normally uses the 'symbolic differentiation' algorithm of [48, section VI] to compute $\mathbf{J}$ directly from the angular ∂_u and ∂_{uv} finite difference molecule coefficients and the (continuum) Jacobian coefficients $\partial\Theta/\partial(\partial_u h)$ and $\partial\Theta/\partial(\partial_{uv} h)$. Temporarily neglecting the interpatch interpolation, the Jacobian is thus given by

$$\mathbf{J}_{\mathrm{IJ}} \equiv \frac{\mathrm{d}\Theta_{\mathrm{I}}}{\mathrm{d}h_{\mathrm{J}}} = \left\{ \begin{array}{cc} \dfrac{\partial\Theta}{\partial(\partial_u h)} \mathtt{D}_u[\mathrm{J}-\mathrm{I}] & \text{if } \mathrm{J}-\mathrm{I} \in \mathtt{D}_u \\ 0 & \text{otherwise} \end{array} \right\} + \left\{ \begin{array}{cc} \dfrac{\partial\Theta}{\partial(\partial_{uv} h)} \mathtt{D}_{uv}[\mathrm{J}-\mathrm{I}] & \text{if } \mathrm{J}-\mathrm{I} \in \mathtt{D}_{uv} \\ 0 & \text{otherwise} \end{array} \right\} + \left\{ \begin{array}{cc} \partial_r\Theta & \text{if } \mathrm{I} = \mathrm{J} \\ 0 & \text{otherwise} \end{array} \right\} \tag{14}$$

where the first two terms describe the variation in Θ at a fixed spatial position with respect to h, and the last term describes the variation in Θ due to a change in h changing the evaluation position of (and thus the position-dependent coefficients in) Θ. Note that there is no term here for $\partial\Theta/\partial h$, since this dependence is included in the $\partial_r\Theta$ term.

As mentioned in section 6.2, the Jacobian (14) must be modified to take into account the ghost-zone symmetry operations and interpatch interpolations. This is described in detail in appendix A.3.

Because Θ depends on g_{ij}, K_{ij} and $\partial_k g_{ij}$ (cf (13)), in theory the $\partial_r\Theta$ term in (14) also requires interpolating $\partial_k K_{ij}$ and $\partial_{k\ell} g_{ij}$ (cf section 6.1). However, doing the computation this way would require a much larger number of interpolations (a total of 80 geometry-interpolator outputs instead of 30), and the expressions for computing $\partial_r\Theta$ from the interpolated values would be quite complicated[8].

To avoid these problems, I approximate $\partial_r\Theta$ by a one-sided radial finite difference, $\partial_r\Theta \approx [\Theta(h+\varepsilon) - \Theta(h)]/\varepsilon$, with ε typically chosen to be 10^{-6} [16], [47, pp 266–7]. Even though this approximation is only $O(\varepsilon)$ accurate, in practice this does not impair the convergence of Newton's method, and it is fairly cheap to compute (one extra $\Theta(h)$ evaluation per Jacobian computation).

8. Solving the linear system $\mathbf{J} \cdot \delta\boldsymbol{h} = -\boldsymbol{\Theta}$

The Jacobian matrix is an $N_{\mathrm{ang}} \times N_{\mathrm{ang}}$ sparse matrix; for typical angular resolutions N_{ang} is in the range 300–3000. Thus for good efficiency it is important to exploit the sparsity of $\mathbf{J}$ in both storage and computation. I have tried several different linear-equation codes and storage formats: for debugging purposes I have found it very useful to store $\mathbf{J}$ as a dense

[7] An important optimization is to only re-evaluate Θ within an angular-molecule-sized neighbourhood of the perturbed point J.

[8] The arguments of section 6.1 would suggest also having the geometry interpolator guarantee at least C^0 continuity of the second derivative values here, although it is not clear if this would actually be necessary in practice.

matrix and solve the linear system with LAPACK routines[9,10]. For better efficiency I now use either an incomplete-LU-decomposition preconditioned conjugate gradient code ILUCG [31], or the UMFPACK sparse-LU-decomposition code [17–20][11,12]; both of these codes use the standard 'compressed row storage' sparse storage scheme for **J**. Neither code has been entirely satisfactory, so I plan to explore other sparse LU-decomposition codes in the near future.

9. Performance and accuracy

In this section I outline the general factors affecting performance (how quickly it can find an AH, or try to find one) and accuracy (how accurately is an AH found) of AHFINDERDIRECT. I also briefly compare AHFINDERDIRECT to other AH finders in these respects. I defer detailed numerical results to section 10.

9.1. Performance

The performance (the time taken to find, or try to find, an AH) of AHFINDERDIRECT depends on two main factors: the total number of angular grid points in the multiple-patch system, and the number of Newton iterations. Since there are no computations done at each Cartesian-grid grid point, the performance is almost independent of the size and resolution of the Cartesian grid[13].

The total number of angular grid points, N_{ang}, is determined by the angular resolution chosen, and whether there are any discrete symmetries in the multiple-patch system. Since practical values of N_{ang} vary over roughly an order of magnitude, and empirically the performance scales very roughly as $N_{\text{ang}}^{1.4}$, the performance varies over a wide range from this factor alone.

The number of Newton iterations performed by AHFINDERDIRECT is mainly determined by the type of AH being searched for:

- AHFINDERDIRECT is fastest when searching for—and successfully finding—an AH at each time step of a numerical evolution. In this case the AH typically only moves a small distance from one time step to the next, so (using the previous-time-step position as an initial guess for the Newton iteration, cf section 5) typically only 3 Newton iterations are needed to locate it at each time step.
- If AHFINDERDIRECT finds an AH in an initial data slice, typically the initial guess is much less accurate, then 6–10 Newton iterations are needed.
- AHFINDERDIRECT is at its slowest when searching for—but failing to find—an AH at each time step of a numerical evolution. In this case (again cf section 5) it typically takes 20 Newton iterations at each time step.

As discussed in appendix B, AHFINDERDIRECT can search for multiple AHs in parallel on a multiprocessor computer system. In practice, for large-scale runs there are usually (many)

[9] LAPACK is available from NETLIB (`http://www.netlib.org`).

[10] The condition number estimator of LAPACK is a particularly valuable debugging and diagnostic tool. For example, incorrect symmetry boundary conditions often result in **J** being singular (infinite condition number). Another example was in investigating why the Newton iteration sometimes failed to converge in an early version of AHFINDERDIRECT which used Lagrange rather than Hermite interpolation for the geometry variables (cf section 6.1): it was useful to be able to rule out ill-conditioning of the linear system as a possible cause of the convergence failure.

[11] UMFPACK is available from `http://www.cise.ufl.edu/research/sparse/umfpack`.

[12] UMFPACK also has a condition number estimator, but as of version 4.0 it appears to be unreliable.

[13] On an idealized computer there would be no Cartesian grid resolution dependence at all, but on actual computers cache effects in the geometry interpolator may cause a slight slowdown at higher Cartesian grid resolutions.

more processors available than the number of AHs being searched for. Assuming this, the elapsed time taken to search for all the AHs in parallel is basically the maximum of the time taken to search for each individual AH; this is roughly independent of both the number of AHs searched for, and the number of processors available. Part (b) of table 4 in appendix B should make this clearer.

9.2. Accuracy

The accuracy with which AHFinderDirect can find an AH is mainly determined by the finite differencing errors in the evaluation of the expansion Θ. There are two main error contributions: the geometry interpolation from the Cartesian grid to the AH position, and the angular finite differencing within the multiple-patch system on S^2. (Other error sources such as the interpatch interpolation, the non-zero $\|\Theta\|$ at which the code considers the Newton iteration to have 'converged', and floating-point roundoff errors, are generally negligible in comparison to the main finite differencing errors.)

For given (smooth) g_{ij} and K_{ij}, the errors from the geometry interpolator are determined by the 3D (Cartesian) grid spacing Δxyz, and by the order of the interpolation scheme. In the limit of small Δxyz, a cubic Hermite geometry interpolator gives g_{ij} and K_{ij} to $O((\Delta xyz)^4)$ and $\partial_k g_{ij}$ to $O((\Delta xyz)^3)$, contributing $O((\Delta xyz)^3)$ errors to Θ. However, at practical resolutions of $\Delta xyz \sim 0.03m$–$0.1m$ I find that the convergence is often 0.5–1.0 power of Δxyz better than this, only dropping to the theoretical limits for very high-resolution grids (in practice, $\Delta xyz \lesssim 0.01m$).

AHFinderDirect uses fourth-order angular finite differencing within the multiple-patch system on S^2, which contributes $O((\Delta\rho\sigma)^4)$ errors to Θ, where $\Delta\rho\sigma$ is the angular resolution.

9.3. Comparison to other AH finding methods

Curvature-flow or fast-flow methods are widely used for AH finding (see, for example, [25, 30, 45, 51]). Conceptually, a flow method starts with a large 2-surface, and flows this inwards, in such a manner that the flow velocity vanishes on the AH. Unfortunately, this means that the method must move the 2-surface through a large part of the 3D grid—and thus must do nontrivial computations at a large number of 3D grid points—before the surface can closely approximate the AH. In contrast, an elliptic-equation method such as that used by AHFinderDirect need only do computations on a 2D set of (AH-surface) grid points, so it can potentially be must faster.

However, a flow method can (at least modulo numerical errors) guarantee to find the *outermost* MTS in a slice, whereas an elliptic-equation method is only locally convergent, and hence offers no information on what other MTSs might be outside any 'AH' it finds.

Another common class of AH finding methods are function-minimization methods such as those described in [5, 10]. These parametrize a trial AH surface by spherical harmonic or other spectral coefficients, define a surface-integral error norm $\int \Theta^2 \, dA$ which has a global minimum of 0 at the AH surface, then use a general-purpose function-minimization algorithm to minimize the error norm over the surface-coefficient space. These methods are inherently quite slow because (for a generic slice with with no continuous symmetries) they must determine a fairly large number of surface coefficients, and the generic function-minimization algorithm only 'learns' a single number (the error norm) for each surface evaluation, and thus requires many surface evaluations to converge. For example, using a spherical harmonic expansion up to order N to parametrize the AH surface, there are $O(N^2)$ surface coefficients, so $O(N^2)$ iterations are needed to converge. Each iteration takes $O(N^2)$ work to evaluate

the surface integral, so the total work is $O(N^4)$. The exponential convergence of spectral series allows N to be chosen to be fairly small for a given surface accuracy, but in practice function-minimization AH finders are still very slow.

Minimization methods are also inherently somewhat limited in their accuracy, because the location of the error-norm minimum is very sensitive to small numerical errors. (In general, relative errors of $O(\varepsilon)$ in a smooth function result in relative errors of $O(\sqrt{\varepsilon})$ in the location of the function's minima.)

9.4. What makes AHFinderDirect *fast?*

Based on the above analyses, I think the key algorithm component which makes AHFinderDirect fast is the posing of the AH equation (1) as an elliptic PDE on S^2 for the AH shape function h. Given this, I believe that any efficient implementation would result in an AH finder with roughly the same performance and accuracy as AHFinderDirect.

A notable example of this is Schnetter's AH finder [41, 42], which poses the AH equation in the same manner as mine, but uses a rather different finite differencing scheme and solution method for the finite difference equations. We have not yet made a detailed comparison of our AH finders, but it appears they are broadly comparable in performance and accuracy.

Huq's AH finder [27, 28] also poses the AH equation as an elliptic PDE on S^2, but he uses Cartesian grid finite differencing to evaluate the surface expansion and Jacobian matrix, rather than the angular grid finite differencing which Schnetter and I use. Because of this, and because he uses numerical perturbations to compute the Jacobian matrix (cf section 7), Huq's AH finder is roughly an order of magnitude slower than mine.

10. Sample results

In this section I present various sample results to test and demonstrate performance of AHFinderDirect. For comparison, I also show some results for another AH finder implemented in the Cactus toolkit, the fast-flow method of [25][14]. (This was the main Cactus AH finder prior to AHFinderDirect.) Although some of the test slices are in fact axisymmetric, I configured both AH finders to treat the slices as fully 3D, with only the discrete symmetries of reflection across the x, y and/or $z = 0$ planes as appropriate. All timings are user-mode CPU times on a 1.7 GHz dual Pentium IV processor system (256 kB cache per processor) with 1.0 GB of memory.

10.1. Boosted Kerr slices

As a first test case, I first consider Kerr spacetime in Kerr–Schild coordinates [35, exercise 33.8], where the AH is a coordinate ellipsoid with radia (semi-major axes)

$$r_z = (1 + \sqrt{1 - a^2})m \tag{15a}$$

$$r_x = r_y = r_z\sqrt{1 + \left(\frac{am}{r_z}\right)^2} = \sqrt{\frac{2r_z}{m}}m \tag{15b}$$

and area

$$A = 4\pi\left(r_z^2 + a^2m^2\right) \tag{15c}$$

[14] Cactus thorn AHFinder, slightly modified to allow a spherical harmonic expansion up to degree $\ell_{\max} = 50$ (by default the limit is 19). (As discussed below, in practice AHFinder is limited to $\ell_{\max} \lesssim 20$.)

where $a \equiv J/m^2$ is the dimensionless angular momentum of the black hole. I then Lorentz-boost this with a velocity v in the x direction. The horizon area is invariant under the boost, but in coordinate system of the code, length-contraction makes the AH a triaxial ellipsoid, and the interaction of the black-hole spin and the boost results in the slice *not* being symmetric across either the $x = 0$ or $y = 0$ planes.

Table 1 shows the accuracy and performance of AHFinderDirect and the fast-flow AH finder on various boosted Kerr slices, for a number of choices of the various numerical parameters.

The first section of the table shows the behaviour of AHFinderDirect as the resolution of the underlying Cartesian grid is varied, using the default cubic Hermite geometry interpolator. At very low resolution ($\Delta xyz = 0.2$) AHFinderDirect fails to find the AH, due to the geometry interpolation 'seeing' the Kerr ring singularity. At higher resolution (decreasing Δxyz) the accuracy improves rapidly, until it levels out at high resolutions due to the angular finite differencing errors. For the computer system used here, the time taken to find the AH is essentially independent of the Cartesian grid resolution.

The second section of the table shows the behaviour of AHFinderDirect as the resolution of the underlying Cartesian grid is varied, using a lower-order (quadratic) geometry interpolator. Compared to the default (cubic) geometry interpolator, this makes AHFinderDirect a factor of 2 to 3 faster, and roughly an order of magnitude less accurate. Also, at the very lowest resolution AHFinderDirect is now able to find the AH, when it could not find it using the cubic interpolator.

Comparing the first two sections of the table shows that changing the interpolation order seems to make only a minor difference to the behaviour of the fast-flow method; all the remaining tests use its default (quadratic Lagrange) geometry interpolator. As discussed in section 9.3, the fast-flow method becomes much slower at high Cartesian grid resolutions.

The third section of the table shows the behaviour of AHFinderDirect as the angular resolution is varied. As the resolution is increased (decreasing $\Delta\rho\sigma$) AHFinderDirect becomes slower but more accurate, until the error levels off at high angular resolutions due to the Cartesian grid geometry-interpolation errors.

The third section of the table also shows the fast-flow method becoming slower as its resolution parameter $\ell_{\max}$ is increased. Unfortunately, beyond $\ell_{\max} \approx 10$ the accuracy of the method stops improving and begins to worsen, and beyond $\ell_{\max} \approx 20$ the fast-flow method fails to find the AH. I suspect this is due to numerical ill-conditioning, but I have not investigated this in detail.

The fourth section of the table shows the behaviour of AHFinderDirect when the local coordinate origin is offset from the coordinate origin. Note that the accuracy with which the AH is found is not significantly changed, and the time taken to find the AH is only mildly increased, even when the local coordinate origin is offset by up to $1/2$ the AH radius. The fast-flow method is still able to find the AH with the offset local coordinate origins, but it requires changes to the initial guess, and (even after correcting for the larger grid) it slows dramatically and becomes less accurate.

The final section of the table shows the behaviour of AHFinderDirect on some more difficult boosted Kerr slices, where the spin is closer to maximal and/or the boost is larger. Because the ring singularity in Kerr moves closer to the AH at high spins, and length contraction makes the AH strongly triaxial at high boosts, these tests used higher Cartesian and angular resolutions than the previous tests. AHFinderDirect still finds the horizon rapidly and with high accuracy in these cases, although in the two most difficult cases quite good initial guesses were required. The fast-flow method is not able to find the AH for any of these cases, even

Table 1. This table shows the accuracy and performance of AHFinderDirect on various boosted Kerr slices. In each case the black hole has dimensionless rest mass $m = 1$. Except as noted, the Cartesian grid is of size ± 2.5 (more precisely, $[-2.5, +2.5]$ in x and y and $[0, 2.5]$ in z, with $z \leftrightarrow -z$ reflection symmetry across the $z = 0$ plane). Except as noted, the AHFinderDirect initial guess is a coordinate sphere of radius 1.5, and the fast-flow initial guess is a coordinate sphere of radius 2. AHFinderDirect used the Lucg sparse matrix routines in all cases. In most cases the ∞-norm error in the AHFinderDirect AH shape was less than twice the rms-norm error shown here; in no case did it exceed 5 times the rms-norm error.

					AHFinderDirect						Fast flow			
a	v_x	$(r_x/\gamma, r_y, r_z)$	Δxyz	Origin	$\Delta\rho\sigma$	Interp	$N_{\rm ang}$	Time	$\|\delta h\|_{\rm rms}$	$(\delta A)/A$	$\ell_{\rm max}$	Interp	Time	$(\delta A)/A$
0.8	0.8	(1.07, 1.79, 1.60)	0.20	(0.0, 0.0)	5°	H3	1121	2.0	failed	failed	10	L3	25	1.2×10^{-2}
0.8	0.8	(1.07, 1.79, 1.60)	0.15	(0.0, 0.0)	5°	H3	1121	4.0	3.4×10^{-4}	4.7×10^{-4}	10	L3	26	6.5×10^{-3}
0.8	0.8	(1.07, 1.79, 1.60)	0.10	(0.0, 0.0)	5°	H3	1121	4.1	5.4×10^{-5}	7.8×10^{-5}	10	L3	33	2.2×10^{-3}
0.8	0.8	(1.07, 1.79, 1.60)	0.05	(0.0, 0.0)	5°	H3	1121	4.2	2.5×10^{-5}	4.1×10^{-5}	10	L3	96	4.6×10^{-4}
0.8	0.8	(1.07, 1.79, 1.60)	0.03	(0.0, 0.0)	5°	H3	1121	4.3	2.6×10^{-5}	4.1×10^{-5}	10	L3	350	1.0×10^{-3}
0.8	0.8	(1.07, 1.79, 1.60)	0.20	(0.0, 0.0)	5°	H2	1121	1.8	3.6×10^{-3}	4.7×10^{-3}	10	L2	24	1.1×10^{-2}
0.8	0.8	(1.07, 1.79, 1.60)	0.15	(0.0, 0.0)	5°	H2	1121	1.8	4.0×10^{-4}	4.3×10^{-4}	10	L2	26	6.2×10^{-3}
0.8	0.8	(1.07, 1.79, 1.60)	0.10	(0.0, 0.0)	5°	H2	1121	1.9	6.5×10^{-4}	1.0×10^{-3}	10	L2	32	2.1×10^{-3}
0.8	0.8	(1.07, 1.79, 1.60)	0.05	(0.0, 0.0)	5°	H2	1121	1.7	1.3×10^{-4}	2.4×10^{-4}	10	L2	95	4.5×10^{-4}
0.8	0.8	(1.07, 1.79, 1.60)	0.03	(0.0, 0.0)	5°	H2	1121	2.0	3.4×10^{-5}	4.7×10^{-5}	10	L2	350	1.0×10^{-3}
0.8	0.8	(1.07, 1.79, 1.60)	0.05	(0.0, 0.0)	7.5°	H3	533	2.0	1.3×10^{-4}	2.0×10^{-4}	7	L2	69	2.7×10^{-3}
0.8	0.8	(1.07, 1.79, 1.60)	0.05	(0.0, 0.0)	5.0°	H3	1121	4.2	2.5×10^{-5}	4.1×10^{-5}	10	L2	95	4.5×10^{-4}
0.8	0.8	(1.07, 1.79, 1.60)	0.05	(0.0, 0.0)	3.0°	H3	2945	13	4.4×10^{-6}	7.3×10^{-6}	15	L2	170	6.9×10^{-4}
0.8	0.8	(1.07, 1.79, 1.60)	0.05	(0.0, 0.0)	1.8°	H3	7905	43	1.3×10^{-6}	1.8×10^{-6}	20	L2	280	1.3×10^{-3}
0.8	0.8	(1.07, 1.79, 1.60)	0.05	(0.0, 0.0)	1.0°	H3	25025	220	9.5×10^{-7}	1.3×10^{-6}	28	L2	2600	failed
0.8	0.8	(1.07, 1.79, 1.60)	0.05	$(-0.5, -0.9)$[a]	5°	H3	1121	5.7	2.4×10^{-5}	3.3×10^{-5}	10	L2	960	1.2×10^{-3}
0.8	0.8	(1.07, 1.79, 1.60)	0.05	(0.0, 0.0)	5°	H3	1121	4.2	2.5×10^{-5}	4.1×10^{-5}	10	L2	95	4.5×10^{-4}
0.8	0.8	(1.07, 1.79, 1.60)	0.05	$(+0.5, +0.9)$[a]	5°	H3	1121	7.2	1.7×10^{-5}	2.7×10^{-6}	10	L2	960	5.6×10^{-3}
0.99	0.8	(0.91, 1.51, 1.14)	0.03	(0.0, 0.0)	3°	H3	2945	15	3.3×10^{-6}	4.8×10^{-6}	15	L2	2300	failed
0.999	0.8	(0.87, 1.45, 1.04)	0.03	(0.0, 0.0)	3°	H3	2945	16	9.5×10^{-6}	1.4×10^{-5}	15	L2	1600	failed
0.999	0.95	(0.45, 1.45, 1.04)	0.02	(0.0, 0.0)[b,d]	3°	H3	2945	13	4.0×10^{-4}	6.5×10^{-4}	not tested on this slice			
0.999	0.98	(0.29, 1.45, 1.04)	0.02	(0.0, 0.0)[c,d]	3°	H3	2945	12	1.3×10^{-4}	1.1×10^{-4}	not tested on this slice			

Origin: the (x, y) components of the local coordinate origin (the z component is always 0); interp: the geometry interpolator—$H(L)$ means Hermite (Lagrange) polynomial interpolation, the following integer gives the order; time: user-mode CPU time in seconds; $\|\delta h\|_{\rm rms}$: the rms-norm over the angular grid of the error in the computed AH radius h; $(\delta A)/A$: the relative error in the computed AH area; $\ell_{\rm max}$: the maximum order of the spherical harmonic expansion.

[a] Fast-flow initial guess changed to a coordinate sphere of radius 2.5, and Cartesian grid enlarged to size ± 4 (the larger Cartesian grid size points should have only minimal effects on performance of AHFinderDirect but should slow the fast-flow method by a factor of $(4/2.5)^3 \approx 4$).

[b] AHFinderDirect initial guess changed to a coordinate ellipsoid of radii (0.5, 1.5, 1.0).

[c] AHFinderDirect initial guess changed to a coordinate ellipsoid of radii (0.3, 1.5, 1.0).

[d] Cartesian grid shrunk to size ± 2 to reduce memory usage (this should have only minimal effects on the performance of AHFinderDirect).

with some adjustment of its initial guesses (this may be due in part to its user interface only allowing for axisymmetric initial guesses).

Across all the boosted Kerr tests, AHFinderDirect is roughly an order of magnitude faster, and two orders of magnitude more accurate, than the fast-flow method.

10.2. Misner and Brill–Lindquist slices

The Misner [33, 34] and Brill–Lindquist [12] initial data slices are standard test problems in numerical relativity. Both are time symmetric ($K_{ij} = 0$), 3-conformally-flat ($g_{ij} = \Psi\eta_{ij}$ for some spatially varying conformal factor Ψ), and (for suitable values of their parameters) may contain any number $N \geqslant 1$ of black holes.

The simplest case of Misner data (and the only case I consider here) is that of two throats, each of bare mass unity. Here the conformal factor is

$$\Psi = 1 + \sum_{n=1}^{\infty} \frac{1}{\sinh(n\mu)} \left(\frac{1}{r_n^+} + \frac{1}{r_n^-} \right) \tag{16}$$

where

$$r_n^{\pm} = \sqrt{x^2 + y^2 + [z \pm \coth(n\mu)]^2}, \tag{17}$$

with $\mu > 0$ a real parameter. The individual throats are located at coordinate positions $(0, 0, \pm \coth \mu)$. For small μ there is only a single AH enclosing both throats, while for large μ there are individual AHs enclosing each throat, but no common AH enclosing both throats.

The conformal factor for N-throat Brill–Lindquist initial data is

$$\Psi(\vec{x}) = 1 + \frac{1}{2} \sum_{i=1}^{N} \frac{m_i}{|\vec{x} - \vec{x}_i|} \tag{18}$$

where the ith throat has bare mass m_i and is located at the coordinate position $\vec{x}_i$. Here I consider the cases $N = 2$ and $N = 3$, where the throats have bare mass unity, and are uniformly spaced in a coordinate circle of radius $R > 0$. Similarly to the Misner data, for small R there is a single common AH, while for large R there are N individual AHs but no common AH.

The AH finder test problem I consider here is to numerically determine the 'critical' value of the parameter (μ for Misner, R for Brill–Lindquist) at which the common horizon appears/disappears for each family of slices. To do this, I used the Cactus thorn IDAnalyticBH to construct the initial data slices, approximating the infinite sum (16) by its first 30 terms[15]. For each of a number of combinations of the Cactus Cartesian grid spacing and the AHFinderDirect angular grid spacing[16], I used a continuation-method binary search (described in detail in appendix C) to determine the critical parameter. I did convergence tests [14] in both grid spacings to verify that the values shown are reliable estimates of the true continuum values, and I used Richardson extrapolation in the angular grid spacing to improve the accuracy[17]. Table 2 shows the results, together with values reported by [1] for comparison. The AHFinderDirect values are in excellent agreement with those of [1], and are dramatically more accurate.

[15] Raising this to 50 terms changed the numerically computed critical μ by $<10^{-12}$, and the horizon area by $<10^{-7}$.

[16] I used very high resolutions here, with grid spacings as small as 0.01 for the Cactus 3D grid and 0.5° for the AHFinderDirect angular grid.

[17] Because the AHFinderDirect angular grid is not commensurate with the Cactus Cartesian grid, the geometry-interpolation errors effectively have a quasirandom phase at each angular grid point. This prevents these errors from being smooth enough to allow Richardson extrapolation on the Cartesian grid spacing. However, the variation of the computed critical parameters with the Cartesian grid spacing can still be used qualitatively to help estimate the accuracy of the critical parameters.

Table 2. This table shows the maximum Misner μ and Brill–Lindquist R for which AHFinderDirect found a common AH, along with the area of that common AH. All uncertainties are in units of the last digits shown. For the 2-throat Brill–Lindquist data, other values in the literature include $R = 0.767 \pm 1$ [52] and $R = 0.768$ [45]. However, [45] reports a critical AH area for this case of 184.16, about 6% different from that of AHFinderDirect. I do not know the cause of this discrepancy.

	Alcubierre *et al*		AHFinderDirect	
Test problem	Parameter	Critical parameter	Critical parameter	Critical AH area
Misner	μ	1.364	$1.365\,071\,172 \pm 3$	$409.549\,358 \pm 3$
Brill–Lindquist 2-throat	R	0.766	$0.766\,197\,45 \pm 5$	$196.407\,951 \pm 3$
Brill–Lindquist 3-throat	R	1.18 ± 4	$1.195\,499\,53 \pm 5$	$444.756\,224 \pm 3$

Table 3. This table gives various parameters for the binary-black-hole collision evolution shown in figures 3 and 4.

Initial data	Misner $\mu = 2.0$ (ADM mass $m = 1.272$) black holes located on the z-axis at $z = \pm 0.99$
Coordinates	1 + log lapse, Γ-driver shift
Numerical grid	$\Delta xyz = 0.0666$ ($0.052m$); $+xyz$ octant symmetry
Time integration	Iterated Crank–Nicholson (3 iterations) Courant number $\Delta t/\Delta xyz = 0.25$
Outer boundaries	In the computational coordinates the outer boundaries were at $xy_{\max} = 5.37$ ($4.22m$), $z_{\max} = 6.43$ ($5.05m$); a 'fisheye' nonuniform-grid transformation was used which placed the physical outer boundaries at $xy_{\max} = 13.1$ ($10.3m$), $z_{\max} = 16.3$ ($12.8m$)
AHFinderDirect	Searching for individual and inner/outer common AHs at each time step cubic Hermite geometry-interpolation; angular resolution $\Delta\rho\sigma = 5^\circ$ for individual AH ($N_{\text{ang}} = 580$, $+xy$ quadrant symmetry, local coordinate origin (0, 0, 0.85)); angular resolution $\Delta\rho\sigma = 3^\circ$ for inner and outer common AHs ($N_{\text{ang}} = 768$, $+xyz$ octant symmetry, local coordinate origin (0, 0, 0)); Umfpack sparse matrix routines
Fast-flow AH finder	Searching for individual and common AHs each 10 time steps; fast-flow method [25], spherical harmonics up to $\ell_{\max} = 10$; same local coordinate origins as AHFinderDirect

10.3. Binary-black-hole collision spacetimes

As a final example, I consider the binary-black-hole collision evolution described in table 3. Figure 3 shows the AH areas found by AHFinderDirect and the fast-flow AH finder for this evolution. For the AHs they both find, the two AH finders agree very well. AHFinderDirect found the outer common AH somewhat sooner than the fast-flow AH finder ($t = 4.633$ ($3.64m$) versus $t = 5.50$ ($4.32m$)), and was the only finder to find the inner common AH. Figure 4 shows the three AHs found by AHFinderDirect at two times during the evolution.

For this evolution the mean CPU times per time step were 5.2 s for AHFinderDirect and (for those time steps for which it ran) 55 s for the fast-flow AH finder, so despite searching for three AHs instead of two, AHFinderDirect was about an order of magnitude faster than the fast-flow method.

Since the runs just described, I have changed AHFinderDirect's default geometry interpolator from cubic to quadratic Hermite. In practice AHFinderDirect is usually used

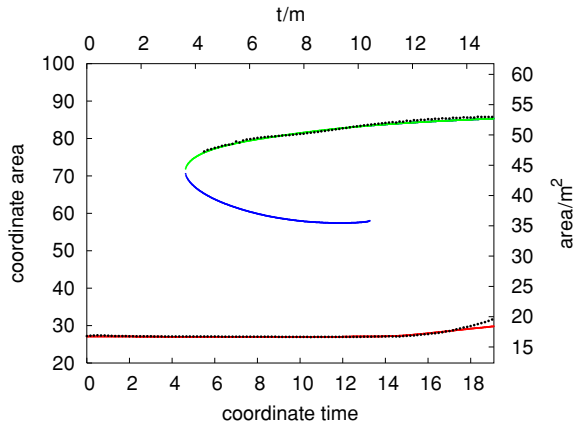

Figure 3. This figure shows the areas of the various AHs in the Misner $\mu = 2.0$ collision described in table 3. The black points are the areas found by the fast-flow AH finder; the other curves are all from AHFinderDirect. The gradual rise in the area of the outer common AH after $t \approx 9$, and in the area of the individual AH after $t \approx 15$, is due to outer boundary reflections making the overall evolution inaccurate.

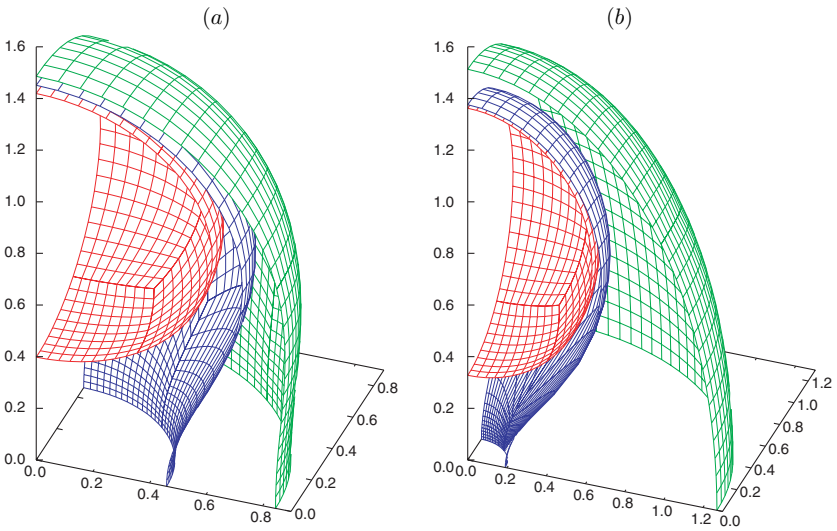

Figure 4. This figure shows the three AHs in the Misner $\mu = 2.0$ collision described in table 3. (*a*) The horizons at $t = 5.00$ ($3.93m$), (*b*) at $t = 8.00$ ($6.28m$). In both parts the colour coding matches that of figure 3.

in numerically computed slices whose geometries have numerical errors large enough to dominate intrinsic errors of AHFinderDirect. Thus (cf section 10.1) the lower-order geometry interpolation makes little difference to the practical accuracy with which the apparent horizons are found, and it speeds up AHFinderDirect by roughly a factor of 2 to 3. For example, in a recent large binary-black-hole collision simulation (details of which will be reported elsewhere), AHFinderDirect (using the Umfpack sparse matrix routines) averaged 1.7 s per time step, as compared with 61 s per time step for the fast-flow AH finder.

11. Conclusions

In this paper I present a detailed description of a new numerical apparent horizon finder for three-dimensional Cartesian grids, AHFinderDirect. AHFinderDirect is typically at least an order of magnitude faster than other widely used apparent horizon finders; in particular AHFinderDirect is fast enough that it is practical to find apparent horizons at each time step of a numerical evolution. This allows apparent horizon positions to readily be used in coordinate conditions (see, for example, [46]) or for other diagnostic purposes.

AHFinderDirect is also very accurate, typically finding apparent horizons to within $\sim 10^{-5} m$ in coordinate position.

AHFinderDirect is implemented within the Cactus computational toolkit, and is freely available (GNU GPL licensed) by anonymous CVS checkout from `cvs.aei.mpg.de`: `/numrelcvs` in the directory `AEIThorns/AHFinderDirect`. It would also be fairly easy to port AHFinderDirect to a different (non-Cactus) numerical relativity code.

Acknowledgments

I thank the Alexander von Humboldt foundation, the AEI visitors programme, and the AEI postdoctoral fellowship programme for financial support. I thank Peter Diener and Ian Hawke for useful conversations. I thank Peter Diener, Ian Hawke, Scott Hawley, Denis Pollney, Ed Seidel and Erik Schnetter for helpful comments on various drafts of this paper. I thank two anonymous referees for a number of helpful comments. I thank Frank Herrmann for providing the last sample binary-black-hole evolution discussed in section 10.3. I thank Tom Goodale, Thomas Radke, and many others for their assistance with the invaluable Cactus computational toolkit. I thank Thomas Radke for our fruitful collaboration on Cactus interpolators, and in particular for his work on the PUGHInterp global interpolator. I thank Erik Schnetter for supplying the Carpet mesh-refinement driver and CarpetInterp global interpolator for Cactus. I thank P Madderom and Tom Nicol for supplying the Ilucg sparse matrix subroutine.

Appendix A. Details of the multiple-patch system

A.1. Coordinates

I define angular coordinates on S^2 based on rotation angles about the local xyz coordinate axes:

$$\begin{aligned} \mu &= \text{rotation angle about the local } x\text{-axis} = \arctan(y/z) \\ \nu &= \text{rotation angle about the local } y\text{-axis} = \arctan(x/z) \\ \phi &= \text{rotation angle about the local } z\text{-axis} = \arctan(y/x) \end{aligned} \tag{A1}$$

where all the arctangents are 4-quadrant based on the signs of x, y and z. I then define coordinate patches covering neighbourhoods of the $\pm z$, $\pm x$, and $\pm y$ axes, using the generic patch coordinates

$$\begin{aligned} &\pm z \text{ patch has generic patch coordinates } (\rho, \sigma) = (\mu, \nu) \\ &\pm x \text{ patch has generic patch coordinates } (\rho, \sigma) = (\nu, \phi) \\ &\pm y \text{ patch has generic patch coordinates } (\rho, \sigma) = (\mu, \phi). \end{aligned} \tag{A2}$$

The resulting set of six patches cover S^2 without coordinate singularities[18]. Alternatively, if the slice has $z \leftrightarrow -z$ reflection symmetry about the local coordinate origin, then the five patches $+z$, $\pm x$ and $\pm y$ cover the $+z$ hemisphere of S^2. Similarly, suitable sets of four or three patches may be used to cover quadrants or octants of S^2 respectively; figure 2 shows an example of this last case.

A.2. Ghost zones

Each patch is a rectangle in its own (ρ, σ) coordinates; I use the usual 'ghost-zone' technique for handling finite differencing near the patch boundaries. I refer to the non-ghost-zone part of a patch's grid as its 'nominal' grid. Adjacent patches' nominal grids just touch. (Grid-function values in) the ghost zones are filled in from values in their own and other patches' nominal grids by symmetry operations and/or interpatch interpolations.

With the coordinate choice (A2), adjacent patches always share the angular coordinate perpendicular to their mutual boundary, so the interpatch interpolations need only be done in one dimension, in the direction parallel to the boundary. Since off-centring an interpolant, particularly a high-order one, significantly degrades its accuracy, I have tried to design the algorithms to keep the interpolations centred wherever possible[19].

The most complicated part of the multiple-patch scheme is in the handling of the 'corner' ghost-zone grid points, those ghost-zone grid points which are outside their patchs' nominal grid in both the ρ and the σ directions. Figure 5 shows the three basic cases:

(a) Figure 5(*a*) shows an example of a corner between two symmetry ghost zones. In this case it takes two sequential symmetry operations (shown by the curved arrows) to fill in the corner from the nominal grid. Fortunately, symmetry operations commute, i.e., the results are independent of the order of the two symmetry operations.

(b) Figure 5(*b*) shows an example of a corner between a symmetry and an interpatch ghost zone. To keep the interpolations centred, I use a 3-phase algorithm here:
   (1) Use symmetry operations (for example, the one shown by the dotted arrow in the figure) to fill in the non-corner ghost-zone points in the neighbouring patch.
   (2) Do a centred interpatch interpolation from the neighbouring patch to this patch; this interpolation may use some points from the neighbouring patch's ghost zone.
   (3) Use symmetry operations (for example, the one shown by the solid arrow in the figure) to fill in the corner ghost-zone points in this patch.

(c) Figure 5(*c*) shows an example of a corner between two interpatch ghost zones (this only happens when three patches meet at a corner). This case requires only a single interpatch interpolation for each ghost-zone grid point.

AHFinderDirect actually uses the following 3-phase algorithm (which includes each of (a)–(c) above as special cases) to perform all the necessary symmetry operations and interpatch interpolations across all patches, in a correct order[20]:

[18] Another way to visualize these patches and coordinates is to imagine an xyz cube with xyz grid lines painted on its face. Now imagine the cube to be flexible, and inflate it like a balloon, so it becomes spherical in shape. The resulting coordinate lines will closely resemble those for (μ, ν, ϕ) coordinates.

[19] Another reason to keep the interpolations centred in AHFinderDirect was to allow re-use of the multiple-patch software from an earlier time evolution code [50], where centring the interpolations helped keeping the evolution stable.

[20] The ordering of the phases is essential to obtain correct results, within each phase the different ghost zones and patches may be processed in any order.

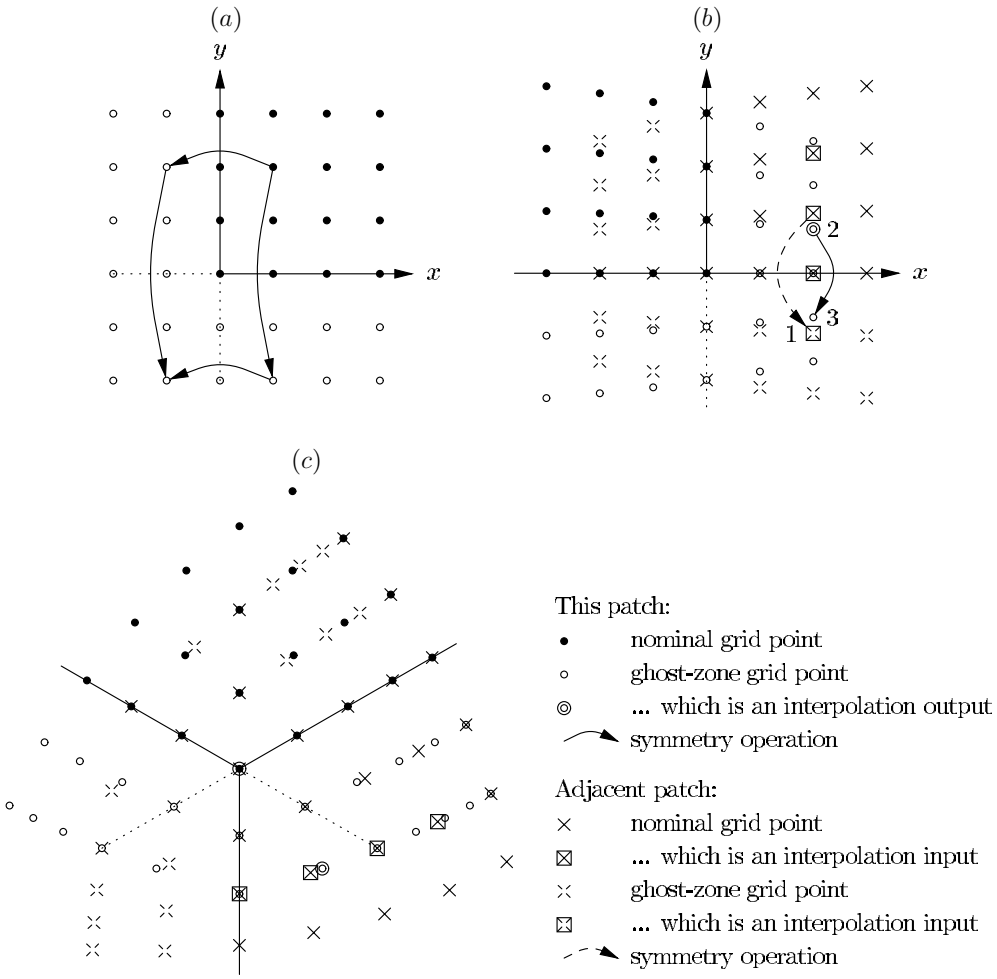

Figure 5. This figure shows the combinations of symmetry operations and interpatch interpolations used to fill in grid function values in ghost-zone corners. (*a*) (Where there are both $x \leftrightarrow -x$ and $y \leftrightarrow -y$ reflection symmetries) a corner between two symmetry ghost zones. (*b*) (Where there is a $y \leftrightarrow -y$ reflection symmetry) a corner between a symmetry and an interpatch ghost zone; sample output points for each phase of the 3-phase algorithm described in the text are labelled as 1, 2 and 3. (*c*) (Where there are no symmetries) a corner between two interpatch ghost zones (this only happens when three patches meet at a corner). In each part, arrows show the symmetry operations, and for (*b*) and (*c*) the boxed and circled points show the inputs and outputs for the interpatch interpolations.

(1) Use symmetry operations to fill in the non-corner parts of all symmetry ghost zones in all patches.
(2) Use interpatch interpolations to fill in all interpatch ghost zones in all patches.
(3) Use symmetry operations to fill in the corners of all symmetry ghost zones in all patches.

A.3. Jacobian computation

The symbolic-differentiation Jacobian (14) must be modified to take into account the ghost-zone symmetry operations and interpatch interpolations described in the previous subsection.

$\mathbf{J} \leftarrow 0$ matrix
for each angular grid point I
{
for each angular coordinate index u and pair of indices uv
{
for each molecule index $\mathtt{m} \in \mathtt{D}_u$ or $\mathtt{D}_{uv}$ respectively
{
$\mathrm{J} \leftarrow \mathrm{I} + \mathtt{m}$
$\mathrm{temp} \leftarrow \dfrac{\partial\Theta}{\partial(\partial_u h)}\mathtt{D}_u[\mathtt{m}]$ or $\dfrac{\partial\Theta}{\partial(\partial_{uv} h)}\mathtt{D}_{uv}[\mathtt{m}]$ respectively
if (J $\in$ nominal grid of the patch containing I)
then $\mathbf{J}_{\mathrm{IJ}} \leftarrow \mathbf{J}_{\mathrm{IJ}} + \mathrm{temp}$
else {
for each angular grid point K used in computing $h[\mathrm{J}]$
via the 3-phase algorithm of appendix A.2
{
$\mathbf{J}_{\mathrm{IK}} \leftarrow \mathbf{J}_{\mathrm{IK}} + \mathrm{temp} \times \left(\dfrac{\partial h[\mathrm{J}]}{\partial h[\mathrm{K}]} \text{ for the 3-phase algorithm} \right)$
}
}
}
}
$\mathbf{J}_{\mathrm{II}} \leftarrow \mathbf{J}_{\mathrm{II}} + \dfrac{\Theta(h+\varepsilon) - \Theta(h)}{\varepsilon}$
}

Figure 6. This figure shows overall Jacobian-computation algorithm of AHFinderDirect including ghost-zone handling.

This is essentially a straightforward application of the chain rule for each of the $\mathtt{D}_u$ and $\mathtt{D}_{uv}$ terms in (14). Figure 6 shows the resulting algorithm in detail.

Appendix B. Multiprocessor and parallelization issues

Cactus (like most modern numerical relativity codes using 3D grids), is designed to run in parallel on multiprocessor computer systems. Cactus uses a domain-decomposition parallelization scheme, where each processor stores and computes the Einstein equations on its own 'chunk' of the spatial grid. Neighbouring chunks overlap slightly[21]; Cactus 'synchronizes' them as necessary. An AH may span multiple processors' grid chunks, and since an AH may move during an evolution, in general we do not know in advance which processors those are.

Because of the domain decomposition, the multiprocessor 'global' interpolator used for the geometry interpolation must in general send each interpolation point to the processor which 'owns' that part of the grid, do the interpolation there, and send the results back to the requesting processor. To ensure that every processor has a flow of control in the interpolator code to (potentially) handle interpolation points in its chunk of the grid, the interpolation must be a *collective* operation: code on every processor *must* call the interpolator synchronously (each processor's code specifying its own choice of interpolation points). Violations of this requirement may result in a deadlock in the interprocessor-communication code.

[21] This is the 3D Cartesian grid analogue of the angular interpatch ghost zones described in the main text of this paper.

Table 4. This table shows two examples of how AHFinderDirect finds multiple horizons in parallel in a multiprocessor environment, for the case where we search for three horizons, which are found (the Newton iteration converges, shown by the $\underline{\Theta}$) after 3, 5 and 4 iterations respectively. The table rows show actions at successive iterations of the algorithm; '–' means a dummy computation (described in the main text). Part (a) shows how the algorithm would work with two processors; part (b) shows how it would work with three or more processors (the last column refers to any processors other than first three).

Processor 1		Processor 2		Processor 3		Any other	
h/it	What	h/it	What	h/it	What	h/it	What
			(a) Two processors				
1/1	Θ	2/1	Θ				
1/1	**J**	2/1	**J**				
1/2	Θ	2/2	Θ				
1/2	**J**	2/2	**J**				
1/3	$\underline{\Theta}$	2/3	Θ				
–	**J**	2/3	**J**				
3/1	Θ	2/4	Θ				
3/1	**J**	2/4	**J**				
3/2	Θ	2/5	$\underline{\Theta}$				
3/2	**J**	–	**J**				
3/3	Θ	–	Θ				
3/3	**J**	–	**J**				
3/4	$\underline{\Theta}$	–	Θ				
			(b) Three or more processors				
1/1	Θ	2/1	Θ	3/1	Θ	–	Θ
1/1	**J**	2/1	**J**	3/1	**J**	–	**J**
1/2	Θ	2/2	Θ	3/2	Θ	–	Θ
1/2	**J**	2/2	**J**	3/2	**J**	–	**J**
1/3	$\underline{\Theta}$	2/3	Θ	3/3	Θ	–	Θ
–	**J**	2/3	**J**	3/3	**J**	–	**J**
–	Θ	2/4	Θ	3/4	$\underline{\Theta}$	–	Θ
–	**J**	2/4	**J**	–	**J**	–	**J**
–	Θ	2/5	$\underline{\Theta}$	–	Θ	–	Θ

h/it = horizon number/iteration number; what = what is this processor doing?

Taking these environmental constraints into account, I have parallelized AHFinderDirect in the following way: to allow the use of standard (uniprocessor) sparse matrix subroutines for solving the Newton-method updating routines, AHFinderDirect assigns each AH to a single processor[22], and searches for that AH only on that processor. However, if there are multiple AHs and multiple processors, AHFinderDirect searches for different AHs concurrently on the multiple processors.

All the processors do their Newton iterations synchronously, each processor working sequentially through its own assigned horizon(s), or doing dummy interpolator calls (to preserve the synchronization across all processors) if it has no assigned horizon(s). If/when a processor finishes with a horizon (either locating it or failing to locate it), the processor moves on to its next assigned horizon if there is one, or switches to doing dummy interpolator calls if it has no more assigned horizons left to process. Table 4 shows two examples of this[23].

[22] A processor may be assigned multiple AHs if there are more AHs than processors.

[23] In part (a) of table 4, note that after horizon 1 converges, processor 1 does a dummy **J** computation before starting on the next horizon. This is slightly inefficient, but considerably simplifies the algorithm by keeping the Θ and **J** computations synchronized across all processors. In the uniprocessor case this dummy **J** operation is unnecessary, and the algorithm omits it.

```
p ← p_start
δp ← δp_start
G ← G_start
    while (|δp| ≥ tolerance)
    {
    try to find a common AH in Σ[p], using G as the initial guess
    if (found a common AH)
      then {
           G ← the common AH just found
           p ← p + δp
           }
      else {
           δp ← (1/2)δp
           p ← p − δp
           }
    }
```

Figure 7. This figure shows the continuation-method binary search algorithm for finding the critical parameter p at which a common AH appears in a 1-parameter family of slices.

This algorithm requires an explicit global synchronization across all processors at each Newton iteration: after evaluating Θ, each processor computes a Boolean flag saying whether that processor needs to continue iterating (this may be true for either or both of two reasons: the Newton iteration has not converged yet on the current horizon, or there is another horizon (or horizons) assigned to this processor which has not yet been processed). All processors then broadcast their flags, and compute the inclusive-or of all the flags to determine whether to continue the algorithm or exit.

Appendix C. Searching for the critical parameter of a 1-parameter initial data sequence

In this appendix I describe my continuation-method binary search algorithm for determining the 'critical' parameter $p_\star$ at which a common AH appears/disappears in a 1-parameter family of initial data slices, $p \mapsto \Sigma[p]$. Without loss of generality I assume that small (large) values of p do (do not) have a common AH.

The main complication here is that AHFinderDirect needs an initial guess for an AH shape, and if this initial guess is inaccurate AHFinderDirect may fail to find the (an) AH. This means that the obvious binary-search algorithm for finding p_* is not reliable, because a failure to find an AH does not rule out the possible existence of that AH.

Instead, I use a continuation method, where p is 'walked up', using the common AHs found in smaller-p slices as initial guesses for trying to find the common AH in larger-p slices. If the algorithm fails to find a common AH, it decreases p and tries again with a smaller 'walking increment' in p. Figure 7 shows this algorithm in detail.

References

[1] Alcubierre M, Brandt S, Brügmann B, Gundlach C, Massó J, Seidel E and Walker P 2000 Test-beds and applications for apparent horizon finders in numerical relativity *Class. Quantum Grav.* **17** 2159–90

[2] Alcubierre M and Brügmann B 2001 Simple excision of a black hole 3 + 1 numerical relativity *Phys. Rev.* D **63** 104006

[3] Alcubierre M, Brügmann B, Pollney D, Seidel E and Takahashi R 2001 Black hole excision for dynamic black holes *Phys. Rev.* D **64** 61501(R)
[4] Anninos P, Bernstein D, Brandt S, Libson J, Massó J, Seidel E, Smarr L, Suen W-M and Walker P 1995 Dynamics of apparent and event horizons *Phys. Rev. Lett.* **74** 630–3
[5] Anninos P, Camarda K, Libson J, Massó J, Seidel E and Suen W-M 1998 Finding apparent horizons in dynamic 3D numerical spacetimes *Phys. Rev.* D **58** 24003
[6] Anninos P, Daues G, Massó J, Seidel E and Suen W-M 1995 Horizon boundary conditions for black hole spacetimes *Phys. Rev.* D **51** 5562–78
[7] Ashtekar A, Beetle C and Fairhurst S 1999 Isolated horizons: a generalization of black hole mechanics *Class. Quantum Grav.* **16** L1–L7
[8] Bank R E and Rose D J 1980 Parameter selection for Newton-like methods applicable to nonlinear partial differential equations *SIAM J. Numer. Anal.* **17** 806–22
[9] Bank R E and Rose D J 1985 Global approximate Newton methods *Numer. Math.* **37** 279–95
[10] Baumgarte T W, Cook G B, Scheel M A, Shapiro S L and Teukolsky S A 1996 Implementing an apparent-horizon finder in three dimensions *Phys. Rev.* D **54** 4849–57
[11] Brandt S *et al* 2000 Grazing collisions of black holes via the excision of singularities *Phys. Rev. Lett.* **85** 5496–9
[12] Brill D and Lindquist R 1963 Interaction energy in geometrostatics *Phys. Rev.* **131** 471–6
[13] Caveny S A, Anderson M and Matzner R A 2003 Tracking black holes in numerical relativity *Preprint* gr-qc/0303099
[14] Choptuik M W 1991 Consistency of finite-difference solutions to Einstein's equations *Phys. Rev.* D **44** 3124
[15] Cook G B *et al* 1998 Boosted three-dimensional black-hole evolutions with singularity excision *Phys. Rev. Lett.* **80** 2512–6
[16] Curtis A R and Reid J K 1974 The choice of step lengths when using differences to approximate Jacobian matrices *J. Inst. Math. Appl.* **13** 121–6
[17] Davis T A 2002 Algorithm 8xx: UMFPACK V3. 2, an unsymmetric-pattern multifrontal method with a column pre-ordering strategy *Technical Report* TR-02-002, University of Florida, CISE Department, Gainesville, FL; www.cise.ufl.edu/tech-reports *(ACM Trans. Math . Softw.* at press)
[18] Davis T A 2002 A column pre-ordering strategy for the unsymmetric-pattern multifrontal method *Technical Report* TR-02-001, University of Florida, CISE Department, Gainesville, FL; www.cise.ufl.edu/tech-reports *(ACM Trans. Math . Softw.* at press)
[19] Davis T A and Duff I S 1997 An unsymmetric-pattern multifrontal method for sparse LU factorization *SIAM J. Matrix Anal. Appl.* **18** 140–58
[20] Davis T A and Duff I S 1999 A combined unifrontal/multifrontal method for unsymmetric sparse matrices *ACM Trans. Math. Softw.* **25** 1–19
[21] Diener P 2003 A new general purpose event horizon finder for 3D numerical spacetimes *Class. Quantum Grav.* **20** 4901–17 (*Preprint* gr-qc/0305039)
[22] Dreyer O, Krishnan B, Shoemaker D and Schnetter E 2002 Introduction to isolated horizons in numerical relativity *Phys. Rev.* D **67** 024018
[23] Gómez R, Lehner L, Papadopoulos P and Winicour J 1997 The eth formalism in numerical relativity *Class. Quantum Grav.* **14** 977–90
[24] Goodale T, Allen G, Lanfermann G, Massó J, Radke T, Seidel E and Shalf J 2003 The CACTUS framework and toolkit: design and applications *VECPAR'2002 Vector and Parallel Processing, 5th Int. Conf. (Lecture Notes in Computer Science)* (Berlin: Springer)
[25] Gundlach C 1998 Pseudo-spectral apparent horizon finders: an efficient new algorithm *Phys. Rev.* D **57** 863–75 (*Preprint* gr-qc/9707050)
[26] Hawking S W and Ellis G F R 1973 *The Large Scale Structure of Spacetime* (Cambridge: Cambridge University Press)
[27] Huq M F 1996 Apparent horizon location in numerical spacetimes *PhD Thesis* The University of Texas at Austin
[28] Huq M F, Choptuik M W and Matzner R A 2002 Locating boosted Kerr and Schwarzschild apparent horizons *Phys. Rev.* D **66** 084024 (*Preprint* gr-qc/0002076)
[29] Dennis J E Jr and Schnabel R B 1978 *Numerical Methods for Unconstrained Optimization and Nonlinear Equations* (Englewood Cliffs, NJ: Prentice-Hall)
[30] Kemball A J and Bishop N T 1991 The numerical determination of apparent horizons *Class. Quantum Grav.* **8** 1361–7
[31] Kershaw D S 1978 The incomplete Cholesky—conjugate gradient method for interative solution of linear equations *J. Comput. Phys.* **26** 43–65
[32] Libson J, Massó J, Seidel E, Suen W-M and Walker P 1996 Event horizons in numerical relativity: methods and tests *Phys. Rev.* D **53** 4335–50

[33] Misner C 1960 Wormhole initial conditions *Phys. Rev.* D **118** 1110–1
[34] Misner C W 1963 The method of images in geometrostatics *Ann. Phys.* **24** 102
[35] Misner C W, Thorne K S and Wheeler J A 1973 *Gravitation* (San Francisco: Freeman)
[36] Moré J J, Garbow B S and Hillstrom K E 1980 User guide for MINPACK-1 *Technical Report* ANL-80-74, Argonne National Laboratory, Argonne, USA, August (available from the NETLIB online software repository, http://www.netlib.org/minpack/)
[37] Nakamura T, Kojima Y and Oohara K 1984 A method of determining apparent horizons in three-dimensional numerical relativity *Phys. Lett.* A **106** 235–8
[38] Nowak U and Weimann L GIANT—a software package for the numerical solution of very large systems of highly nonlinear equations *Technical Report* TR-90-11, Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)
[39] Nowak U and Weimann L 1991 A family of Newton codes for systems of highly nonlinear equations *Technical Report* TR-91-10, Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)
[40] Press W H, Flannery B P, Teukolsky S A and Vetterling W T 1992 *Numerical Recipes* 2nd edn (New York: Cambridge University Press)
[41] Schnetter E 2002 A fast apparent horizon algorithm *Preprint* gr-qc/0206003
[42] Schnetter E 2003 Finding apparent horizons and other two-surfaces of constant expansion *Class. Quantum Grav.* **20** 4719–37 (*Preprint* gr-qc/0306006)
[43] Schroeder M R 1986 *Number Theory in Science and Communication* (Berlin: Springer)
[44] Seidel E and Suen W-M 1992 Towards a singularity-proof scheme in numerical relativity *Phys. Rev. Lett.* **69** 1845–8
[45] Shoemaker D M, Huq M F and Matzner R A 2000 Generic tracking of multiple apparent horizons with level flow *Phys. Rev.* D **62** 124005 12 pp
[46] Sperhake U, Smith K L, Kelly B, Laguna P and Shoemaker D 2003 Impact of densitized lapse slicings on evolutions of a wobbling black hole *Preprint* gr-qc/0307015
[47] Stoer J and Bulirsch R 1980 *Introduction to Numerical Analysis* (Berlin: Springer)
[48] Thornburg J 1996 Finding apparent horizons in numerical relativity *Phys. Rev.* D **54** 4899–918
[49] Thornburg J 1999 A 3 + 1 computational scheme for dynamic spherically symmetric black hole spacetimes: I. Initial data *Phys. Rev.* D **59** 104007
[50] Thornburg J 2003 A multiple-grid-patch evolution scheme for 3-D black hole excision *The Ninth Marcel Grossman Meeting: On Recent Developments in Theoretical and Experimental General Relavtivity, Gravitation, and Relativistic Field Theories* ed V Gurzadyan, R T Jantzen and R Ruffini (Singapore: World Scientific) (*Preprint* gr-qc/0012012)
[51] Tod K P 1991 Looking for marginally trapped surfaces *Class. Quantum Grav.* **8** L115–8
[52] Čadež A 1974 Apparent horizons in the two-black-hole problem *Ann. Phys.* **83** 449–57
[53] Wald R M 1984 *General Relativity* (Chicago, IL: University of Chicago Press)
[54] York J 1979 Kinematics and dynamics of general relativity *Sources of Gravitational Radiation* ed L Smarr (Cambridge: Cambridge University Press)
[55] York J 1989 Initial data for collisions of black holes and other gravitational miscellany *Frontiers in Numerical Relativity* ed C Evans, L Finn and D Hobill (Cambridge: Cambridge University Press) pp 89–109