Multiple Precision, Multiple Processor Vortex Sheet Roll-Up Computation

David H. Bailey, Robert Krasny and Richard Pelz

October 9, 1992

Ref: *Proc. Sixth SIAM Conference on Parallel Processing for Scientific Computing*, 1993, SIAM, pg. 52–56

Abstract

This paper describes a vortex sheet roll-up computation implemented using multiple precision arithmetic on a highly parallel computer (an Intel iPSC/860).

Bailey: NAS Applied Research Branch, NASA Ames Research Center, Moffett Field, CA 94035-1000; email: `dbailey@nas.nasa.gov`. Computer time was provided by the NAS Systems Division at NASA Ames.

Krasny: Dept. of Mathematics, University of Michigan, Ann Arbor, MI 48109; email: `krasny@math.lsa.umich.edu`. This work was supported in part by NSF grant DMS-9204271 and a computer allocation at the NSF San Diego Supercomputer Center.

Pelz: Dept. of Mechanical and Aerospace Eng., Rutgers University, P.O. 909, Piscataway, NJ 08855-0909; email: `pelz@jove.rutgers.edu`.

1. Introduction

A vortex sheet in incompressible flow is a surface across which the tangential fluid velocity has a jump discontinuity. A basic idea in fluid dynamics going back to Prandtl is that the vortex sheet can be obtained as the zero viscosity limit of a sequence of smooth solutions to the Navier-Stokes equations. Thus, the investigation of vortex sheet motion may yield insight into the structure of high Reynolds number flow.

According to linear theory, a flat vortex sheet of constant strength is subject to Kelvin-Helmholtz instability. To study the nonlinear sheet motion, one must solve an integro-differential equation [3, 15]:

$$\overline{\frac{\partial z}{\partial t}}(\Gamma, t) \;=\; \int K(z(\Gamma, t) - z(\Gamma', t))\, d\Gamma' \tag{1}$$

where $z(\Gamma, t)$ is a complex-valued function representing the vortex sheet, Γ is the circulation parameter along the sheet and t is time. In the Kelvin-Helmholtz problem, the kernel is

$$K(z) \;=\; \frac{1}{2i} \cot \pi z \tag{2}$$

and the Cauchy principal value of the integral is taken in (1).

In 1979 Moore [11] found that a singularity forms in a perturbed vortex sheet at a time critical t_c. The sheet remains continuously differentiable with respect to Γ at $t = t_c$ but the curvature becomes infinite at a point. In 1983 Pullin [12] conjectured that the sheet will roll up for $t > t_c$ into a spiral with an infinite number of turns. This idea is motivated by the study of self-similar spiral vortex sheets which have a singularity present initially [13].

2. Numerical Solution

The earliest method for computing vortex sheet motion was the point vortex approximation [14]. The sheet is replaced by a set of point vortices $\{z_j, j = 1, ..., N\}$ whose motion is governed by a system of ordinary differential equations

$$\overline{\frac{dz_j}{dt}} \;=\; \sum_{k=1}^{N} K(z_j - z_k)\sigma_k \tag{3}$$

Here, σ_k are quadrature weights.

This discretization has been the subject of controversy [3], but it is now known that the point vortex approximation converges as $N \to \infty$ for $t < t_c$ [8, 6]. A different approach however is needed to compute the sheet's motion for $t > t_c$. In the vortex blob method [7, 1, 9], equation (1) is regularized by replacing the singular kernel $K(z)$ with a smooth approximation $K_\delta(z)$. The vortex sheet is obtained as the limit of solutions to the regularized equation as the smoothing parameter $\delta \to 0$. Thus, one seeks to compute solutions for a sequence of values of $\delta > 0$ and then infer properties of the limit $\delta \to 0$.

A difficulty arises in computing with small values of δ because roundoff error perturbations are amplified leading to inaccurate results. A Fourier filter can be used to overcome

this difficulty [8, 9], but the approach is limited to $t < t_c$ and to problems with periodic boundary conditions. To maintain accuracy past the critical time and for problems with general boundary conditions, the most promising approach is to use higher precision arithmetic.

3. Multiple Precision Computation Software

The multiple precision (MP) computation in this application was performed using a MP translator (TRANSMP) and a package of MP computation routines (MPFUN), both of which were developed by one of the authors (Bailey).

MPFUN routines are available to perform the four basic arithmetic operations between MP numbers, to compare MP numbers, to produce the integer and fractional parts, to produce a random MP number and to perform binary to decimal and decimal to binary conversion. Some higher level routines sort MP numbers; perform complex arithmetic; compute square roots, cube roots, n-th powers, n-th roots, and π; evaluate the functions exp, log, cos, sin, cosh, sinh, inverse cos and sin; find the real or complex roots of polynomials; and find integer relations in real vectors. For many of these functions, both basic and advanced versions are available. The advanced routines employ advanced algorithms suitable for extra high precision computation.

Conversion of a conventional scientific application program to use the MPFUN routines is generally straightforward, but it is often tedious and error prone. For example, if the slightest error is made in any of the arguments to the many subroutine calls, not only will the results be in error, but the program may abort with little information to guide the programmer. As a result of these difficulties, few serious scientific programs have been manually converted to use the MPFUN routines. Similar difficulties have plagued programmers who have attempted to use other multiprecision systems, such as Brent's package [5].

To facilitate such conversions, one of the authors (Bailey) has developed a translator program that accepts as input a conventional Fortran-77 program to which has been added certain special comments that declare the desired level of precision and specify which variables in each subprogram are to be treated as multiprecision. This translator then parses the input code and generates an output program that has all of the calls to the appropriate MPFUN routines. This output program may then be compiled and linked with the MPFUN package for execution.

This translation program allows one to extend the Fortran-77 language with the datatypes `MULTIP INTEGER`, `MULTIP REAL` and `MULTIP COMPLEX`. These datatypes can be used for integer, floating point or complex numbers of an arbitrarily high, pre-specified level of precision. Ordinary variables in the input program may be treated as multiprecision variables in the output program by placing directives (special comments) in the input file. In this way, the input file remains an ANSI Fortran-77 compatible program and can be run at any time using ordinary arithmetic on any Fortran system for comparison with the multiprecision equivalent.

This translator supports a large number of Fortran-77 constructs involving multipreci-

sion variables, including all the standard arithmetic operators, mixed mode expressions, automatic type conversions, comparisons, logical `IF` constructs, function calls, `READ` and `WRITE` statements and most of the Fortran intrinsics (i.e. `ABS, MOD, COS, EXP`, etc.). Storage is automatically allocated for multiprecision variables, including temporaries, and the required initialization for the MPFUN package is automatically performed.

4. Multiple Processor Implementation

The implementation of this application on multiple processors of an Intel iPSC/860 parallel computer started with a conventional Fortran program previously developed by two of the authors (Krasny and Pelz). To this program was added directives specifying which variables are to be treated as MP. Only 11 of these directives sufficed for the correct MP translation of the entire program file (nearly 600 lines). The resulting program was tested and certified on single processors of both a eight-processor Cray Y-MP and a 128-processor Intel iPSC/860, both of which are at the NAS computer center at NASA Ames.

Unfortunately, however, conversion of this program to MP (with 56 digit numeric precision) increases its run time by a factor of approximately 400. Such large increases are quite typical in MP computation. Even higher ratios have been reported by researchers using other packages. This ratio of 400 is exclusively due to the cost of calling multiprecision routines — the translator introduces almost no overhead. In this particular application, this large ratio means that computations of significant research interest are very much supercomputer class calculations and suggest implementation on a highly parallel scientific computer.

The Intel system is actually a very cost-effective platform for this type of computation, compared to conventional supercomputers such as Crays, for several reasons. First of all, unless very high precision is employed, the modest vector length in such computations results in poor performance on Crays, whereas the performance rates of the RISC processors are not significantly reduced. Secondly, multiprecision computations are very well localized, so that they effectively utilize the cache memories of RISC processors such as the i860 used in the Intel system. As a result of these advantages, the performance of this application on the Intel, using as few as eight nodes, exceeds that of one processor of the Cray Y-MP.

The parallel implementation of this application is greatly simplified by the fact that the entire computation requires only a very modest amount of memory. In fact, in problems attempted to far, the total memory required is less than that found on a single node of the iPSC/860 (eight megabytes). Thus all working arrays may be allocated on each node.

The distribution of the computation among multiple processors of the Intel is achieved by simply changing loops such as

```
        do 100 j = 1, nvort
          zsin = sin (con2p * c(j))
          x(j) = c(j) + amp * zsin
 100    continue
```

to

```
      do 100 j = kp + 1, nvort, np
        zsin = sin (con2p * c(j))
        x(j) = c(j) + amp * zsin
100   continue
```

where `kp` is the processor number and `np` is the total number of processors. Note that this scheme achieves an optimal load balance of the computational work in the loop.

It remains only to note those points in the program where individual processors need to share their computed results with all other processors. Such sharing of data is accomplished by using a global summation routine, which replaces the argument array in every processor with an array of the global sums over all processors. Intel provides a library routine for this purpose (`GDSUM` for DP data), but it has been found by the authors that a simple Fortran routine employing the bidirectional communication techniques of Seidel [10] and Bokhari [4] is nearly twice as fast. Furthermore, the TRANSMP translator was able to translate this routine into a MP global summation routine.

5. Results

We have obtained solutions for $\delta = 0.03$ and $\delta = 0.02$, which are significantly smaller than in previous investigations. These computer runs required 7.4 and 128.2 hours CPU time (based on 32 nodes of the Intel). A plot of the final results for $\delta = 0.02$ is shown in Figure 1. These results provide strong support for Pullin's conjecture that the sheet rolls up into a spiral for $t > t_c$.

That multiple precision arithmetic was required for these computations is indicated by the fact that in the $\delta = 0.02$ run, the first (x, y) pair, which should always be $(0, 0)$, had values roughly 10^{-35} at completion, so that over 20 digits of precision had been compromised. Clearly if ordinary 64-bit arithmetic had been used, all significance would have been lost. In this case it appears that a 56 digit precision level was generous. But for future planned computations with smaller δ, it is expected that this level will be more fully utilized.

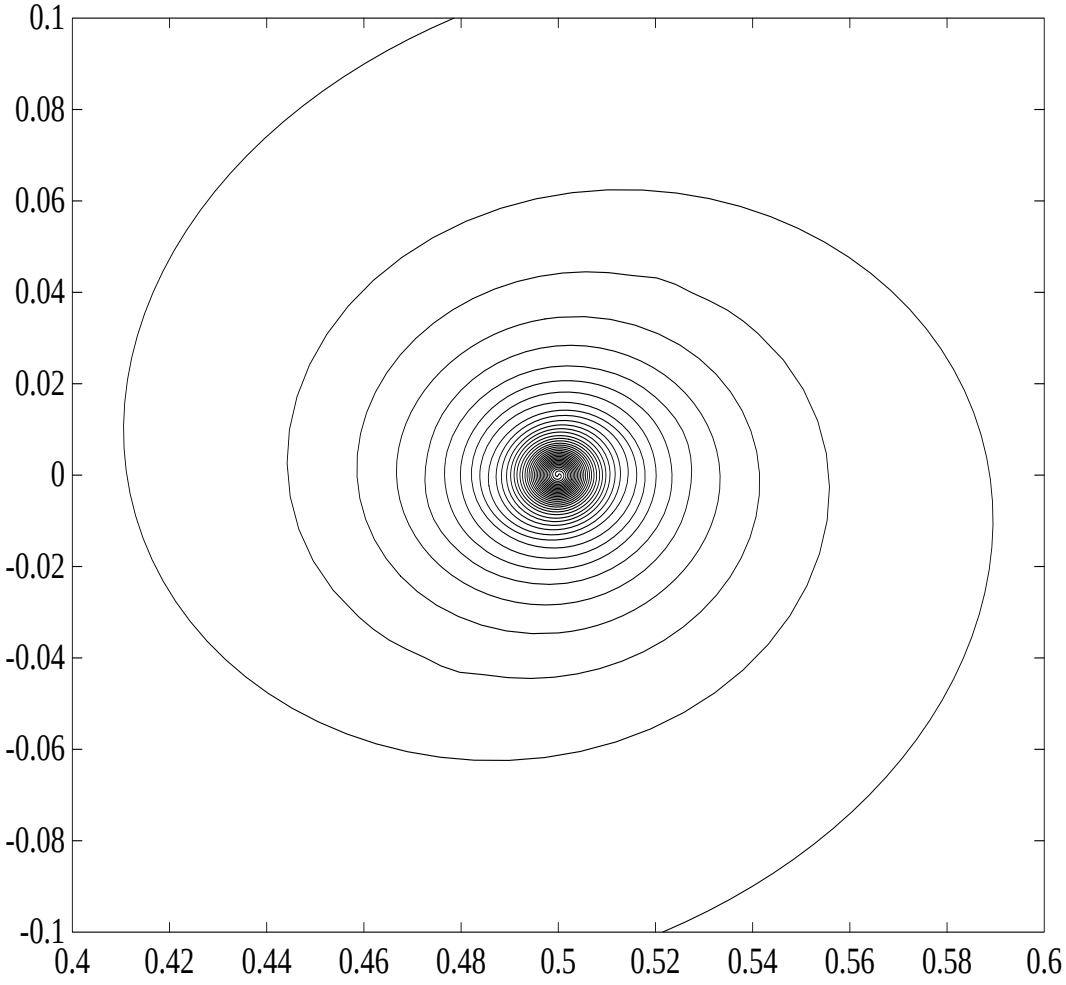

Figure 1: Final Solution for $\delta = 0.02$

References

[1] C. Anderson, "A Vortex Method for Flows with Slight Density Variations", *Journal of Computational Physics*, vol. 61 (1985), p. 417.

[2] D. Bailey, "Multiple Precision Translation and Execution of Fortran Programs", *ACM Transactions on Mathematical Software*, to appear.

[3] G. Birkhoff, "Helmholtz and Taylor Instability", *Proceedings of the Symposium of Applied Mathematics XIII*, American Mathematical Society, 1962, p. 55.

[4] S. H. Bokhari, "Complete Exchange on the iPSC-860", ICASE Report 91-4, ICASE, NASA Langley Research Center, Hampton, VA 23665, 1991.

[5] R. P. Brent, "A Fortran Multiple Precision Arithmetic Package", *ACM Transactions on Mathematical Software*, vol. 4 (1978), p. 57 – 70.

[6] R. Caflisch and J. Lowengrub, "Convergence of the Vortex Method for Vortex Sheets", *SIAM Journal on Numerical Analysis*, vol. 26 (1989), p. 1060.

[7] A. J. Chorin, and P. S. Bernard, "Discretization of a Vortex Sheet", sl Journal of Computational Physics, vol. 13 (1973), p. 423.

[8] R. Krasny, "A Study of Singularity Formation in a Vortex Sheet by the Point Vortex Approximation", *Journal of Fluid Mechanics*, vol. 167 (1986), p. 65.

[9] R. Krasny, "Desingularization of Periodic Vortex Sheet Roll-up", *Journal of Computational Physics*, vol. 65 (1986), p. 292.

[10] S. R. Seidel, M. H. Lee and S. Fotedar, "Concurrent Bidirectional Communication on the Intel iPSC/860 and the iPSC/2", Computer Science Technical Report CS-TR-90-06, Michigan Technological University, Houghton, MI 49931-1295.

[11] D. W. Moore, "The Spontaneous Appearance of a Singularity in the Shape of an Evolving Vortex Sheet", *Proceedings of the Royal Society of London*, ser. A, vol. 365 (1979), p. 65.

[12] D. I. Pullin, private communication, 1983.

[13] D. I. Pullin and W. R. C. Phillips, "On a Generalization of Kaden's Problem", *Journal of Fluid Mechanics*, vol. 104 (1981), p. 45.

[14] L. Rosenhead, "The Formation of Vortices from a Surface of Discontinuity", *Proceedings of the Royal Society of London*, ser. A, vol. 134 (1931), p. 170.

[15] N. Rott, "Diffraction of a Weak Shock with Vortex Generation", *Journal of Fluid Mechanics*, vol. 1, (1956), p. 111.