Implementation of a Multilevel Algorithm for Gas Dynamics in a High-Performance Java Dialect

Peter McCorquodale and Phillip Colella *

April 12, 1999

We discuss an implementation of a multilevel gas dynamics algorithm in Titanium, a Java-derived language for high-performance scientific computing.

1 Titanium, a High-Performance Java Dialect

The Titanium language [5] is a dialect of Java for large-scale scientific computing. Java was chosen as the base because of its widespread use and its object-oriented nature, and its safety features such as bounds checking and exception handling. The compiler translates Titanium into C, so the Java Virtual Machine is not used.

Multidimensional arrays are one of the most useful extensions of Java in Titanium. These are indexed by *points*, which are tuples of integers, and their index sets are rectangular *domains*, or sets of points. The following fragment shows how a two-dimensional array can be constructed.

```
Point<2> l = [21, 10], u = [30, 20];
RectDomain<2> Dom = [l : u];
double [2d] L = new double[Dom];
```

The `foreach` statement allows one to operate on the elements of a domain. The iteration order is not specified.

```
Point<2> dx = [1, 0], dy = [0, 1];
foreach (p in L.domain()) {
   L[p] = field[p + dx] + field[p - dx] + field[p + dy] + field[p - dy]
        - 4.0 * field[p];
}
```

`Point<`N`>` and `RectDomain<`N`>` are also examples of *immutable* classes in Titanium. These are classes whose objects are passed by value, as with C structs, instead of by reference, as other Java objects are passed. Immutable classes cannot be extended or be extensions of another class. User-defined immutable classes, such as a `Complex` class, can improve performance by reducing overhead.

Titanium does not incorporate Java threads. Parallelism in Titanium is based on an SPMD model of computation with a global address space. There is a *broadcast* method for one-to-all communication and an *exchange* method for all-to-all communication. Titanium runs on top of a Posix thread library on SMPs and an Active Message [3] layer on distributed-memory multiprocessors.

*Applied Numerical Algorithms Group, Lawrence Berkeley National Laboratory, Berkeley, CA. {`PWMcCorquodale, PColella`}`@lbl.gov`

LevelAdvance$(\ell, t^\ell, \Delta t^\ell)$

    Initial update of U^ℓ:

        $U^\ell(t^\ell + \Delta t^\ell)_{i,j} = U^\ell(t^\ell)_{i,j} - \Delta t^\ell D^\ell(\vec{F}^\ell)$

    Initialize/update Flux Registers:

        if $(\ell < \ell_{max})$ set $\delta F^{\ell+1}$ using $\vec{F}^\ell$

        if $(\ell > 0)$ adjust δF^ℓ using $\vec{F}^\ell$

    Recursive calls to LevelAdvance for next finer level:

        if $(\ell < \ell_{max})$ **then**

            for $n = 0, n^\ell_{ref} - 1$

                $\Delta t^{\ell+1} = \Delta t^\ell / n^\ell_{ref}$

                $t^{\ell+1} = t^\ell + n\Delta t^{\ell+1}$

                LevelAdvance$(\ell + 1, t^{\ell+1}, \Delta t^{\ell+1})$

            end for

            Synchronize level ℓ with level $\ell + 1$:

                Reflux $U^\ell(t^\ell + \Delta t^\ell)$ using $\delta F^{\ell+1}$

                Average $U^{\ell+1}(t^\ell + \Delta t^\ell)$ to $U^\ell(t^\ell + \Delta t^\ell)$

        end if

end LevelAdvance

Figure 1: Pseudocode for recursive timestep used for hyperbolic conservation laws

2 Multilevel Gas Dynamics Algorithm

We implemented the adaptive mesh-refinement algorithm of Berger and Colella [1] in Titanium. A hyperbolic conservation law of the form

$$u_t + f(u)_x + g(u)_y = 0 \qquad (1)$$

is solved by local refinement in time and space. The multilevel algorithm is generalized for any number of levels by defining it as a series of recursive single-level advances. The function *LevelAdvance*$(\ell, t^\ell, \Delta t^\ell)$, as shown in Figure 1, advances the solution U^ℓ at level ℓ from time t^ℓ to time $t^\ell + \Delta t^\ell$.

At the beginning of *LevelAdvance*, the flux F^ℓ is computed and U^ℓ is updated with the divergence $D^\ell(F^\ell)$.

When $t^{\ell+1}$ has reached t^ℓ, the level-ℓ solution must be corrected by replacing the level-ℓ flux at boundaries by the sum of the fluxes at level $\ell + 1$. For this reason, the *flux registers* δF^ℓ are kept at the boundaries with level $\ell + 1$.

3 Results

We ran a parallel single-grid code for the shock-ramp problem of Berger and Colella [1, 2, 4]. In this problem, a Mach-10 shock impinges upon a solid surface at an angle 30° from the vertical. The size of the domain is 1.0×0.25 and the grid used is 128×32 divided into four 32×32 sections. Figure 3 shows a contour plot of the density at time $t = 0.042$, after 133 timesteps.

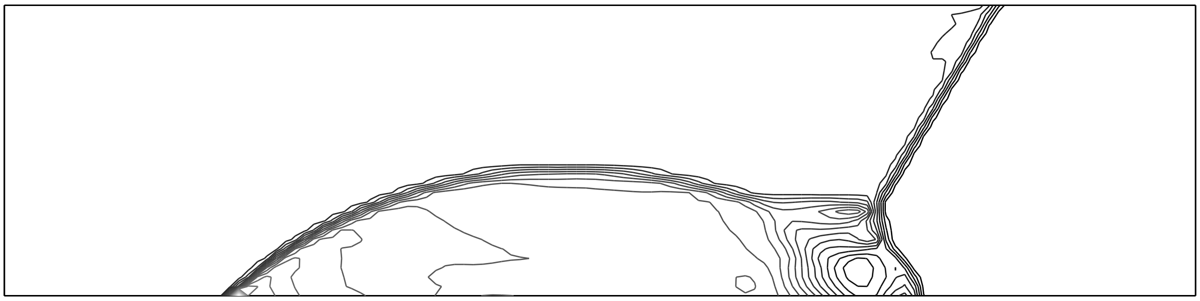

Figure 2: Contour plot of density for the shock-ramp problem at time $t = 0.042$.

References

[1] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82:64–84, 1989.

[2] H. M. Glaz, P. Colella, I. I. Glass, and R. L. Deschambault. A numerical study of oblique shock wave reflections with experimental comparisons. *Proc. Royal Soc. London A*, 398:117–140, 1985.

[3] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauer. Active messages: a mechanism for integrated communication and computation. In *Proc. of the 19th Int'l Symposium on Computer Architecture*, May 1992.

[4] P. Woodward and P. Colella. The numerical simulation of two-dimensional fluid flow with strong shocks. *J. Comput. Phys.*, 54:115–173, 1984.

[5] K. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. Hilfinger, S. Graham, D. Gay, P. Colella, and A. Aiken. Titanium: A high-performance Java dialect. In *ACM Workshop on Java for High-Performance Network Computing*, February 1998.