A DISTRIBUTED PARALLEL ALGORITHM FOR THE NONLINEAR EVOLUTION OF UNSTABLE DISCRETE PLASMA MODES

J. Candy and B. Demsky

Institute for Fusion Studies, University of Texas at Austin
Austin, Texas 78712, USA

Introduction Software to enable the parallelization of specific, previously-existing particle codes (for the bump-on tail [1], TAE [2] and fishbone [3] instabilities) has been developed. The computation is managed by a central server, executing multithreaded Java code, that distributes the workload in an asymmetric manner to a collection of N clients (workstations of nonspecific platform). Each client performs a fraction of the computation using an identical code written in `c++`. For sufficiently large particle number, the performance scales linearly with total collective system FLOPS (floating point operations per second). The efficiency of the distributed time-stepping algorithm relies on the rather special nature of the energetic particle nonlinearity. We show detailed simulation results for a novel problem: the spontaneous formation of phase space holes and clumps in a weakly unstable 2-D plasma [4].

Energetic Particle Phase Space We are generally interested in the evolution of discrete bulk plasma modes interacting resonantly with a population of energetic particles. The unperturbed motion of energetic particles (and thus each "bit" of their phase space) is Hamiltonian. The unperturbed equations of motion are first-order in time: $\dot{\Gamma}_i(t) = h\,[\Gamma_i(t)]$.

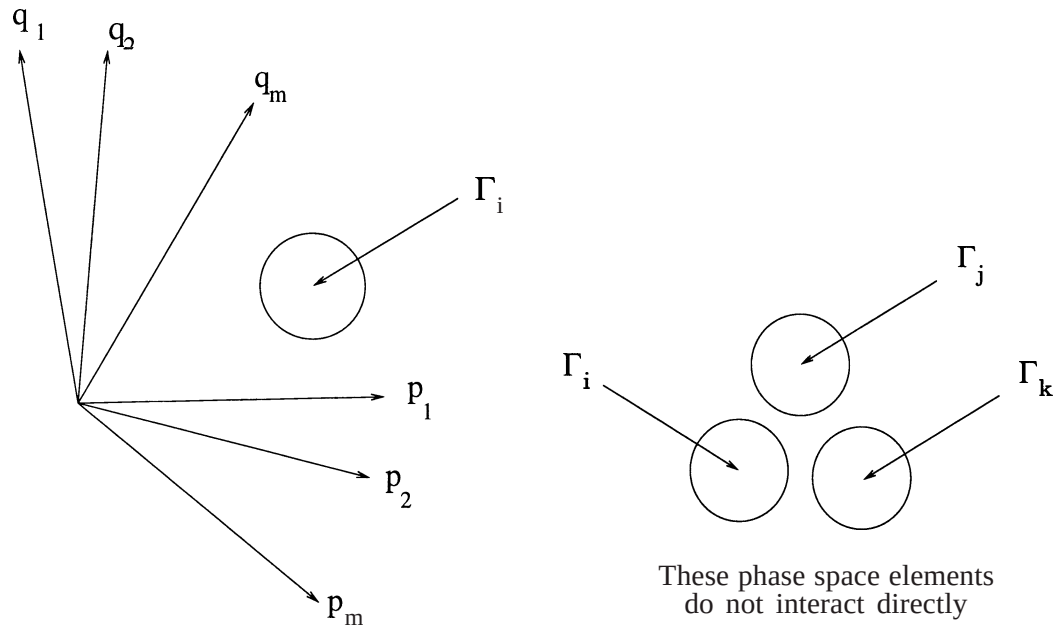

Fig. 1. Individual cells of phase space follow a Hamiltonian flow. In the absence of a perturbation, they are noninteracting.

Interaction with Collective Modes Collective modes evolve (grow exponentially, saturate, chirp, explode) as a consequence of the interaction with resonant energetic particles. This interaction has a special form for a variety of problems in weak turbulence (TAE, Fishbone, bump-on-tail):

$$\dot{\phi}_k(\varepsilon t) = \int d\Gamma\, \delta f\, w(\Gamma, t) \to \sum_i \Delta\Gamma_i\, \delta f_i\, w_i$$

Here, w is a problem-dependent function which determines the specific interaction, while δf satisfies a nonlinear kinetic equation

$$\dot{\delta f} = -\dot{\Gamma}\, \frac{\partial F_0}{\partial \Gamma}$$

It is essential that the exact trajectories be used when solving the kinetic equation. These trajectories are still Hamiltonian (with wave degrees of freedom considered as explicit function of time): $\dot{\Gamma}_i(t) = h[\Gamma_i(t), \{\phi_k(\varepsilon t)\}]$. Thus, phase space elements do not interact directly, but stream about in phase space in a nearly free manner. Their trajectories are perturbed by a wave which evolves slowly by comparison.

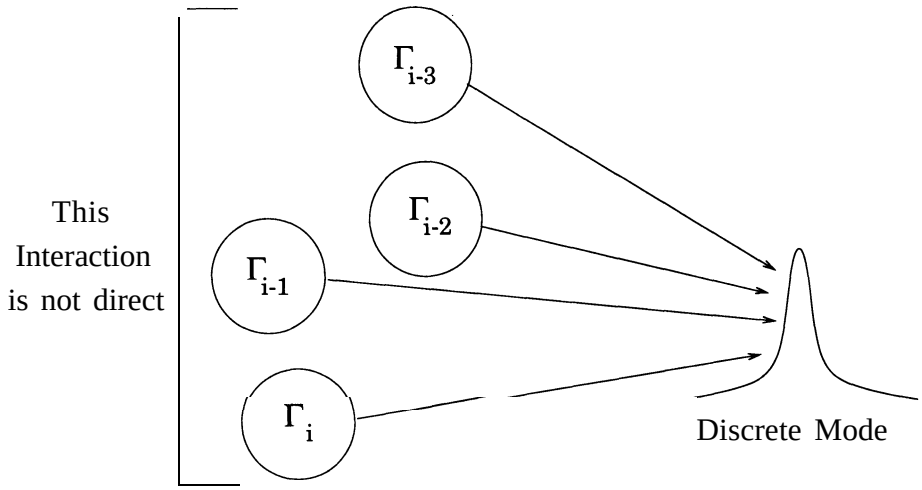

Fig. 2. Indirect interaction of phase space cells via unstable discrete mode.

A suitable high-order method can then be used to integrate each $\Gamma_i(t)$ forward to $\Gamma_i(t+\Delta t)$. In this "particle update", each $\{\phi_k(\varepsilon t)\}$ is fixed. Although additional complications arise for fishbones, this essential feature is unchanged.

Parallelization of Trajectory Calculation Since the $\{\phi_k(\varepsilon t)\}$ are fixed and known a priori, the update of Γ_i can be distributed among a pool of connected clients – with each client benchmarked on startup. The jth client is assigned a suitable fraction of the total number of particles, n, according to the offset $c(j)$: The jth client computes its share of the particle trajectories (determined by results from a benchmark on startup)

$$\{\Gamma_i(t + \Delta t)\} \quad \text{for} \quad i = n_{j-1}, \ldots, n_j - 1.$$

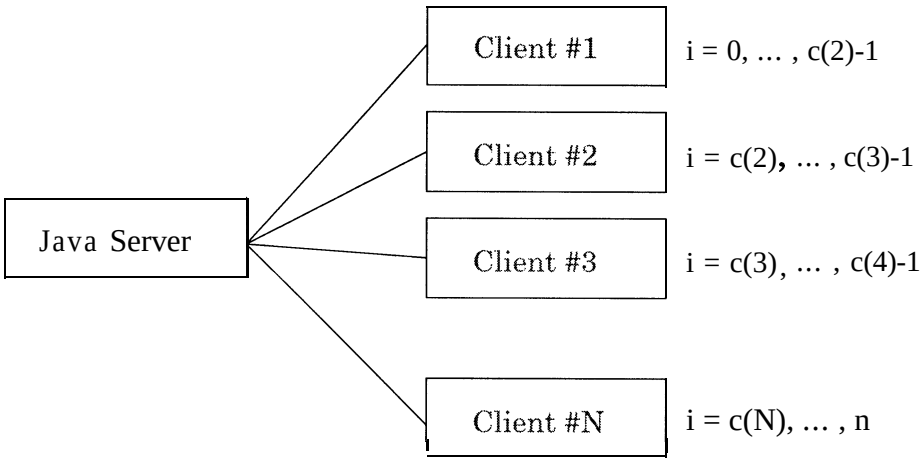

Fig. 3. Client-server architecture, showing distribution of client workload.

The server does not care about this information; it is *local.* The *j*th client must also compute the partial wave sums

$$\phi_k^{(j)}(t + \Delta t) = \phi_k^{(j)}(t) + \delta t \sum_{\{i\}} d\Gamma_i \, \delta f_i \, w_i$$

The server cares about this information; it is *global.* In fact, the server must wait for all clients in the pool to return these partial sums. Then, the server can compute the global sum (a virtually insignificant fraction of the computational overhead) and give the unique result back to each client.

Sample Parallel Efficiency Neglecting time for I/O, a single machine operates with 100% efficiency. Were the benchmarking perfect, the latency due to distribution/communication would be negligible. For the unsophisticated benchmark currently implemented, this is not the case. The worst possible efficiencies are obtained using different CPUs and different `c++` compilers. A 400,000 particle simulation of the two-species bump-on-tail problem with 3,200,000,000 total 4th-order integrator steps (`g++ -02` compilation) gave:

machine	CPU	Clock	efficiency
zuni	P-MMX	233MHz	100 %
yuma	P-MMX	233MHz	99.9 %
kendall	P-II	266MHz	92.7 %
pauling	P-II	266MHz	92.7 %
born	P-II	266MHz	91.9 %
milou	AMD	200MHz	88.9 %
farside	P-pro	200Mhz	71.8 %

Table 1. Relative efficiency (CPU utilization) for 7-machine distributed simulation.

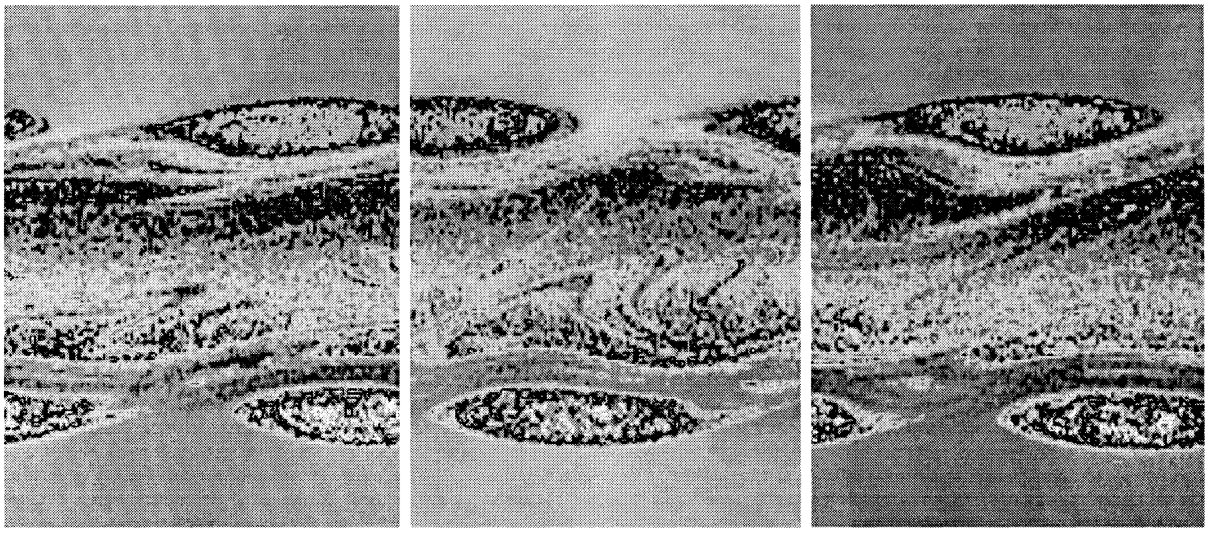

Fig. 5. 500,000 particle simulation showing formation of phase space hole and clump. Unstable distribution of light plasma particles damps nonlinearly on stable distribution of heavy particles. Three nearby time slices of (x,v) phase space for unstable distribution are shown.

The code hook is accomplished through the routine communicate. It is simple and can be added easily to an existing scalar code:

```
for (k = 0; k < c.nwave; k++)
{
      // Send local partial sums cc,ss to the server.
      // Receive completed sum.
      communicate (cc [k] , ss [k] , k) ;
      a[k] .x +=-cc[k]*h.h;
      a[k] .y += ss[k]*h.h;
}
```

Listing 1. Section of code which exchanges data, via `communicate`, with Java server.

Additional start-up calls to the Java server are required to initialize threads. These also have the same form for all weak-turbulence applications. The UNIX `tar` archive bump. `tar` contains a scalar version of the bump-on-tail code, and is available via anonymous FTP from `ftp://mildred.ph.utexas.edu/pub`. The input GUI requires Tcl/Tk and the post-processing GUI requires IDL.

References

[1] J. Candy: J. Comput. Phys. 129 (1996) 160.
[2] J. Candy, D.N. Borba, G. Huysmans, W. Kerner and H.L. Berk: Phys. Plasmas **4** (1997) 2597
[3] J. Candy, F. Porcelli, B.N. Breizman and H.L. Berk: *Proc. 24th EPS Conf.*, Berchtesgaden, Vol **2lA** III (1997) 1189.
[4] H.L. Berk, B.N. Breizman and N.V. Petviashvili: Phys. Lett. A **234** (1997) 213.
[5] H.L. Berk, B.N. Breizman, J. Candy and N.V. Petviashvili: *submitted to* Phys. Plasmas