PARTICLE-IN-CELL SIMULATION CODES IN HIGH PERFORMANCE FORTRAN

Erol Akarsu,Kivanc Dincer,Geoffrey Fox, Tom Haupt
Northeast Parallel Architectures Center
at
Syracuse University

November 20, 1996

Outline

- **Main Phases of PIC codes**
- **Approach to Parallelization**
- **Implementation Details**
- **Performance Results**
- **Conclusion**

Main Phases of PIC codes

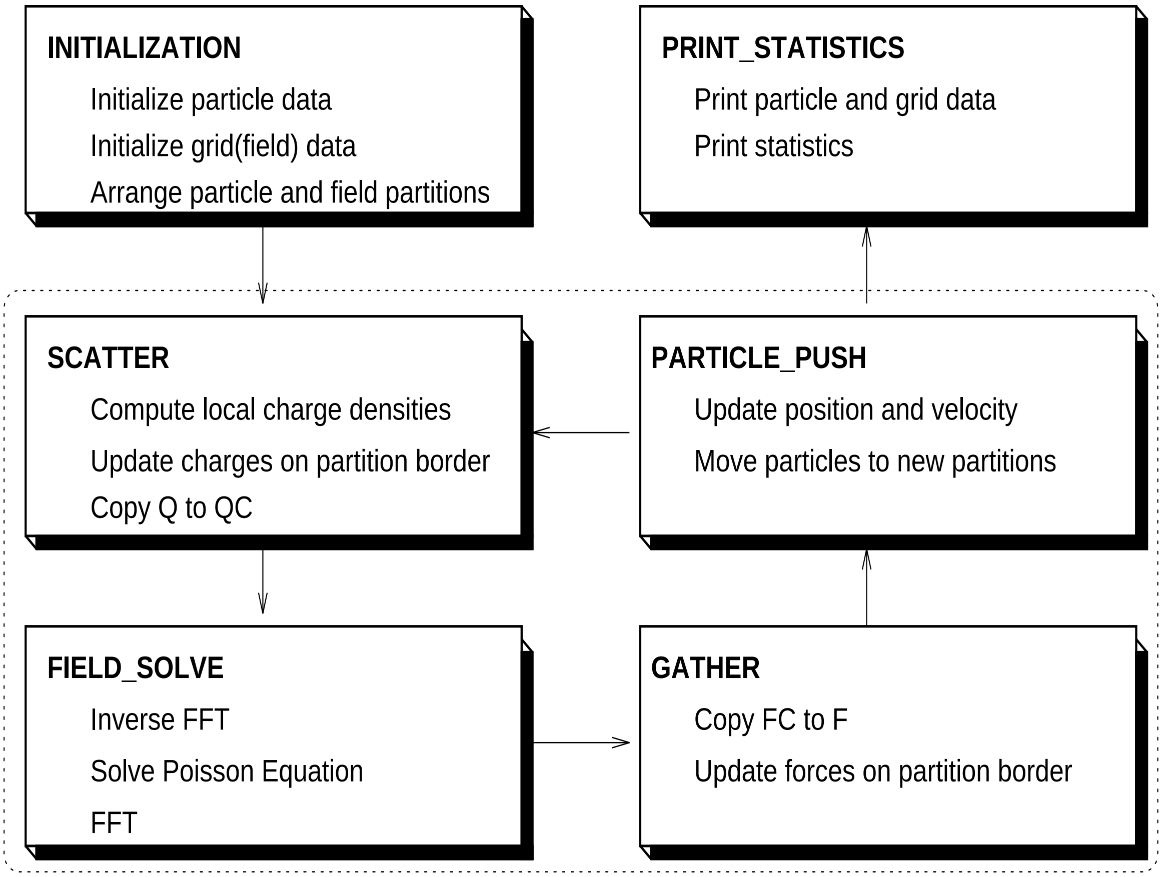

Figure 1: Main phases of a typical PIC code.

Approach to Parallelization

- **Decomposition of Principal Data Structures**
- **Loop Level Parallelization(FORALL and INDEPENDENT DO)**
- **Communication Optimizations(CSHIFT)**
- **Passing Distributed Data to Subroutines(Descriptive Directive)**
- **Extrinsic Procedures**
- **Compiler Restrictions(Nonlinear Dependency)**

$$-E_l^i = -j\omega A_l - \frac{\partial \phi}{\partial l} \; on \; S \tag{1}$$

$$A = \mu \int_{axis} I(l) \frac{e^{-jkR}}{4\pi R} dl \tag{2}$$

$$\phi = 1/\epsilon \int_{axis} \sigma(l) \frac{e^{-jkR}}{4} dl \tag{3}$$

$$\sigma = \frac{-1}{j\omega} \frac{dI}{dl} \tag{4}$$

Decomposition of Data Structures

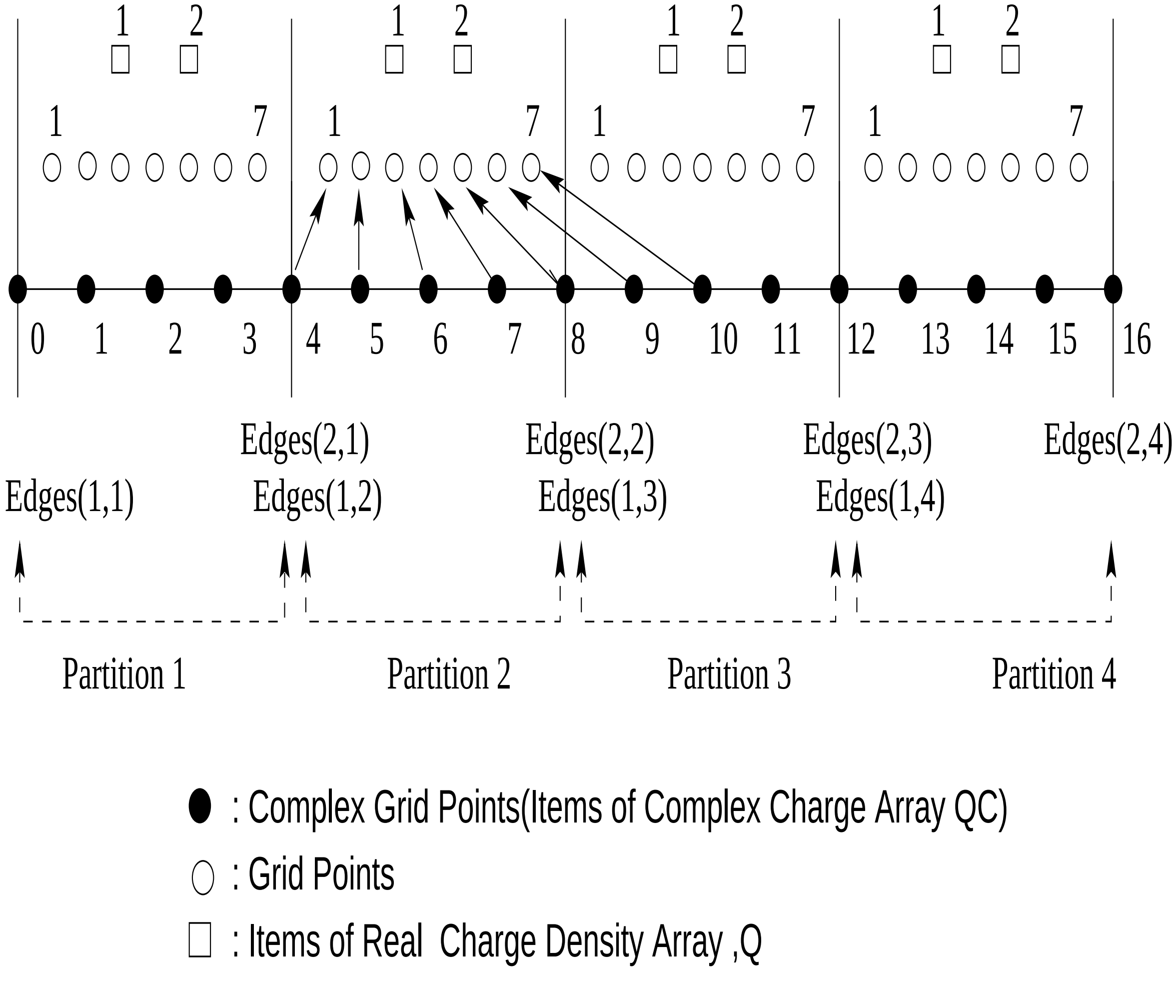

Figure 2: Illustration of particle and grid decompositions (Eulerian Decomposition) and associated charge densities at each grid point and interaction among them.

- **Particle Array (and Particle Partition(PP))**
- **Grid Array (and Field Partition(FP))**
- **Guard Points**

Extrinsic Procedures

We wrote :

- **Fast Fourier Transform (FFT)**
- **Each Nested Loop**

as Extrinsic Subroutine

The version of HPF compiler was not able to parallelize some loops in PIC code. Therefore we used extrinsic routines to make these type of loops parallel.

Implementation Details

- **How we wrote down extrinsic functions**
- **Scatter Phase**
- **Field Solve Phase**
- **Gather Phase**
- **Particle Push Phase**

How extrinsic functions was written

We distributed particles and fields onto processors statically. **PART** holds the information of particles and Q the charge densities on grid points.

- DIMENSION PART(DIM=2,NUMBER_OF_PARTICLES, NUMBER_OF_BLOCKS)
- DIMENSION Q(NUMBER_OF_GRID_POINTS,NUMBER_OF_BLOCKS)

We distributed these arrays block-wise in the last dimension. That is ,

- **!HPF$ DISTRIBUTE (*,BLOCK) :: Q**
- **!HPF$ DISTRIBUTE (*,*,BLOCK) :: PART**

Here are how we constructed our extrinsic functions:

- **Analyze the nested loop to find the read & written variables in the loop**
- **Build one extrinsic function whose body is the body of loop and whose parameters are those found at previous step**

ONE EXTRINSIC FUNCTION EXAMPLE

Here is one nested loop to compute charge densities on grid points:

```
DO k = 1, number_of_particle_partitions
      DO i = 1, number_of_particles_in_partition
            dx = part(POS,j,k) -⌊part(POS,j,k) + 0.5⌋
            n = ⌊ part(POS,j,k) +0.5 ⌋ -noff(k) +2
            q(n, k) = q(n,k) + qm*(0.75-dx)^2
            q(n+1, k) = q(n+1,k) +0.5*qm*(0.5+dx)^2
            q(n-1, k) = q(n-1,k) +0.5*qm*(0.5-dx)^2
      END DO
END DO
```

Figure 3: Calculating the particle charge densities at each grid point.

Here is the extrinsic function to compute above nested loop in parallel:

```
Subroutine Scatter1(part,q,noff)
          Dimension part(idimp,npmax,nblok),q(nxpmx,nblok),noff(nblok)
!HPF$ Distribute *(*,*,block) :: part
!HPF$ Distribute *(*,block) :: q
!HPF$ Distribute *(block) :: noff
!HPF$ INDEPENDENT, NEW(k,i,n, dx)
k = 1
DO i = 1, number_of_particles_in_partition
          dx = part(POS,j,k) -⌊part(POS,j,k) + 0.5⌋
          n = ⌊ part(POS,j,k) +0.5 ⌋ -noff(k) +2
          q(n, k) = q(n,k) + qm*(0.75-dx)^2
          q(n+1, k) = q(n+1,k) +0.5*qm*(0.5+dx)^2
          q(n-1, k) = q(n-1,k) +0.5*qm*(0.5-dx)^2
EDNDO
```

Figure 4: Calculating the particle charge densities at each grid point.

Scatter Phase

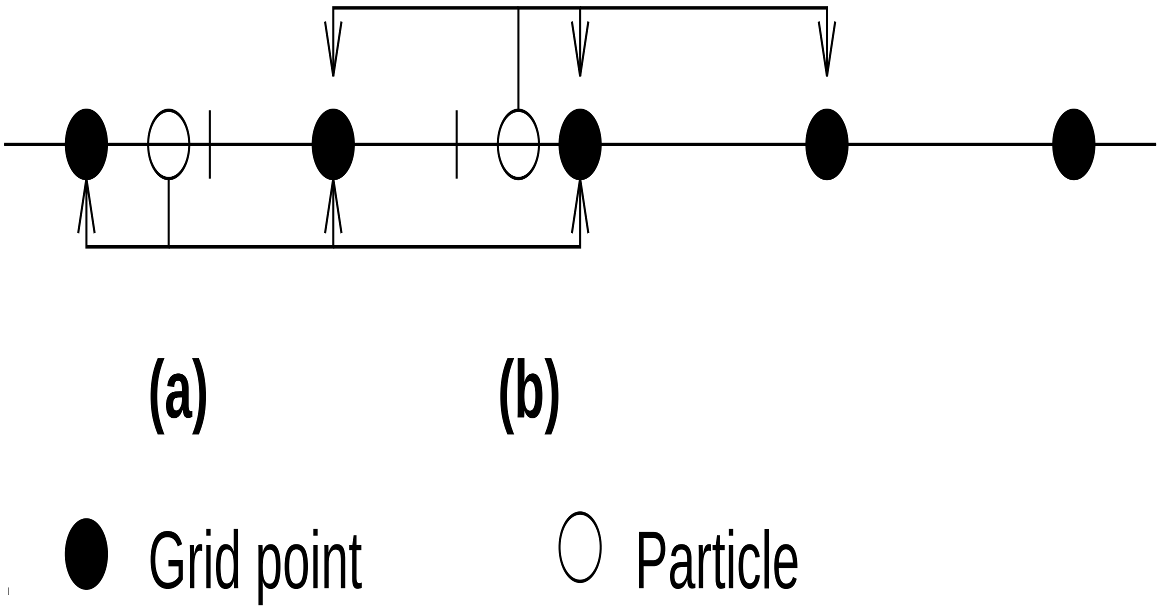

Figure 5: For a one-dimensional simulation, interaction of a particle (a) on the lower half of a grid interval; (b) on the upper half of a grid interval with the neighboring grid points.(by Inpedendent Do or Extrinsic function)

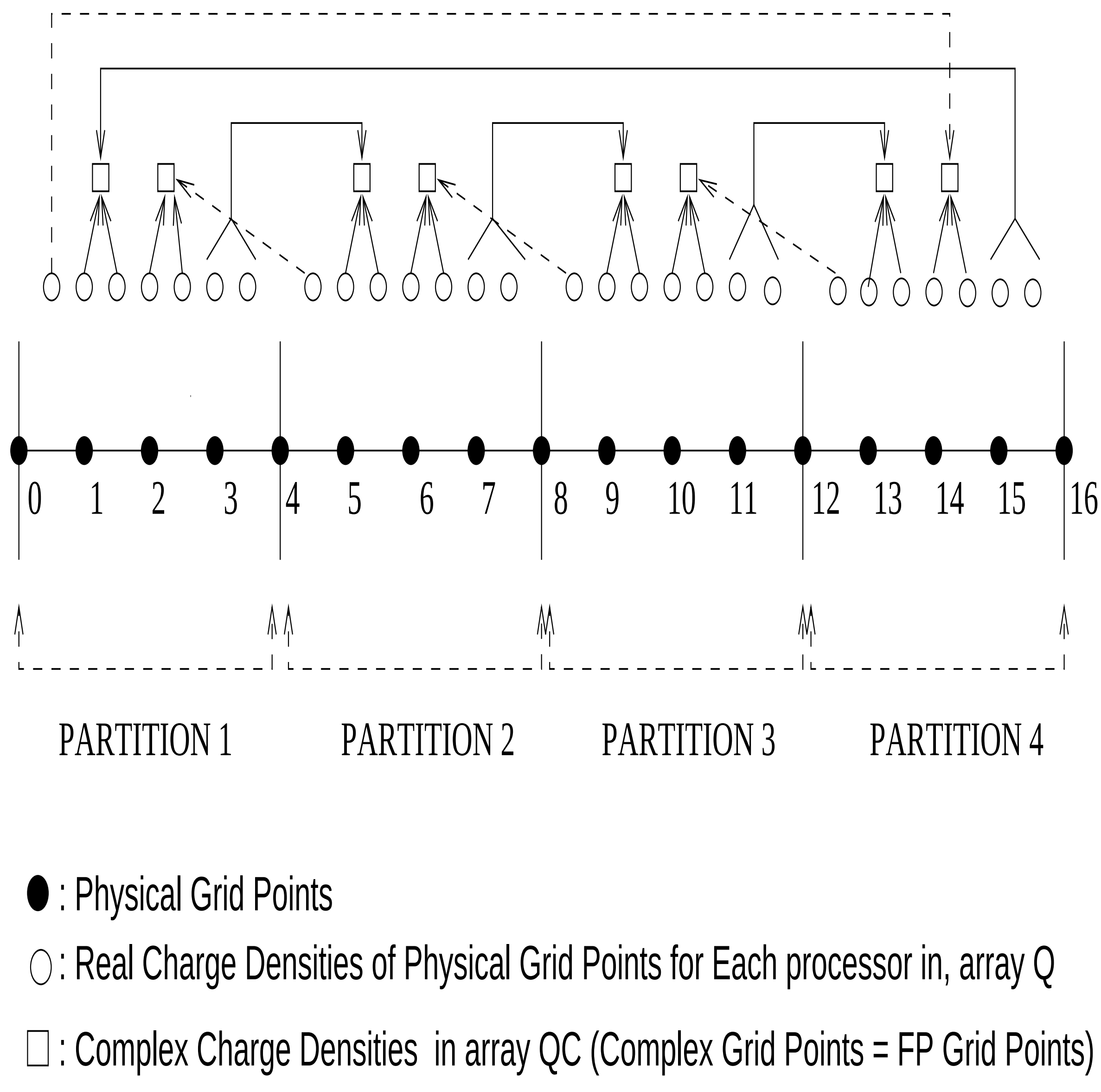

Figure 6: Copying data from real charge density to complex charge density. (by Inpedendent Do or Extrinsic Function)

Particle Push Phase

- **Each processor fills its send buffers,SbufR and SbufL**
- **Each processor also calculates the number of outgoing particles in each send buffer**
- **Using CSHIFT function,shift these send buffers and number of outgoing particles to neighbors**
- **Each processor deposits the appropriate particles with position between its boundaries ,if any,into its local data and constructs again from others ,if any, the send buffers from further going particles**
- **Repeat above cycle until each particle reaches into appropriate region**

Above steps,except third one, can be calculated in local.

- **We wrote extrinsic function for each step above,except third one**
- **For third one, we use CSHIFT function to shift trough third dimension**

Phase	Subphase	Number of Processors					
		1	2	4	8	16	32
Scatter	Phase 1	2.910	1.450	0.730	0.360	0.188	0.094
	Phase 2	0.001	0.004	0.003	0.003	0.005	0.007
Field Solve	IFFT	0.123	0.075	0.045	0.030	0.023	0.024
	Poisson	0.003	0.003	0.003	0.002	0.003	0.004
Gather	Copy	0.001	0.004	0.003	0.003	0.005	0.007
Particle Push	Push	4.880	1.930	0.970	0.480	0.248	0.136
	Migrate	0.000	0.800	0.410	0.210	0.115	0.070
Loop Iteration Time		1818.0	999.0	513.0	261.0	144 .0	94.5

Table 1: Execution times(in seconds) for 1D PIC code in HPF Number of Particles = 4M Number of Grid Points = 16K .

Phase	**Subphase**	**Number of Processors**					
		1	2	4	8	16	32
Scatter	Phase 1	2.910	1.450	0.730	0.360	0.189	0.090
	Phase 2	0.001	0.004	0.003	0.003	0.005	0.007
Field Solve	FFT	0.123	0.075	0.029	0.030	0.029	0.009
	Poisson	0.003	0.003	0.003	0.002	0.005	0.004
Gather	Copy	0.001	0.004	0.003	0.003	0.005	0.007
Particle Push	Push	4.880	1.930	0.980	0.490	0.250	0.136
	Migrate	0.00	0.81	0.40	0.20	0.10	0.07
Loop Iteration Time		1818.0	1004.0	505.6	263.8	132.1	85.5

Table 2: Execution times(in seconds) for 1D PIC code in MPL Number of Particles = 4M Number of Grid Points = 16K .

Phase	**Subphase**	**Number of Processors**				
		1	4	8	16	32
Scatter	Phase 1	5.450	1.300	0.650	0.329	0.167
	Phase 2	0.0009	0.004	0.004	0.004	0.005
Field Solve	FFT	0.005	0.015	0.009	0.009	0.015
	Poisson	0.009	0.003	0.003	0.003	0.004
Gather	Copy	0.0009	0.0040	0.0040	0.0040	0.0050
Particle Push	Push	8.150	1.890	0.950	0.486	0.251
	Migrate	0.000	0.380	0.230	0.178	0.204
Loop Iteration Time		4453.8	1196.0	624.0	325.0	24 7.0

Table 3: Execution times(in seconds) for 2D PIC code in HPF Number of Particles = 3M Number of Grid Points = 128 * 256 .

Phase	**Subphase**	**Number of Processors**					
		1	2	4	8	16	32
Scatter	Phase 1	5.45	2.66	1.32	0.66	0.33	0.15
	Phase 2	0.000	0.000	0.009	0.009	0.004	0.005
Field Solve	FFT	0.005	0.025	0.009	0.009	0.009	0.015
	Poisson	0.009	0.009	0.009	0.009	0.003	0.004
Gather	Copy	0.000	0.009	0.009	0.009	0.004	0.005
Particle Push	Push	8.150	3.820	1.880	0.940	0.479	0.250
	Migrate	0.000	0.689	0.370	0.180	0.100	0.200
Loop Iteration Time		4453.8	2368.6	1206.5	614.2	303.0	237.9

Table 4: Execution times(in seconds) for 2D PIC code in MPL Number of Particles = 3M Number of Grid Points = 128 * 256 .

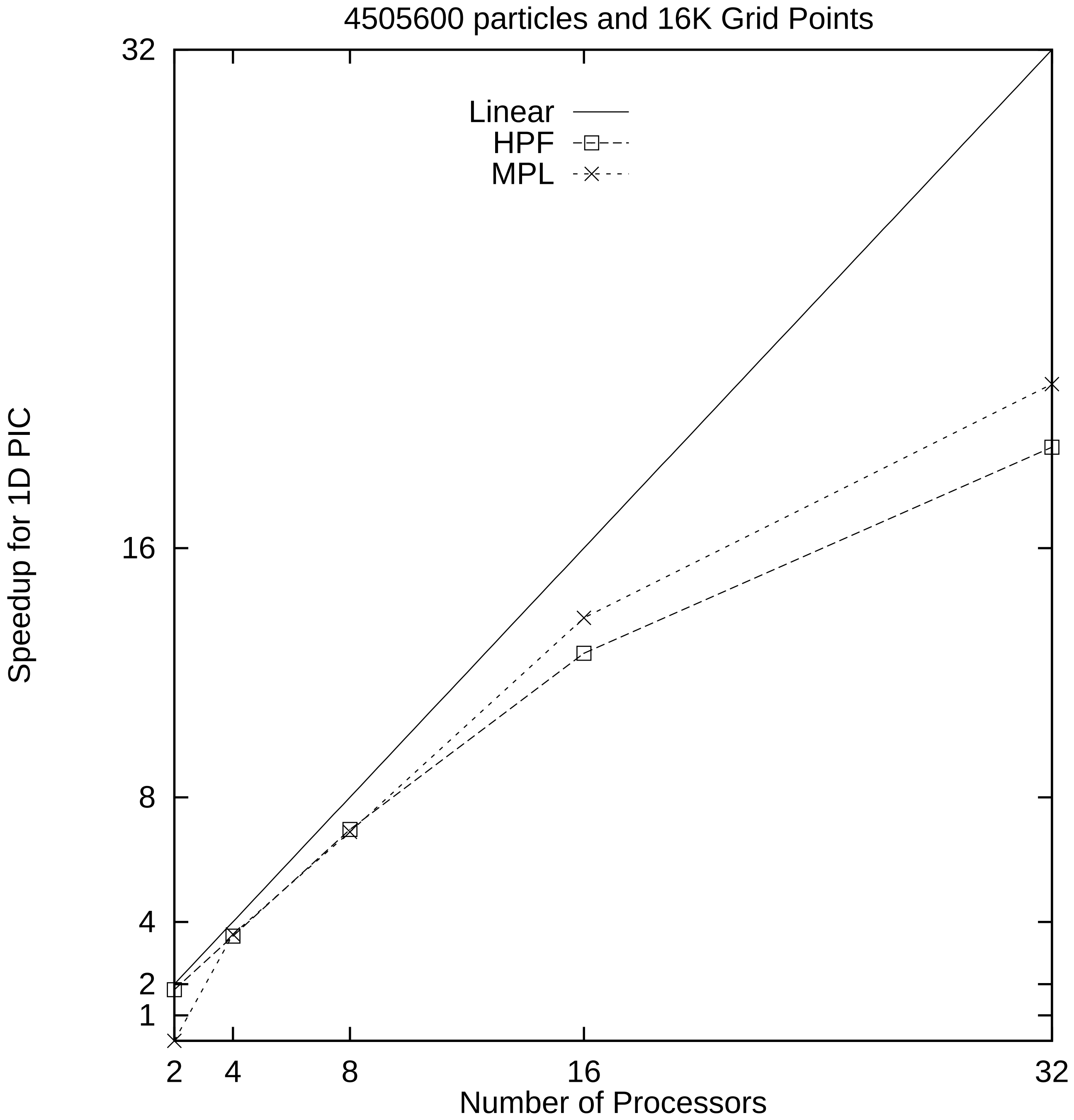

Figure 7: Speedup Comparison of 1D MPL & HPF PIC Codes

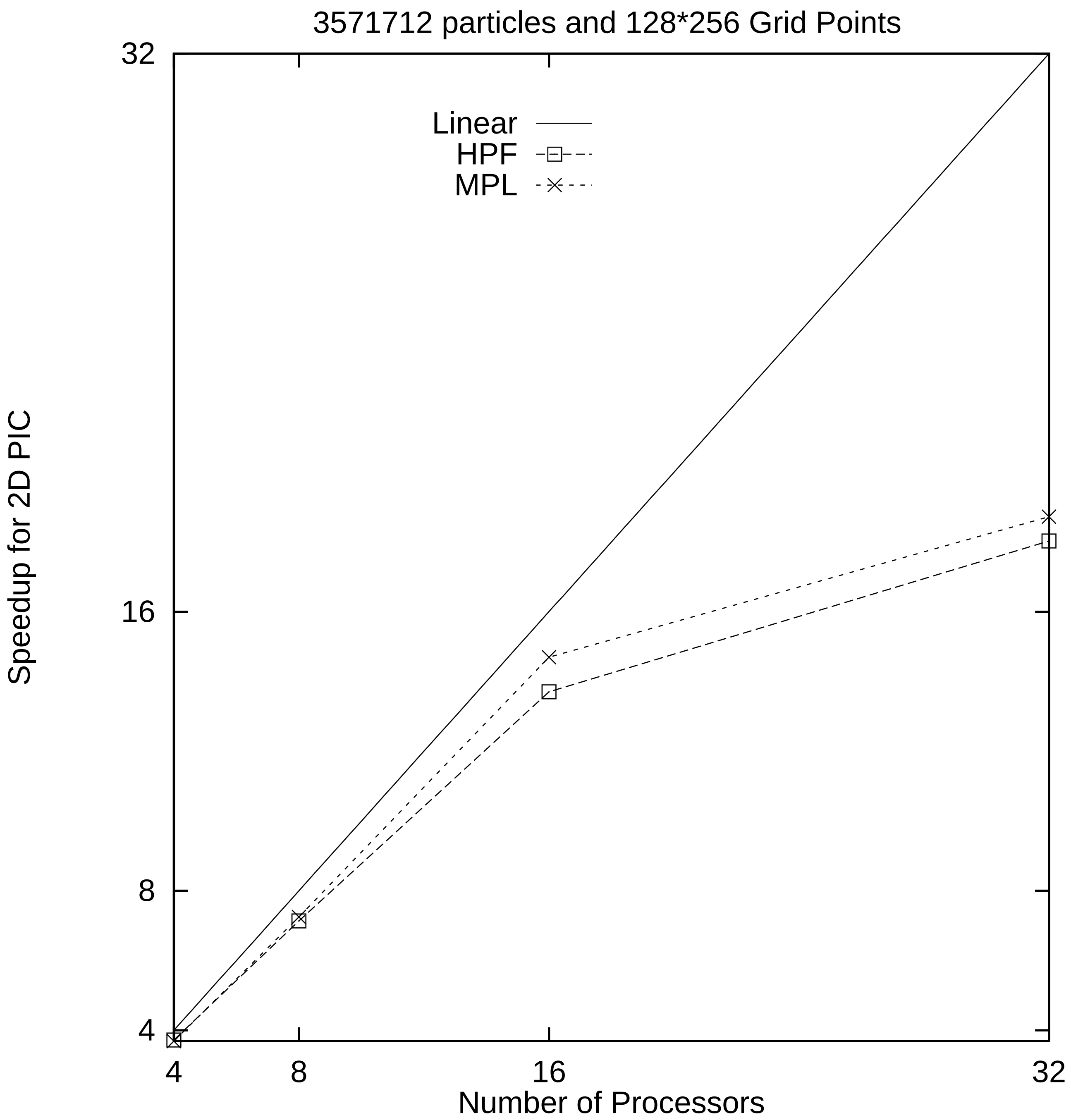

Figure 8: Speedup Comparison of 2D MPL & HPF PIC Codes

Performance Results

We got approximately same timing results & speedups as MPL version of PIC code with HPF implementation.

If we looked the tables following, every subroutine of PIC code shows a near-linear speedup.

CONCLUSION

- **We implemented PIC which is grand-challenge in HPF successfully**
- **If the HPF compilers could support independent loops properly, then we could replace these extrinsics with them**
- **We got linear speedup from our application**
- **HPF can compete with Message Passing Implementations**