University
of Southampton

Porting the GENESIS Benchmarks to HPF

Bryan Carpenter and Vladimir Getov

HPC Centre
University of Southampton
Southampton SO9 5NH
United Kingdom

dbc@ecs.soton.ac.uk

The Genesis benchmarks

- Developed in ESPRIT project 2702—GENESIS
- Designed to evaluate performance of distributed-memory MIMD computers applied to scientific and engineering problems.
- Contributors include Liverpool and Southampton Universities, PALLAS GmbH, the European Centre for Medium Range Weather Forecasting, Universitat Politechnica de Catalunya.
- Suite currently distributed by Southampton HPC Centre

Codes written in standard Fortran 77 + explicit message-passing via calls to portable communication libraries—originally PARMACS, later PVM.

HPF versions of benchmarks

A subset of benchmarks were translated to HPF data-parallel style, eliminating explicit message-passing:

trans1: Communication benchmark involving transposition of a distributed matrix.

pde1: Solver for the Three-dimensional Poisson equation, using red-black relaxation.

fft1: Fast Fourier Transform.

pde2: Solver for the Two-dimensional Poisson equation, using a recursive “Multigrid” algorithm.

md1: Molecular dynamics program, using the “Particle-in-Cell” method.

Policies

Most benchmarks be extensively rewritten to make them data-parallel.

For maximum portability we avoided using

- Features not in the HPF official Subset.
- HPF language extensions which are not are part of standard Fortran 90—viz, FORALL

In the absence of FORALL, all explicit parallelism obtained through Fortran 90 array syntax.

availability

- Versions of HPF codes are in release 3.0 of the GENESIS benchmarks.
- *Correct* versions are included in the Southampton HPF beta release.

The codes. *trans1*

Transposes a square matrix distributed blockwise on a grid of processors. The HPF code is essentially

```
!HPF$ PROCESSORS GRID(SIDE,SIDE)

      DOUBLE PRECISION DATA(MAXD,MAXD)
!HPF$ DISTRIBUTE DATA(BLOCK,BLOCK) ONTO GRID

      ! ...

         DATA = TRANSPOSE(DATA)
```

pde1

Solves the Poisson equation on a 3-dimensional grid.

Original message-passing code used successive over-relaxation (SOR) with Chebyshev acceleration. HPF version uses pure red-black relaxation—avoids clutter but retains the relevant features of the original program.

Algorithm sweeps repeatedly through the lattice replacing the value at site by the average of values of neighbours. Convergence improved by doing sweep in two phases; first update all odd sites (“red squares”), then even sites (“black squares”).

pde1 code

```
      DOUBLE PRECISION U(NX,NY,NZ), F(NX,NY,NZ)
      LOGICAL RED(NX,NY,NZ)

!HPF$ INHERIT RED
!HPF$ ALIGN U(I,J,K) WITH *RED(I,J,K)
!HPF$ ALIGN F(I,J,K) WITH *RED(I,J,K)
!HPF$ DISTRIBUTE RED *(BLOCK,BLOCK,BLOCK)

      ! ...

        WHERE(RED(2:NX-1,2:NY-1,2:NZ-1))
          U(2:NX-1,2:NY-1,2:NZ-1) =                                  &
     &        FACTOR*(HSQ*F(2:NX-1,2:NY-1,2:NZ-1)+                   &
     &        U(1:NX-2,2:NY-1,2:NZ-1)+U(3:NX,2:NY-1,2:NZ-1)+         &
     &        U(2:NX-1,1:NY-2,2:NZ-1)+U(2:NX-1,3:NY,2:NZ-1)+         &
     &        U(2:NX-1,2:NY-1,1:NZ-2)+U(2:NX-1,2:NY-1,3:NZ))
        ELSEWHERE
          U(2:NX-1,2:NY-1,2:NZ-1) =                                  &
     &        FACTOR*(HSQ*F(2:NX-1,2:NY-1,2:NZ-1)+                   &
     &        U(1:NX-2,2:NY-1,2:NZ-1)+U(3:NX,2:NY-1,2:NZ-1)+         &
     &        U(2:NX-1,1:NY-2,2:NZ-1)+U(2:NX-1,3:NY,2:NZ-1)+         &
     &        U(2:NX-1,2:NY-1,1:NZ-2)+U(2:NX-1,2:NY-1,3:NZ))
        END WHERE
```

pde1 code: FORALL version

```
   FORALL (X = 2:NX-1, Y = 2:NY-1, Z = 2:NZ-1, RED(X, Y, Z))       &
&    U(X,Y,Z) = FACTOR * (HSQ * F(X, Y, Z) +                        &
&        U(X-1, Y, Z) + U(X+1, Y, Z) +                              &
&        U(X, Y-1, Z) + U(X, Y+1, Z) +                              &
&        U(X, Y, Z-1) + U(X, Y, Z+1))

   FORALL (X = 2:NX-1, Y = 2:NY-1, Z = 2:NZ-1, .NOT. RED(X, Y, Z)) &
&    U(X,Y,Z) = FACTOR * (HSQ * F(X, Y, Z) +                        &
&        U(X-1, Y, Z) + U(X+1, Y, Z) +                              &
&        U(X, Y-1, Z) + U(X, Y+1, Z) +                              &
&        U(X, Y, Z-1) + U(X, Y, Z+1))
```

fft1

1d FFT benchmark has

- relatively high level of parallelism
- low level of vectorisation
- and a hypercube type communication.

GENESIS code follows the classical vectorised implementation (scheme A). Kernel is the subroutine which implements the FFT butterfly.

Can consider the butterfly calculation as a two-dimensional matrix of $\log N$ layers with $5N$ floating point operations in each. Layers represent the outer loop. Each layer contains a variable number of inner loops—number doubles from one layer to the next, beginning with one inner loop in the first layer.

Code is a good test of ability of HPF compilers to generate communication structures with a hypercube connectivity.

fft1 code

```
      DOUBLE PRECISION FR(0:NTOTAL-1), FI(0:NTOTAL-1)
      DOUBLE PRECISION WR(0:NTOTAL/2-1), WI(0:NTOTAL/2-1)
      DOUBLE PRECISION G(0:NTOTAL-1,0:1)

!HPF$ PROCESSORS LINE(NPROC)

!HPF$ DISTRIBUTE FR *(BLOCK) ONTO LINE
!HPF$ ALIGN FI(I) WITH *FR(I)

!HPF$ DISTRIBUTE G(BLOCK, *) ONTO LINE

      STEP = NTOTAL

      DO L = 1, LOGN
         IMAX = STEP/2

         DO M = 0, NTOTAL/STEP-1
            K = M*STEP
            J = K+IMAX
            JMAX = J+IMAX-1

            G(J:JMAX,0) = WR(M)*FR(J:JMAX) - WI(M)*FI(J:JMAX)
            G(J:JMAX,1) = WR(M)*FI(J:JMAX) + WI(M)*FR(J:JMAX)
            FR(J:JMAX) = FR(K:J-1) - G(J:JMAX,0)
            FI(J:JMAX) = FI(K:J-1) - G(J:JMAX,1)
            FR(K:J-1) = FR(K:J-1) + G(J:JMAX,0)
            FI(K:J-1) = FI(K:J-1) + G(J:JMAX,1)
         ENDDO

         STEP = IMAX
```

```
ENDDO
```

pde2

Solves a Poisson equation again, but this time by Multigrid scheme.

- Differential equation is discretised on a grid.
- Sequence of auxilliary grids introduced of twice, four times, etc spacing or original grid, up to a top level "grid" with a single non-boundary point.
- At each grid level use red-black relaxation.
- As the solution is updated, error terms are propagated up to coarser grids, corrections propagated down again.

Coarser grids give a more global view of solution—corrections propagated rapidly to all points. Finer grids take care of details on shorter length scales.

pde2 code sample

```
      SUBROUTINE INT2A (NPC, NPF, UC, UF)
      IMPLICIT NONE
      INTEGER NPC, NPF
      DOUBLE PRECISION UC (NPC, NPC), UF (NPF, NPF)
CHPF$ DISTRIBUTE UC * (BLOCK, BLOCK)
CHPF$ DISTRIBUTE UF * (BLOCK, BLOCK)
C
C     COARSE-TO-FINE TRANSFER (BILINEAR INTERPOLATION) AND CORRECTION.
C     (THIS MODIFIES ODD SITES ONLY OF UF)
C
        INTEGER NC

        NC = NPC - 1
        UF (2 : 2 * NC : 2, 3 : 2 * NC - 1 : 2) =
     1      UF (2 : 2 * NC : 2, 3 : 2 * NC - 1 : 2) +
     2          0.5D0 * (UC (1 : NC, 2 : NC) + UC (2 : NC + 1, 2 : NC))
        UF (3 : 2 * NC - 1 : 2, 2 : 2 * NC : 2) =
     1      UF (3 : 2 * NC - 1 : 2, 2 : 2 * NC : 2) +
     2          0.5D0 * (UC (2 : NC, 1 : NC) + UC (2 : NC, 2 : NC + 1))
      END
```

pde2 code sample

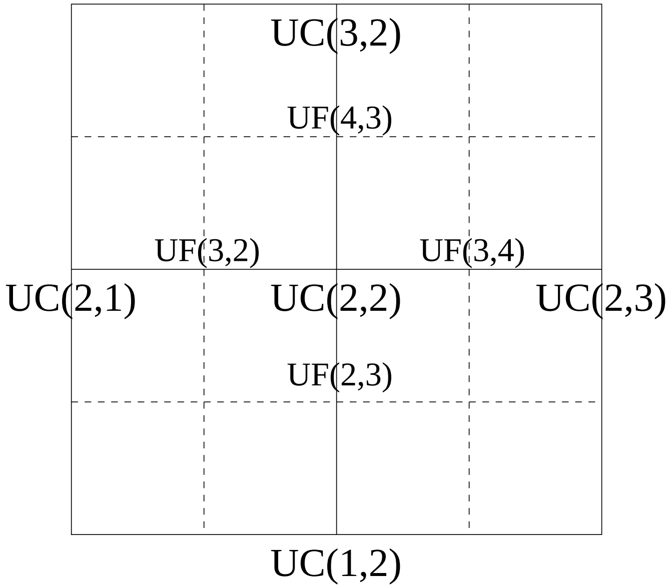

Figure 1: The coarse and fine grids in subroutine INT2A.

pde2 comments

The main problem was the difficulty in describing a stack of grids of different sizes in Subset HPF.

Could probably be achieved in Full HPF by creating a vector of derived data types, each containing a pointer to an allocatable array.

Subset HPF doesn't have the dynamic facilities for this—had to statically declare a fixed set of arrays of size 3, 5, 9, 17, . . . up to some fixed maximum problem size.

md1

Molecular dynamics program simulating a Lennard-Jones fluid.

- Molecules interact through short-range forces. Exploit locality by using particle-in-cell method.
- Space is divided into a set of small cells, size is approx equal to the range of the force.
- Only molecules in neighbouring cells can interact with one another.

The equations of motion for the molecules are integrated by a straightforward Verlet algorithm. For updating positions and velocities, easiest to use one-dimensional arrays (vectors) subscripted by molecule number.

The three-dimensional cell array is only used during the calculation of the forces on the molecules.

md1

Update has the following phases

1. Start from a vector of molecule positions, indexed by the molecule's identifier.
2. Assign the molecules to cells on the basis of their position. Gives array of cells with some number of molecules in each cell.
3. From cellular array of positions, compute the cellular array of forces on on the molecules.
4. Scatter array of forces is to a vector of forces indexed by molecular identifiers.
5. Use vector of forces to update the vector of velocities and finally the vector of positions.

md1 code

Most of the computation is in phase 3. This phase was straightforward but tedious to transcribe to array syntax. Would have been much neater, probably more efficient, using pure procedures in FORALL.

The communication in phases 2 and 4 is less trivial.

Phase 4 can be expressed using vector subscripts:

```
NATMCV = RESHAPE (NATMCL, (/NCELLS/))
DO I = 1, MAXCL
  IDV = RESHAPE (IDL (I, :, :, :), SHP1D)
  FXV = RESHAPE (FXL (I, :, :, :), SHP1D)
  FYV = RESHAPE (FYL (I, :, :, :), SHP1D)
  FZV = RESHAPE (FZL (I, :, :, :), SHP1D)
  WHERE (I .LE. NATMCV)
    FX (IDV) = FXV
    FY (IDV) = FYV
    FZ (IDV) = FZV
  ENDWHERE
END DO
```

md1 comments

Again much neater using (nested) FORALL:

```
FORALL (X = 1 : MX, Y = 1 : MY, Z = 1 : MZ)
  FORALL (I = 1 : NATMC (X, Y, Z))
    FX(IDL (I, X, Y, Z)) =  FXL(I, X, Y, Z)
    FY(IDL (I, X, Y, Z)) =  FYL(I, X, Y, Z)
    FZ(IDL (I, X, Y, Z)) =  FZL(I, X, Y, Z)
  END FORALL
END FORALL
```

Phase 2 more problematic. Doesn't seem to be an efficient way to implement it in parallel using HPF. In practise, resorted to sequential code.

Really needed a new combining scatter intrinsic—a "binning scatter"—which returns an array of lists directly.