Porting LCPFCT to the CRAY T3E Using HPF

Jay Boisseau, Ken Steube, and Bob Sinkovits
San Diego Supercomputer Center
San Diego, CA USA

OUTLINE

I. The Goal: Port a CRAY C90 Code to the CRAY T3E Using HPF

II. The Candidate: LCPFCT

III. Introduction to HPF

IV. SDSC's CRAY T3E and PGI's HPF Compiler

V. Initial Quick & Dirty Porting Plans

VI. Porting Experiences

VII. Timings

VIII. Remarks & Conclusions

IX. Future Efforts

I. THE GOAL: PORT A CRAY C90 CODE TO THE CRAY T3E USING HPF

1996 User Survey:

81% used C90 in 1996

12% of respondents used T3D in 1996 (9% used Paragon)

25% expected to use T3E in 1997

Allocations for 2Q97:

14 SDSC & 15 MetaCenter allocation requests (135 users in April)

Usage expected to grow as system becomes familiar, robust

Future Availability of Cycles

MPP cycles will be more plentiful in future than vector cycles

II. THE CANDIDATE: LCPFCT

Algorithm

Solves generalized continuity equations

Flux-Corrected transport algorithm

4th order phase accuracy, minimal residual diffusion

Monotone, conservative, positivity-preserving

Code

Finite difference, non-uniform grid, inflow/outflow, periodic boundary conditions, multiple coordinate systems

1D, extensible to multiple dimensions

Efficient on Cray vector platforms

Library of routines passes variables in COMMON blocks

Viability for Porting

Mostly nearest-neighbor communications

Mostly stride 1 loops

Representative of serial codes and vector codes

Easy to use in multidimensional codes

Less intensive than other more easily parallelized processes in typical reactive flow codes (chemical reactions, diffusion, etc.)

Satisfies main criteria for parallel candidate (Pancake 1996):

frequency of use, execution time, resolution needs

III. SDSC'S CRAY T3E AND PGI'S HPF COMPILER

SDSC CRAY T3E Configuration

256 PEs: 153.6 Gflop/s theoretical peak

300 MHz EV5 chips

240 Application PEs; 8–192 PE, express and dedicated queues

128 MB / PE (116 MB available to user codes)

PGI's HPF Compiler

Translates to F77 + PGI library calls

PGI library calls converted to SHMEM calls

Will use global memory addressing hardware directly

Will include CRAFT functionality

IV. INTRODUCTION TO HPF

Data Parallel Programming Language

Single-threaded control structure

Global name space

Loosely synchronous parallel execution

Single Program Multiple Data (SPMD) Model in HPF

Works primarily with array operations

No explicit messages, locks, synchronization

User controls data distribution

Extrinsic procedures allow MIMD operation

What is in HPF?

- All of Fortran 90
- Adds several compiler directives: DISTRIBUTE, ALIGN, INDEPENDENT, etc.
- Adds one executable statement: FORALL (added to Fortran 95)
- Adds one attribute: PURE (added to Fortran 95)

Parallelism in HPF:

- Array syntax
- Loops (FORALLs, INDEPENDENT DOs)
- Intrinsic functions

Key to Parallelism in HPF:

- Data distribution

Why Use HPF?

Easier for programmer than MPI (harder for compiler writer!)

Portable, and useful on a variety of parallel machines

Fortran is still language of choice for scientific programming—large installed user base

Fortran 90 has most useful features of C++/Ada plus unique features

Why Should I *Not* Use HPF?

Not as efficient as MPI (ever?)

Not as portable as MPI (yet)

Not widely available for networked computers (yet)

Not as cheap as MPI (yet)

V. INITIAL QUICK & DIRTY PORTING IDEAS

Is the algorithm amenable to data parallelism? Is the domain regular? If so:

- **Decompose the data using DISTRIBUTE and ALIGN directives. Use PROCESSORS and TEMPLATE if they seem warranted.**
- **Check DO loops for dependencies, and add INDEPENDENT directives where possible (may require splitting loops)**
- **Use FORALL loops and WHERE masks to operate on entire arrays simultaneously.**
- **Simplify complicated statements in more simple ones. Use array syntax, call HPF intrinsic functions, etc.**

Is the code performing well? Is it scalable? If not, choose a different algorithm or go to message passing.

VI. PORTING EXPERIENCES

(1) Converting DO loops to INDEPENDENT DO loops is neither straightforward nor always efficient

DO loops must be searched for *parallel* dependencies:

```
 DO I = 1, N
    FLXH(I+1) = FSGN(I+1) * AMAX1(0.0,
&                          AMIN1 (TERM(I+1),
&                          FABS(I+1), TERP(I+1)))
    RHON(I) = RLN(I) * ( LNRHOT(I) +
&                          (FLXH(I) - FLXH(I+1)) )
    SOURCE(I) = 0.0
 END DO
```

and split to avoid them

INDEPENDENT DO loops have excessive communications if RHS arrays have shifted indices:

```
!HPF$ INDEPENDENT
DO I = 2, N
   FLXH(I) = MULH(I) * (RHOT(I)-RHOT(I-1))
   DIFF(I) = RHOTD(I) - RHOTD(I-1)
END DO
```

Communication is best done *before* DO loops for efficiency:

```
tmp1(2:N) = RHOT(1:N-1)   ! ALIGN tmp1
tmp2(2:N) = RHOTD(1:N-1)  ! and tmp2
!HPF$ INDEPENDENT
DO I = 2, N
   FLXH(I) = MULH(I) * (RHOT(I)-tmp1(I))
   DIFF(I) = RHOTD(I) - tmp2(I)
END DO
```

FORALLs and array syntax are more efficient (if there is communication in the loop) because they are evaluated line by line

```
FORALL (i=1:n)
    y(i) = a*x(i) + b
    z(i) = c*y(i) + d
END FORALL
```

is executed as

```
FORALL (i=1:n) y(i) = a*x(i) + b
FORALL (i=1:n) z(i) = c*y(i) + d
```

which is similar to

```
y = a*x + b
z = c*y + d
```

Note: DO loops are perhaps most flexible

(2) Data management is difficult, making single-PE optimization *very* difficult

PGHPF violates Fortran standard of arrays in COMMON blocks being mapped to contiguous memory by default

PGHPF tends to destroy the old COMMON blocks and creates new COMMON blocks to hold the data

Even on a single PE, PGHPF gives a different mapping of data to cache lines than the same code compiled using f90

Declaring common blocks SEQUENTIAL fixes the problem but causes other problems

VII. TIMINGS

PEs	N=10000	N=100000
1	1.1790	10.6875
2	0.7502	5.5221
4	0.3768	2.8627
8	0.2515	1.5354
16	0.1973	0.8694
32	0.2046	0.5663

Compiled with: -O3 -O unroll2 -O bl -O aggress -O msgs

Executed for 20 LCPFCT calls

VIII. REMARKS & CONCLUSIONS

Keys to Parallelism in HPF:

Data distribution/layout across processors

Number operations in loops/arrays operations vs. overhead:

- **communication whenever RHS contains offset indices**
- **synchronization whenever next loop uses just-computed values**

Modifications to Quick and Dirty Porting Ideas

Split loops to remove dependencies, but merge to reduce overhead

Reorder statements to reduce overhead

Count operations—don't expect scalability on small problems

Use array syntax, FORALLs, intrinsic functions as much as possible

Rough Criteria for Scalable Parallel Loops

Latency is ~ 1 microsecond

Operations are ~ 1 nanosecond

Therefore:

need 1000 operations per loop per PE just to balance latency

need 100,000 operations per loop per PE to be scalable, or

 # loop iterations * # operations in loop / # PEs > 100,000

IX. FUTURE EFFORTS

LCPFCT-HPF

Store values in N-dimensional arrays for N-dimensional problem; push N-1 outer loops into subroutines

LCPFCT2D-HPF

Rick Devore (NRL) solved analytic problem in 2D and developed CMF, CRAFT versions using F90 array syntax; convert to HPF and optimize for T3E, extend to 3D

LCPFCT3D-HPF

Rick Devore (NRL) solved analytic problem in 3D and developed SHMEM, MPI code using F77 syntax; possibly convert to HPF?