Advanced Optimization Techniques for High Performance Fortran

Vikram Adve    Rob Fowler    Guohua Jin
Ken Kennedy    John Mellor-Crummey    Qing Yi

Center for Research on Parallel Computation and Department of Computer Science
Rice University
• Motivation and goals
• Background
  - Key features of Rice dHPF compiler
  - NAS Benchmarks
• Compilation techniques
  - Computation partitioning (CP) support
    » CP propagation for privatizable arrays
    » CP selection
    » interprocedural CP propagation
  - Coarse-grain pipelining
• Status and summary
Motivation and Goals

Problem: Good HPF performance requires extensive rewriting
- Existing Fortran codes have challenging (but essential) features
- Example: Excellent scalability for NAS HPF benchmarks
- **BUT** codes completely rewritten; grew to 2x in size

Goal: shift the burden of parallelization to the compiler
- Require only minimal source code modifications
- Enable effective parallelization of common code features
Parallelization Challenges

- Privatizable array temporaries
  - must avoid frequent communication, useless replication of work
- Intra-loop data reuse (within and across processors)
  - must distribute loops selectively to eliminate communication
  - must preserve intra-processor data reuse
  - must use aggressive coalescing within (and across) loops
- Non-PURE procedure calls within parallel loops
  - must partition computation across procedure calls
- Multi-dimensional data distributions with wavefront parallelism
  - must generate efficient coarse-grain pipelining
Background: Key Features of dHPF

- General computation partitioning model
  - Flexible owner-computes rule
- Integer set framework for communication analysis and code generation
  - Based on Omega Calculator
- Aggressive optimizations for message-passing systems
  - Communication aggregation and placement
- Integrated support for source-level performance tools
  - Collaboration with Pablo group
- Compiler and language support for out-of-core problems
- More details:
Core Optimizations in dHPF

- Aggressive CP selection to minimize communication costs
- Message vectorization for arbitrary communication patterns
- Message coalescing for arbitrary affine references
- Dataflow-based communication placement:
  - overlap communication with computation
  - tile communication under memory constraints
- Local/non-local index set splitting:
  - minimize runtime checks for non-local data access
  - overlap communication with local computation
- Recognize contiguous data sections to minimize buffer costs
- Sophisticated reduction recognition
- Logical constraint propagation to simplify control flow
NAS Benchmarks 2.3

• Versions
  - NAS Parallel Benchmarks: hand-coded MPI implementations
  - NAS Serial Benchmarks: (near parallel) serial form
    » adapted from SPMD parallel code with as few changes as possible

• Ideal for study of automatic parallelization with HPF
  - Regular, dense matrix codes
  - Near-parallel serial code is the best that one should realistically expect users to provide
  - Hand-coded parallelization available for comparison
Compilation Techniques

• Privatizable array temporaries
  - computation partition propagation from array uses to definitions
    » ensures that values are computed only where they are needed
    » reduces communication frequency through vectorization

• Communication minimization
  - computation partition selection
  - loop distribution for loop independent dependences
  - use aggressive coalescing within (and across) loops

• Multiple procedures
  - interprocedural propagation of computation partitions

• Multi-dimensional data distributions
  - coarse-grain pipelining for efficient wavefront parallelism
**Privatizable Array Temporaries**

**Definition:** an array is privatizable within a loop if
- all values are defined within the loop before their uses
- no definitions within the loop reach uses outside the loop

```fortran
do 10 k = 1, grid_points(3)-2
C$HPF INDEPENDENT NEW(cv, rhoq)
do 10 i = 1, grid_points(1)-2
do 20 j = 1-1, grid_points(2)-1
  rul = c3c4*rho_i(i,j,k)
cv(j) = vs(i,j,k)
rhoq(j) = dmax1(dy3+con43*rul, dy5 + c1c5*rul, dymax+rul, dy1)
do 30 j = 1, grid_points(2)-2
  lhs(i,j,k,1) = 0.0d0
  lhs(i,j,k,2) = -dtty2 * cv(j-1) - dtty1 * rhoq(j-1)
  lhs(i,j,k,3) = 1.0 + c2dtty1 * rhoq(j)
  lhs(i,j,k,4) = dtty2 * cv(j+1) - dtty1 * rhoq(j+1)
30       lhs(i,j,k,5) = 0.0d0
10 continue
```
Parallelization Options for Privatizables

• Unsatisfactory Options
  - forward substitution of definitions: multiplies computation
  - replicate privatizable array temporaries
    » all loops assigning to privatizable arrays are unpartitioned and replicated across all processors
  - distribute privatizable array temporaries
    » unless definitions and uses are perfectly aligned, communication needed between definition and use

• dHPF Approach
  - CP propagation from privatizable uses to definitions
    » values are computed only where they are needed
    » reduce communication frequency through vectorization
do 10 k = 1, grid_points(3)-2
C$HPF INDEPENDENT NEW(cv, rhoq)
do 10 i = 1, grid_points(1)-2
do 20  j = 1-1, grid_points(2)-1
    ru1 = c3c4*rho_i(i, j, k) ON_HOME lhs(i,j+1,k,2), lhs(i,j,k,3), lhs(i,j-1,k,4)
cv(j) = vs(i, j, k) ON_HOME lhs(i,j+1,k,2), lhs(i,j-1,k,4)
rhoq(j) = ON_HOME lhs(i,j+1,k,2), lhs(i,j,k,3), lhs(i,j-1,k,4)
      *               dmax1(dy3+con43*ru1, dy5 + c1c5*ru1, dymax+ru1, dy1)
do  30 j = 1, grid_points(2)-2
lhs(i,j,k,1) =  0.0d0
lhs(i,j,k,2) = -dtty2 * cv(j-1) - dtty1 * rhoq(j-1)
lhs(i,j,k,3) =  1.0 + c2dtty1 * rhoq(j)
lhs(i,j,k,4) =  dtty2 * cv(j+1) - dtty1 * rhoq(j+1)
30       lhs(i,j,k,5) =  0.0d0
10 continue

Loop from NAS 2.3-serial SP subroutine lhsy
do 40 k = 1, grid_points(3)-2
  do 40  j = 0, grid_points(2)-3
    do 40  i = 1, grid_points(1)-2
      fac1                  = 1.d0/lhs(i,j,k,n+3)
      lhs(i,j,k,n+4)    = fac1*lhs(i,j,k,n+4)
      lhs(i,j,k,n+5)    = fac1*lhs(i,j,k,n+5)
      do 10  m = 1, 3
        rhs(i,j,k,m)    = fac1*rhs(i,j,k,m)
        lhs(i,j+1,k,n+3) = lhs(i,j+1,k,n+3) - lhs(i,j+1,k,n+2) * lhs(i,j,k,n+4)
        lhs(i,j+1,k,n+4) = lhs(i,j+1,k,n+4) - lhs(i,j+1,k,n+2) * lhs(i,j,k,n+5)
        do 20  m = 1, 3
          rhs(i,j+1,k,m) = rhs(i,j+1,k,m) - lhs(i,j+1,k,n+2) * rhs(i,j,k,m)
          lhs(i,j+2,k,n+2) = lhs(i,j+2,k,n+2) - lhs(i,j+2,k,n+1) * lhs(i,j,k,n+4)
          lhs(i,j+2,k,n+3) = lhs(i,j+2,k,n+3) - lhs(i,j+2,k,n+1) * lhs(i,j,k,n+5)
          do 30  m = 1, 3
            rhs(i,j+2,k,m) = rhs(i,j+2,k,m) - lhs(i,j+2,k,n+1) * rhs(i,j,k,m)
            continue
    continue
  continue
40 continue

Loop from NAS 2.3-serial SP subroutine y_solve
(J and K array dimensions partitioned)
do 40 k = 1, grid_points(3)-2
  do 40 j = 0, grid_points(2)-3
    do 40 i = 1, grid_points(1)-2
      fac1 = 1.d0/lhs(i,j,k,n+3)
      ON HOME lhs(i,j,k,n+4), lhs(i,j,k,n+5), rhs(i,j,k,*)
      lhs(i,j,k,n+4) = fac1*lhs(i,j,k,n+4)
      ON HOME lhs(i,j,k,n+4)
      lhs(i,j,k,n+5) = fac1*lhs(i,j,k,n+5)
      ON HOME lhs(i,j,k,n+5)
      do 10 m = 1, 3
        rhs(i,j,k,m) = fac1*rhs(i,j,k,m)
        ON HOME rhs(i,j,k,m)
      enddo
      lhs(i,j+1,k,n+3) = lhs(i,j+1,k,n+3) - lhs(i,j+1,k,n+2) * lhs(i,j,k,n+4)
      ON HOME lhs(i,j,k,n+4)
      lhs(i,j+1,k,n+4) = lhs(i,j+1,k,n+4) - lhs(i,j+1,k,n+2) * lhs(i,j,k,n+5)
      ON HOME lhs(i,j,k,n+5)
      do 20 m = 1, 3
        rhs(i,j+1,k,m) = rhs(i,j+1,k,m) - lhs(i,j+1,k,n+2) * rhs(i,j,k,m)
        ON HOME rhs(i,j,k,m)
      enddo
      lhs(i,j+2,k,n+2) = lhs(i,j+2,k,n+2) - lhs(i,j+2,k,n+1) * rhs(i,j,k,m)
      ON HOME rhs(i,j,k,m)
      lhs(i,j+2,k,n+3) = lhs(i,j+2,k,n+3) - lhs(i,j+2,k,n+1) * rhs(i,j,k,m)
      ON HOME rhs(i,j,k,m)
      do 30 m = 1, 3
        rhs(i,j+2,k,m) = rhs(i,j+2,k,m) - lhs(i,j+2,k,n+1) * rhs(i,j,k,m)
        ON HOME rhs(i,j,k,m)
      enddo
    enddo
  enddo
enddo

Loop from NAS 2.3-serial SP subroutine y_solve
(J and K array dimensions partitioned)
do k=1,grid_points(3)-2
  do j=1,jsize-1
    CHPF$ INDEPENDENT, NEW(temp_rhs, temp_lhs)
    do i=1,grid_points(1)-2
      temp_rhs(:) = rhs(:,i,j-1,k)
      call matvec_sub(i,j,k,lhs(1,1,aa,i,j,k), temp_rhs, 
                     rhs(1,i,j,k))
      temp_lhs(:, :) = lhs(:, :, cc,i,j-1,k)
      call matmul_sub(i,j,k,lhs(1,1,aa,i,j,k), temp_lhs, 
                      lhs(1,1,bb,i,j,k))
      call binvcrhs(i,j,k,lhs(1,1,bb,i,j,k), 
                    lhs(1,1,cc,i,j,k), rhs(1,i,j,k))
    enddo
  enddo
enddo

Loop from NAS 2.3-serial BT subroutine y_cell
(J and K array dimensions partitioned)
**Problem:** communication due to loop-carried data dependences
- if carrying loop is innermost, parallelism is maximized, but communicate lots of small messages
- if carrying loop is outermost, computation is serialized, but minimum number of large messages

**Approach:** strip mine to balance communication and parallelism
- dHPF strip mines partitioned loops for multi-dim partitionings
Speedups for BT class A on SP-2 *

- Perfect
- MPI
- dHPF

97% of hand coded MPI
74% of hand coded MPI

*All speedups relative to 4 processor MPI
Preliminary Performance Results

Trace of hand-generated generated code for NPB-2.3 BT

Trace of dHPF parallelized code for NPB-2.3-serial BT
• Parallelizing realistic codes with HPF requires either:
  - substantial effort from programmer to rewrite code to remove privatizable array use (see PGI NAS HPF benchmarks)
  - sophisticated compiler support to transform the program to minimize communication and partition computation effectively

• dHPF project goals:
  - enable effective semi-automatic parallelization of sophisticated scientific codes
    » for MP or DSM systems
  - require minimal programmer effort