An Emerging, Portable Co-array Fortran Compiler for High-Performance Computing

John Mellor-Crummey, Yuri Dotsenko, Cristian Coarfa, Daniel Chavarria-Miranda
{johnmc, dotsenko, ccristi, danich}@cs.rice.edu

Co-Array Fortran Language

- SPMD process images
  - number of images fixed during execution
  - images operate asynchronously
- Both private and shared data
  - real a(20,20) private: a 20x20 array in each image
  - real a(20,20) (T) shared: a 20x20 array in each image
- Simple one-sided shared memory communication
  - x(:,p+2) = a(:,p+2) copy rows from p:p+2 into local columns
- Flexible synchronization
  - sync_team(tag [, wait])
  - wait = a vector of process ids to synchronize with
- Pointers and dynamic allocation
- Parallel I/O

Explicit Data and Computation Partitioning

integer A(10,10)[*]

Finite Element Example

subroutine assemble(start, prin, ghost, neib, x)
  integer :: start(1), prin(1), ghost(1), neib(1)
  integer :: k1, k2, p
  real :: x(*)
  call sync_all(neib)
  do p = 1, size(neib) ! Update from ghost regions
    k1 = start(p); k2 = start(p)+1
    x(prin(k1:k2)) = x(ghost(k1:k2)) (neib(p))
  enddo
  call sync_all(neib)
  do p = 1, size(neib) ! Update the ghost regions
    k1 = start(p); k2 = start(p)+1
    x(ghost(k1:k2)) (neib(p)) = x(prin(k1:k2))
  enddo
end subroutine assemble

PUT Translation Example

... real(8) a(0:N+1,0:N+1)[*]
  me = this_image()
... ! ghost cell update
  a(1:N,1:N+1)(left(me)) = a(1:N,0)
...

type CafHandleReal8
  integer:: handle
  real(8):: ptr(;;)
end type
type(CafHandleReal8) a_caf
...
allocate( cafBuffer_1%ptr(1:N,0:0) )
call cafBuffer_2%ptr => a_caf%ptr(1:N,1:N+1)
call cafBuffer_1%ptr = a_caf%ptr(1:N,0)
call CafArmciPutS(a_caf%handle,left(me),
call cafBuffer_1, cafBuffer_2)
deallocate( cafBuffer_1%ptr )
...

Research Focus

- Enhancements to Co-Array Fortran model
  - Point-to-point one-way synchronization
  - Hints for matching synchronization events
  - Collective operation intrinsics
  - Split-phase primitives
  - Synchronization strength-reduction
  - Communication vectorization
  - Platform-driven communication optimization
  - Transform as useful from 1-sided to two-sided and collective communication
  - Generate both fine-grain load/store and calls to communication libraries as necessary
  - Multi-model code for hierarchical architectures
  - Convert Gets into Puts
  - Compiler-directed parallel I/O with UIUC
  - Interoperability with other programming models

Implementation Status

- Source-to-source code generation for wide portability
- Open source compiler will be available
- Working prototype for a subset of the language
- Current compiler implementation performs no optimization
- each co-array access is transformed into a get/put operation at the same point in the code
- Code generation for the widely-portable ARCMI library for one-sided communication
- Front-end based on production-quality Open64 front end, modified to support source-to-source compilation

Early Performance Results

NAS MG class C

IA64 / Myrinet 2000