CMSC 714
Lecture 6
High Performance Fortran (HPF)

Alan Sussman

HPF Model of Computation
- goal is to generate loosely synchronous program
  - main target was distributed memory machines
- Explicit identification of parallel work
  - forall statement
- Extensions to FORTRAN90
  - the forall statement has been added to the language
  - the rest of the HPF features are comments/pragmas
- Key Feature: Data Distribution
  - how should data be allocated to nodes?
  - critical questions for distributed memory machines
  - turns out to be useful for SMP too since it defines locality

HPF Language Concepts
- Virtual processor
  - an abstraction of a CPU
  - can have one and two dimensional arrays of VP's
  - each VP may map to a physical processor
    - several VP's may map to the same processor
- Template
  - a virtual array (no data)
  - used to describe how real array are aligned with each other
  - templates are distributed onto virtual processors
- Align directives
  - expresses how data different arrays should be aligned
  - uses affine functions of array indexes
    - align element I of array A with element I+3 of B

Distribution Options
- BLOCK
  - divide data into N (one per VP) contiguous units
- CYCLIC
  - assign data in round robin fashion to each processor
- BLOCK(n)
  - groups of n units of data are assigned to each processor
  - must be at least (array size)/n virtual processors
- CYCLIC(n)
  - n units of contiguous data are assigned round robin
  - CYCLIC is the same as CYCLIC(1)
- Each can be applied separately to each dimension of a multi-dimensional array

Computation
- Where should the computation be performed?
- Goals:
  - do the computation near the data
  - non-local data requires communication
  - keep it simple
  - HPF compilers are already complex
- Compromise: “owner computes”
  - computation is done on the node that contains the lhs of a statement
  - non-local data for the rhs operands are sent to the node as needed, often before a forall loop starts

Notes
- Programming assignment
  - First part (sequential and either MPI or OpenMP) due next Thursday, Oct. 6
  - You should have received email with which parallel version to do – if not, send email to Dr. Sussman
- Next class
  - Dr. Sussman will finish talking about Titanium, and take questions on Titanium and OpenMP vs. MPI
  - Nick Rutar will talk about the functional programming language Sisal
Finding the Data to Use

- **Easy Case**
  - the location of the data is known at compile time
- **Challenging case**
  - the location of the data is a known (invertible) function of input parameters such as array size
- **Difficult Case (irregular computation)**
  - data location is a function of data
  - indirection array used to access data \(A[index[i], j] \equiv \ldots\)

Challenging Case

- Each processor can identify its data to send/receive
  - use a pre-processing loop to identify the data to move
  
  ```
  for each local element i
  
  receive_list = global_to_proc(f(I))
  send_list = global_to_proc(f^{-1}(I))
  
  send data in send_list and receive data in receive_list
  
  for each local rhs element I
  
  perform the computation
  ```

Irregular Computation

- Pre-processing step requires data to be sent/received
  - since we might need to access non-local index arrays
- **two possible cases**
  - Gather: \(a(i) = b(u(i))\)
    - pre-processing builds a receive list for each processor
    - send list is known based on data layout
  - Scatter: \(a(u(i)) = b(i)\)
    - pre-processing builds a send list for each processor
    - receive list is known based on data layout

Communication Library

- **How is HPF different from PVM/MPI?**
  - abstraction based on distributed, but global arrays
    - provides some support for index translation
    - PVM/MPI only has local arrays
    - multicast is in one dimension of an array only
    - shifts and concatenation provided
    - special ops for moving vectors of send/recv lists in the library for the compiler to use
      - `precomp_read`
      - `postcomp_write`
- **Goals**
  - written in terms of native message passing
  - tries to provide a single portable abstraction to compile to

Performance Results

- **How good are the speedup results?**
  - only one application shown
    - speedup is similar to hand tuned message passing program
      - one extra log(n) communication operations decreases performance
    - how good is the hand tuned program?
      - speedup is only 6 on 16 processors
- **What is Figure 4 showing?**
  - compares performance on two different machines
  - no explanation
    - is this showing the brand x is better then brand y?
    - does it show that their compiler doesn’t work on brand y?
    - lesson: figures should always tell a story
      - don’t require the reader to guess the story

HPF on the Earth Simulator
IMPACT-3D

- **HPF Code**
  - Uses data distribution in one dimension
- **Vector Code**
  - Uses inner most array dimension
- **Achieves 14.9 Tiflops (45% of peak)**
- **Got 39% of peak using traditional HPF**
  - 45 lines of directives
  - 1,334 lines of executable code