COMPILER AND RUNTIME SUPPORT FOR PARALLEL I/O IN DISTRIBUTED MEMORY MACHINES

Alok N. Choudhary
Syracuse University
IBM, T. J. Watson Research Center
February 2, 1994
Outline

- High-Performance Fortran/Fortran 90D
- Compiler Design
- Integration of Data and Task Parallelism
- Models for Out-of-Core computations
- Compiler Support (HPF)
- Runtime Support
- Summary
Integration of the HPF and the FM compiler

- Language Integration
  - Each FM process/task coded in HPF and compiled using the Syracuse HPF compiler
  - FM tasks connected using FM channels
  - Each process is assigned to a FM submachine
  - Process initiated using the statement:
    HPFCALL p_name (c1, c2) submachine(a:b)
    * Sets up the input and output channels c1 and c2.
    * Allocates a set of processors (submachine) for the process.
Integration of the HPF and the FM compiler

- Compilation
  - Currently done in 2 phases
    * Phase 1: Compiling each FM process written in HPF using the HPF compiler.
    * Phase 2: Task parallel procedures are compiled separately using the FM compiler.

- Runtime System
  - FM uses a set of thread management and communication libraries based on TCP/IP.
  - HPF compiler uses a set of communication libraries based on express.
  - A FM compatibility library was written to interface these two runtime systems.
Code: Data Parallel Version

C HPF Code

program data_parallel
complex A(512,512), B(512,512), C(512,512)
!HPF$ processors p(32)
!HPF$ align B, C with A
!HPF$ distribute A(BLOCK,*)
do i = 1,nimages    ! Repeat for each image:
call read(A,B)
call fft(A)        ! FFT first input array
call fft(B)        ! FFT second input array
C = A*B
forall(i=1:512, j=1:512) C(i,j) = CONJG(C(i,j))
call fft(C)        ! Inverse FFT
call write(C)
enddo
C

FM Code

program mixed_parallel
processors pr(24)
outport (complex x(512,512/8)) outs1(8), outs2(8)
inport (complex x(512,512/8)) ins1(8), ins2(8)
channel(in=ins1(:), out=outs1( :))
channel(in=ins2(:), out=outs2( :))

processes
    hpfcall fft(nimages, outs1) submachine(1:8)
    hpfcall fft(nimages, outs2) submachine(9:16)
    hpfcall ifft(ins1, ins2) submachine(17:24)
endprocesses

• Each of the FM process is written in HPF.
• Processes communicate using the parallel sends and parallel receives.
• Each process is assigned to a set of eight processors.
A Case Study: 2D Convolution

Figure 1: Convolution Algorithm Structure

- Each process is Data-Parallel.
- Process Compiled using HPF Compiler.
- Processes communicate using the FM channels.
- The FM channels communicate data using parallel sends and parallel receives.

E.g. Vesta could be used

Alok N. Choudhary (Syracuse University)
Bhaven Avalani (Syracuse University)
Ian Foster (ANL)
Ming Xu (ANL)
Convolution Performance: IBM SP1 with TCP/IP
Convolution Performance: IBM SP1 with TCP/IP

![Graph showing convolution performance with varying numbers of processors and data points. The x-axis represents processors, the y-axis represents time in seconds, and different lines represent different data point sizes: DP 32, DP 64, DP 256, DP 512, and DP 1024.](image)
Types of I/O Requirements

- Parallel READ/WRITE (SC'93, IPPS93)
- Out-of-Core (OOC) computations (report)
- Checkpointing/Restart
- Debugging
- Real-Time Output
A Model for OOC Compilation

- Discussion in the context of HPF
- HPF provides constructs to distribute data
- This should be handled even when data does not fit in memory

!HPF$ DISTRIBUTE (BLOCK,BLOCK) :: A,B
OOC Compilation Issues

- Consider the following HPF program fragment

\[
\text{REAL } A(1024,1024), B(1024,1024) \\
\ldots \\
!\text{HPF}$\text{ PROCESSORS } P(4,4) \\
!\text{HPF}$\text{ TEMPLATE } T(1024,1024) \\
!\text{HPF}$\text{ DISTRIBUTE(BLOCK,BLOCK) } \\
!\text{HPF}$\text{ ALIGN } (I,J) \text{ with } T(I,J) :: A, B \\
\ldots \\
\text{FORALL } (I=1:N, J=1:N) \\
A(I,J) = (B(I,J-1) + B(I,J+1) + \\
B(I+1,J) + B(I-1,J))/4 \\
\ldots \\
B = A
\]

- OOC Compilation for the above assignment statement
Out-of-core Communication

1. Stripmine Code based on memory size.
2. Schedule communication for entire Out-of-Core data.
3. Repeat $k$ times ($k$ is the stripmine factor).
   3.1 Perform local I/O access.
   3.2 Perform Computation.
   3.3 Store Data.
4. If iteration < $k$ then go to 3.

In-core Communication

1. Stripmine Code based on memory size.
2. Repeat $k$ times ($k$ is the stripmine factor).
   2.1 Schedule communication for In-Core data (gather).
   2.2 Perform local I/O access.
   2.3 Perform Computation.
   2.4 Schedule communication for In-core data (scatter).
   2.5 Store local Data.
3. If iteration < $k$ then go to 2.
OOC Compilation: Runtime Support

Array distributed on 16 processors
(A)

Out-of-core Local Array on P5
(B)

In core Local Array on P5
(C)

Local Array File on P5
(D)

Example of OCLA, ICLA and LAF
Comm. Primitive Design Alternatives

- Explicit Communication: Each processor access only its local files and sends/receives data explicitly
- Direct Access Methods: Communication is performed by processors reading each others files
Runtime Support

- Array Management
  - read_vec(file, A, OCAD, i, j, start_vec, end_vec, stride)
  - write_vec(file, A, OCAD, i, j, start_vec, end_vec, stride)
  - write_vec_with_reuse(file, A, OCAD, i, j, start_vec, end_vec, stride, left_shift, right_shift)
  - ...

- Communication
  - out_of_core_shift(file, OCAD, nvec, direction)
  - out_of_core_multicast(file, OCAD, i, j, nelements, vec, source, proclist)
  - ...
Preliminary Results

Laplace Equation Solver (time in sec. for 10 iterations)

<table>
<thead>
<tr>
<th></th>
<th>64 Procs</th>
<th>64 Procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct File Access</td>
<td>79.12</td>
<td>280.8</td>
</tr>
<tr>
<td>Explicit Communication</td>
<td>75.12</td>
<td>274.7</td>
</tr>
<tr>
<td>Explicit Communication</td>
<td>71.71</td>
<td>269.1</td>
</tr>
<tr>
<td>with data reuse</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Slab Size (assumed available memory) $1/4^{th}$ of local array size
Summary and Discussion

Summary

- Basic Design of HPF/Fortran 90D Compiler
- Integration of Task and Data Parallelism
- Models for Out-of-Core Computations
- Preliminary compiler support for parallel I/O
- Initial (Partial) design of runtime support

Discussion

- Need support to control the number of disks
- Need support to control striping and data distribution parameters on disks
Global Model for Out-of-Core Computations

- Separate (at least logically) Compute and I/O nodes
- Global view of File maintained
- Access strategy may require shuffling data
Summary of Contributions

- Developed a portable compiler for F90D/HPF (Demo SC'92)
- Technology transfer to commercialize HPF compiler
- Language Extensions to F90D to support irregular problems
- Benchmark Suite for evaluating compiler and machine performance
- FortranD formed a basis for HPF.
- HPF is expected to serve as an informal standard for portable parallel programming