COMPILER AND RUNTIME SUPPORT FOR PARALLEL I/O
Outline

- Parallel I/O Requirements
- Models for Out-of-Core computations and Parallel I/O
- Compiler Support
- Runtime Support
- Performance Issues
- Use of Parallel I/O for Paradigm Integration
# I/O requirements for GC applications

## A. Environmental and Earth Sciences:

<table>
<thead>
<tr>
<th>Application</th>
<th>I/O requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environmental modeling</td>
<td>T: 10s of GB. S: 100s of MB - 1 GB per PE. A: Order of 1 TB.</td>
</tr>
<tr>
<td>Eulerian air-quality modeling</td>
<td>S: Current 1 GB/model, 100 GB/application; projected 1 TB/application. A: 10 TB at 100 model runs/application.</td>
</tr>
<tr>
<td>Earth system model</td>
<td>S: 108 MB/simulated day. 100 GB/decade-long simulation. B: 100 MB/s.</td>
</tr>
<tr>
<td>4-D data assimilation</td>
<td>S: 100 MB - 1 GB/run. A: 3 TB database. Expected to increase by orders of magnitude with the Earth Observing System (EOS) - 1 TB/day.</td>
</tr>
<tr>
<td>Ocean-Atmosphere Climate modeling</td>
<td>S: 100 GB/run (current). B: 100 MB/sec. A: 100s of TB.</td>
</tr>
</tbody>
</table>

A: Archival Storage  
T: Temporary Working Storage  
S: Secondary Storage  
B: I/O Bandwidth

(Ref: del Rosario and Choudhary: Computer, March 94)
Types of I/O Requirements

- Parallel READ/WRITE
- Out-of-Core (OOC) computations
- Checkpointing/Restart
- Debugging
- Real-Time Output
Compilation Issues

- Data Parallel Language like HPF.
- Out-of-core program due to large data set.
- Most MPPs do not provide virtual memory on nodes.
- Even if virtual memory is provided (e.g., Intel Paragon), performance is not good.
- True even in the case of sequential machines.
- Hence, translate to message passing node program with explicit calls to I/O routines.
Architectural Model
Global VS. Local View of Data Distributions

- Although Data is Distributed (Local View) in Memory Files could be shared (Global View)
- At what point is this view changed governs compiler and runtime support

(A) Local Model

(B) Global Model
Local 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
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
- Local Array of each processor stored in a separate file called Local Array File (LAF).

- If each processor has its own disk, the LAF is stored on that disk.

- If there is a common set of disks, the LAF is distributed across one or more of these disks.

- The portion of the local array in memory is called the In-core Local Array (ICLA).

- Parts of the LAF have to be fetched into the ICLA and then stored back on disk after the computation.
Out-of-Core Compilation

HPF Program

In-core Phase
1. Partition Computation.
2. Determine Communication.
3. Determine Local Space Bounds.

Out-of-core Phase
1. Strip-mine in Local Space.
2. Modify communication to incorporate I/O.
3. Modify loops to insert I/O calls.

Node + MP + I/O Code
OOC Compilation Issues

- Consider the following HPF program fragment

```
REAL A(1024,1024), B(1024,1024)

............
!HPF\$ PROCESSORS P(4,4)
!HPF\$ TEMPLATE T(1024,1024)
!HPF\$ DISTRIBUTE(BLOCK,BLOCK)
!HPF\$ ALIGN (I,J) with T(I,J) :: A, B
............
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
............
B = A
```

- OOC Compilation for the above assignment statement

![Diagram of OOC Compilation](image-url)
OOC Compilation: Communication Options

Out-of-core Communication

1. Strip-mine Code based on memory size.
2. Schedule communication for entire Out-of-Core data.
3. Repeat $k$ times ($k$ is the strip-mine 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

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)
  - ...

Data Reuse

- Move columns
- ICLA
- Overlap Areas
do k=1 to no_of_iterations
    call oc_shift(unit1,OCAD,1,right) ! right shift
    call oc_shift(unit1,OCAD,1,left)  ! left shift
    do l=1, no_of_slabs
        call read_vec(unit1, array1, OCAD, i, j, start_vec, end_vec, stride)
        do j=j1, j2
            do i=i1, i2
                array2(i,j)= ((array1(i,j-1)+array1(i,j+1)+array1(i+1,j)+
                                array1(i-1,j))/4.0)
            end do
        end do
    end do
    call write_vec(unit2, array2, OCAD, i, j, start_vec, end_vec, stride)
end do

C  exchange file unit numbers instead of explicitly
C  copying files (optimization)
    unit1 ← unit2
end do
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 with data reuse</td>
<td>71.71</td>
<td>269.1</td>
</tr>
</tbody>
</table>

- Slab Size (assumed available memory) $\frac{1}{4}$th of local array size