A Model and Compilation Strategy for Out-of-Core Data Parallel Programs

Rajesh Bordawekar, Alok Choudhary, Ken Kennedy, Charles Koelbel, Michael Paleczny

PPoPP
July 19, 1995

Presented by: Michael Paleczny
Overview

1. Out-of-core problems
2. Models
3. Compilation Strategy
4. Results
5. Conclusions
Motivation

I/O Types

- Initial and final I/O
  All applications

- Checkpoint I/O
  Particle algorithms in astrophysics

- Real-time I/O
  Computational fluid dynamics

- Out-of-core I/O
  Environmental modeling
### Optional

**I/O Requirements for GC Applications**

<table>
<thead>
<tr>
<th>Applications</th>
<th>A: 1 TBytes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational Fluid and Combustion Dynamics</td>
<td>B: 0.5 GBytes/s to disk, 45 MBytes/s to disk for visualization.</td>
</tr>
<tr>
<td>Solar Activity and Heliospheric Dynamics</td>
<td>B: 200 MB/sec</td>
</tr>
<tr>
<td></td>
<td>A: Up to 500 GB</td>
</tr>
<tr>
<td>Convective turbulence in Astrophysics</td>
<td>S: 5-10 GB/run</td>
</tr>
<tr>
<td></td>
<td>B: 10-100 MB/s</td>
</tr>
<tr>
<td>Particle algorithms in Cosmology and Astrophysics</td>
<td>S: 1-10 GB/file; 10-100 files/run.</td>
</tr>
<tr>
<td></td>
<td>B: 20-200 MB/s</td>
</tr>
<tr>
<td>Radio synthesis imaging</td>
<td>S: 1-10 GB</td>
</tr>
<tr>
<td></td>
<td>A: 1 TB</td>
</tr>
</tbody>
</table>

A: Archival Storage  
T: Temporary Working Storage  
S: Secondary Storage  
B: I/O Bandwidth  
(Ref: del Rosario and Choudhary)
Key Points

- Goals
  - Manage I/O for programmers
  - Optimize I/O

- Compilation strategy
  - Extend data-parallel "distribution" to I/O

- Data storage model
  - Expressed by choice of Placement Model
Models

*Programmer, Compiler, System, Storage*

- **Programmer:**
  HPF + I/O directives

- **Compiler:**
  Message-passing node program + I/O statements

- **System:**
  Distributed memory

- **Storage:**
  Data striping over multiple disks
Execution Models

Global Placement Model (GPM)

- Maintain global view of arrays for I/O
- Store global array in one common file
- Read and write in global name-space
  Compute in local name-space
Execution Models

Local Placement Model (LPM)

- Maintain distributed view of arrays
- Store each OCLA into a separate *logical* file
- Read, write, and compute in local name space
Optional

Architectural Model

- A set of processors connected to a set of disks via a high-speed interconnection network.

- Two basic Models
  1. Each processor connected to a local disk.
  2. Processors share the disks.
Optional

Programmer Model

- Data Parallel Programming Paradigm
- High Performance Fortran provides compiler directives
  - BLOCK, CYCLIC, BLOCK-CYCLIC
- User writes computations in global name space
- Data-parallel operations - FORALL
- Single Program Multiple Data (SPMD) Execution
- Local Array per processor
Example

Stencil Application in HPF, code

REAL A(1024,1024), B(1024,1024)
!
!HPF$ PROCESSORS P(4,4)
!
!HPF$ DISTRIBUTVE (BLOCK,BLOCK) ONTO P :: A, B
!
!HPF$ IO-DISTRIBUTIVE (::,100)

FORALL (I=2:N-1, J=2:N-1)

A(I,J) = (B(I,J-1) + B(I,J+1) + B(I+1,J) + B(I-1,J))/4

ENDFORALL
5 Point Stencil

Data-Parallel Distribution, diagram

Data Distribution in P1 to P16
5 Point Stencil

Data-Parallel and I/O Distribution, diagram

Data Distribution in P1 to P16  I/O Distribution in P5
do j = lower_bound, upper_bound
   do i = lower_bound, upper_bound
      A(i,j) = (B(i,j-1) + B(i,j+1) + B(i-1,j) + B(i+1,j))/4
   end do
end do
Optional

Data-Parallel and I/O Distribution

- Out-of-core array
  An array stored on disk during execution

- In-core array
  The "block" of an out-of-core array residing in memory

- Local array
  The section of an array "owned" by a processor

- In-core-local array
  The intersection of an in-core array and a local array
5 Point Stencil

Out-of-Core Communication, diagram

Data Distribution in P1 to P16

I/O Distribution in P5

Data to be Communicated
5 Point Stencil

*Out-of-Core Communication, code*

Call communication routine to perform *overlap shift*.

```
do i=1, k
   Call I/O routine to read the ICLA and overlap region.
   do j = lower_bound, upper_bound
      do i = lower_bound, upper_bound
         A(i,j) = ((B(i,j-1) + B(i,j+1) + B(i-1,j) + B(i+1,j))/4)
      end do
   end do
   Call I/O routine to store the result.
end do
```
5 Point Stencil

*In-Core Communication, diagram*

Data Distribution in P1 to P16

I/O Distribution in P5

Data to be Communicated
5 Point Stencil

*In-Core Communication, code*

do l=1, k

    Call I/O routine to read the ICLA.
    Call communication routine for overlap region.
    do j = lower_bound, upper_bound
        do i = lower_bound, upper_bound
            \[
            A(i,j) = \frac{(B(i,j-1) + B(i,j+1) + B(i-1,j) + B(i+1,j))}{4}
            \]
        end do
    end do

    Call I/O routine to store the results.
end do
Run-Time Support

- Mapping Routines
  - Map the Placement Model on the underlying machine.

- Access Routines
  - Read and write array sections

- Collective Communication Routines
  - Coordinate the activity of processor groups

(PASSION Runtime System, Choudhary et.al.)
# Results

*File Access Methods*

<table>
<thead>
<tr>
<th></th>
<th>Array Size: $2K \times 2K$</th>
<th>Array Size: $4K \times 4K$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32 Procs</td>
<td>64 Procs</td>
</tr>
<tr>
<td>Direct File Access</td>
<td>73.45</td>
<td>79.12</td>
</tr>
<tr>
<td>Explicit Communication</td>
<td>68.84</td>
<td>75.12</td>
</tr>
<tr>
<td>Explicit Communication with data reuse</td>
<td>62.11</td>
<td>71.71</td>
</tr>
</tbody>
</table>

Performance of Laplace Equation Solver
Time in sec. for 10 iterations
Results

File I/O or Virtual Memory

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Virtual Memory</th>
<th>Synchronous I/O</th>
<th>Percent Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>483</td>
<td>284</td>
<td>41.2</td>
</tr>
<tr>
<td>2</td>
<td>303</td>
<td>195</td>
<td>35.6</td>
</tr>
<tr>
<td>4</td>
<td>121</td>
<td>71.6</td>
<td>40.8</td>
</tr>
</tbody>
</table>

Performance of LU factorization using virtual memory
Array Size: 1600 × 1600
Execution times in seconds
Results

File I/O or Virtual Memory

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Virtual Memory</th>
<th>Synchronous I/O</th>
<th>Overlapped I/O</th>
<th>Percent Overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>634</td>
<td>316</td>
<td>277</td>
<td>12.3</td>
</tr>
<tr>
<td>2</td>
<td>368</td>
<td>227</td>
<td>181</td>
<td>20.3</td>
</tr>
<tr>
<td>4</td>
<td>375</td>
<td>186</td>
<td>136</td>
<td>27.0</td>
</tr>
<tr>
<td>8</td>
<td>334</td>
<td>132</td>
<td>119</td>
<td>10.1</td>
</tr>
<tr>
<td>16</td>
<td>62</td>
<td>159</td>
<td>153</td>
<td>3.9</td>
</tr>
</tbody>
</table>

With 16 processors, the problem is almost entirely in-core.

Performance of Red-Black relaxation
Array Size: 320 × 320 × 320
Execution time in sec. per iteration
Results

*Overlapping I/O and Computation*

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Synchronous I/O</th>
<th>Overlapped I/O</th>
<th>Percent Overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12339</td>
<td>10991</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>7920</td>
<td>7324</td>
<td>7.5</td>
</tr>
<tr>
<td>4</td>
<td>3464</td>
<td>3206</td>
<td>7.4</td>
</tr>
<tr>
<td>8</td>
<td>1804</td>
<td>1700</td>
<td>5.7</td>
</tr>
<tr>
<td>16</td>
<td>1110</td>
<td>999</td>
<td>10</td>
</tr>
</tbody>
</table>

Performance of LU factorization with pivoting
Array Size: $6400 \times 6400$
Execution time in sec.
Conclusions

- Data-storage layout affects I/O strategy
- Language directives provide implicit I/O
- Compilers can optimize I/O