CMSC 714
Lecture 18
Runtime Parallelization

Gary Jackson and Alan Sussman
Notes

• Midterm exam in 2 weeks, on Tuesday, Nov. 13
  • on readings through previous week
• Group Project interim report due Nov. 9
Outline

• Overview
• Compiler-driven: Multiblock Parti
• Library-driven: Global Arrays
• Conclusion
Overview

• Writing good parallel programs for distributed memory systems is hard.
• Idea: abstraction on top of message passing to get results
  • We can do this where communication is regular: block-structured applications
  • Trade off: (somewhat) reduced performance for reduced effort
Multiblock Parti

• Provide High Performance Fortran-like language enhancements to support block-structured applications

• Treat things statically, where we can
  • Like Fortran D, High Performance Fortran, etc.

• Use run-time support where we can't establish compile-time bounds
Runtime Support

- **Regular.Section_Move_Sched**
  - Schedule a regular section move
  - Accommodates block, cyclic, and block-cyclic distributions when the bounds & strides are known at run-time

- **Overlap_Cell_Fill_Sched**: schedule moves for overlap / ghost cells
Compiler Support

• Additional HPF-like directives
• Static analysis for data distribution
• Insert calls for runtime workload partitioning based on data distribution
Static Analysis

• Done on for_all loop parameters
• Categorize one of three ways
  • No communication necessary
  • Copy overlap (ghost) regions
  • Copy regular sections
Experiment: Overhead

- Extra time from library calls and schedule building isn't too bad
Experiment: Multiblock Code

- Within 20% of hand-parallelized F77
- Difference between compiler-parallelized & hand-parallelized F90 is mostly in computing loop bounds and searching for previously-used schedules
Experiment: Multigrid Code

• Within 10% of hand-parallelized code

<table>
<thead>
<tr>
<th>No. of Proc.</th>
<th>Compiler: First Iteration</th>
<th>Compiler: Per-subsequent Iteration</th>
<th>By Hand: First Iteration</th>
<th>By Hand: Per-subsequent Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4.80</td>
<td>2.29</td>
<td>4.60</td>
<td>2.14</td>
</tr>
<tr>
<td>16</td>
<td>3.84</td>
<td>1.38</td>
<td>3.41</td>
<td>1.35</td>
</tr>
<tr>
<td>32</td>
<td>3.03</td>
<td>.95</td>
<td>2.48</td>
<td>.88</td>
</tr>
</tbody>
</table>

Fig. 7. Semicoarsening multigrid performance (sec).
Experiment: Compiler Optimizations

• Performance stinks if schedules are not saved (Version I)
• Hand-implemented reuse improves over runtime reuse (II vs. III)
• Un-implemented optimization for loop-bounds in subroutines also improves (Version IV)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>13.45</td>
<td>7.63</td>
<td>7.41</td>
<td>7.33</td>
<td>6.79</td>
</tr>
<tr>
<td>8</td>
<td>15.51</td>
<td>4.78</td>
<td>4.58</td>
<td>4.54</td>
<td>4.19</td>
</tr>
<tr>
<td>16</td>
<td>11.72</td>
<td>2.85</td>
<td>2.71</td>
<td>2.62</td>
<td>2.39</td>
</tr>
<tr>
<td>32</td>
<td>8.01</td>
<td>1.85</td>
<td>1.79</td>
<td>1.66</td>
<td>1.47</td>
</tr>
</tbody>
</table>

Fig. 8. Effects of various optimizations (sec).
Global Arrays

• Library for parallelization abstraction
  • On distributed memory systems (clusters)
  • SPMD model

• Idea is to program as if shared memory, but move data between distributed memory and local memory as needed
  • Only operate on local data within each process

• Compatible with MPI, so can mix GA calls with MPI calls as needed
  • Built on top of ARMCI library for one-sided communication (put/get) – portable and efficient
  • One-sided can be more efficient than send/receive, as shown for some applications, since less synchronization
Global Arrays

• Programmer can map both ways between global and local views of data objects (arrays)
  • But only compute on local view

• GA is also aware of SMP (multi-core) nodes
  • To support “mirrored view” – caching distributed memory data in shared memory for multiple processes to use

• Also has direct support for ghost cells
  • To avoid distributed to local copies for structured grid applications
  • And for periodic boundary conditions

• Paper also talks about sparse data management
  • But not clear how efficient GA is for computing with sparse matrices/vectors
Global Arrays

• Data parallel interfaces to operate on global arrays
  • To interface with other libraries like BLAS, SCALAPACK to perform data parallel collective operations

• Disk Resident Arrays allow extending global arrays to out-of-core
  • Basically distributed memory stored on (local) disks
  • With operations to move data between disks (instead of distributed memory) and local memory in each process

• Support for mapping global arrays onto subsets of processors

• Many similarities to Multiblock Parti, but also supports copies from global to local view

• Performance results show good scaling on several applications for parallel systems available at that time
  • All the applications employ large, dense, multi-dim data grids
  • And can take advantage of both low-latency and high-bandwidth networks (through ARMCI)
Overall Conclusion

• We can get close to hand-coded performance with these systems
• Are they easier to use?