Hypergraph Partitioning for Automatic Memory Hierarchy Management

Sriram Krishnamoorthy

Sriram Krishnamoorthy¹, Umit Catalyurek², Jarek Nieplocha³, Atanas Rountev¹ and P. Sadayappan¹

¹Dept. of CSE, ²Dept. of ECE
The Ohio State University
³Pacific Northwest National Lab

*Supported in part by Dept. of Energy & National Science Foundation*
Outline

- Background
- Motivation
- Programming Abstractions
- Disk I/O Minimization Problem
- Modeling of the Minimization Problem
- Results
- Related Work
- Conclusion
Background

- Computations with data too large to fit into physical memory
- Out-of-core programming challenging
  - Schedule computation
  - Schedule of data movement between disk and main memory
  - Ensure data required by computation is available in memory
  - Ensure size of data in memory at any time does not exceed the memory available
- Automatic Memory Hierarchy Management
  - Relieve burden of out-of-core programming
  - User specifies computation
  - Runtime system schedules disk I/O and computation
Tensor Contraction Engine

- Automatic transformation from high-level specification
  - Chemist specifies computation in high-level mathematical form
  - Synthesis system transforms it to efficient parallel program
  - Code is tailored to target machine
  - Code can be optimized for specific molecules being modeled

- Multi-institutional collaboration (OSU, LSU, U. Waterloo, ORNL, PNNL, U. Florida)

- Significant interest in quantum chemistry community

- Built on top of the Gobal Arrays library

- Tensor Contractions: Multi-dimensional summations of products of block-sparse arrays
  
  \[ p_3, p_4, p_5, p_7 : V(100, 100, 60, 60) \quad h_1, h_2, h_6, h_8 : O(40, 40, 20, 20) \]

  \[ i_1[h_6, p_3, h_1, p_5] + = v_1[h_6, p_3, h_1, p_5] \]

  \[ i_1[h_6, p_3, h_1, p_5] + = t[p_3, p_7, h_1, h_8] * v_2[h_6, h_8, p_5, p_7] \]

  \[ i_0[p_3, p_4, h_1, h_2] + = t[p_3, p_5, h_1, h_6] * i_1[h_6, p_4, h_2, p_5] \]
Global Arrays Library

Distributed dense arrays that can be accessed through a shared view

Physically Distributed data

single shared data structure with indexing.
e.g., access $A(4,3)$ rather than $A_{local}(1,3)$ on process 4

Global Address Space
Global Arrays Model of Computation

- Shared memory view for distributed dense arrays
- MPI-Compatible; Currently usable with Fortran, C, C++, Python
- Data locality and granularity control similar to message passing model
- Used in large scale efforts, e.g. NWChem (million+ lines/code)
Pros and Cons of the GA model

● Advantages
  ◆ Provides convenient global-shared view
  ◆ Get-compute-put model ensures user focus on data-locality optimization => good performance
  ◆ Inter-operates with MPI to enable general applicability

● Limitations
  ◆ Only supports dense multi-dimensional arrays
  ◆ Data view more convenient than MPI, but computation specification is still process-centric
  ◆ No support for load balancing of irregular computations
Programming Abstractions

- Decoupled task and data abstractions
- Layered, multi-level data abstraction
  - Global-shared view and transparent access
  - Chunked access for efficiency
  - Multi-dimensional block-sparse array represented as collection of small dense multi-dimensional bricks (chunks)
  - Bricks distributed among local disks of processors
- Computation Abstraction
  - Non-process-specific collection of independent tasks
  - Only non-local data task can access: bricks in global data
  - Specification of tasks includes the global data accessed: runtime information
  - Automatic scheduling of computation, communication, and disk I/O
Disk I/O Minimization Problem

- Block-sparse tensors
  - Irregular data access pattern
  - Techniques such as tiling — non-trivial to apply
- Data too large to fit into collective physical memory in a parallel system
- Problem characteristics known only at runtime
- Schedule computation and data movement
  - Minimize disk I/O
  - Maximize reuse
Hypergraph Partitioning Problem

- Set of vertices/cells and hyperedges/nets with weights
- **Given**: #parts \( p \)
- **Solution**: A vertex partition of the hypergraph
- **Objective**: Minimize cost of *cut-net* weights
  - Weight of cut-nets counted once
  - Weight of cut-nets counted for every cut
- **Constraint**: Balance for all parts
  - Sum of vertex weights in each part
  - Sum of net weights in each part
- Efficient solutions exist (PaToH)
- Mismatch
  - Typically used in context of parallelization
  - #parts unknown
  - No memory limit constraint, only balancing constraint
Modeling Disk I/O Minimization

- **Vertices**: One per task, weight = computation cost
- **Hyper-edges**: One per data-brick, Weight = size of brick (I/O cost)
- Each hyper-edge connects tasks accessing the brick
- **Minimize**: Sum of cut-net weights, counting every cut
- **Constraint**: Balance net weights
  - Balancing computation cost (vertex weights) unnecessary
  - Ensures a feasible solution, if one exists
- **#parts** - Number of stages in the computation
- **Dynamically determine #parts**
  - Modify recursive procedure of hypergraph partitioning
  - Stop further partitioning when memory constraint satisfied
Read-Once Partitioning

- One-level Partitioning
  - Memory usage = Sum of data sizes accessed in a part
  - No refined reuse relationships
  - All tasks within a part have reuse, and none outside

- Read-Once Partitioning
  - Group tasks into steps
  - Identify data common across steps and load into memory
  - For each step, read non-common (step-exclusive) data, process tasks, and write/discard step-exclusive data
  - Better utilization of memory available -> reduced disk I/O
Read-once Partitioning Example

Disk I/O Cost: 9 data elements

Disk I/O Cost: 8 data elements
# Results - Configuration

<table>
<thead>
<tr>
<th></th>
<th>ia64-osc</th>
<th>p4-osc</th>
<th>ia64-pnl</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processor</strong></td>
<td>Dual Itanium-2</td>
<td>Dual Pentium 4</td>
<td>Dual Itanium-2</td>
</tr>
<tr>
<td><strong>Clock Frequency</strong></td>
<td>900MHz</td>
<td>2.4GHz</td>
<td>1GHz</td>
</tr>
<tr>
<td><strong>Physical Memory</strong></td>
<td>4GB</td>
<td>4GB</td>
<td>6GB</td>
</tr>
<tr>
<td><strong>Local Disk</strong></td>
<td>80GB</td>
<td>80GB</td>
<td>80GB</td>
</tr>
<tr>
<td><strong>Messaging Layer</strong></td>
<td>GM</td>
<td>VAPI</td>
<td>GM</td>
</tr>
</tbody>
</table>

- One process per node used in all experiments
- **Alternative Scheme: GetNext**
  - Replicate, compute, reconcile model
  - Based on codes in NWChem
- **Evaluation on CCD sub-computation described earlier**
Results - Disk I/O Cost on ia64-osc

Factor of Improvement: 6.5
Results - Disk I/O Cost on p4-osc

Factor of Improvement: 11.0
Results - Disk I/O Cost on ia64-pnl

Factor of Improvement: 7.9
## Results - Total Execution Time

<table>
<thead>
<tr>
<th>System</th>
<th>Scheme</th>
<th>nprocs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ia64-osc</td>
<td>GetNext</td>
<td>9710</td>
</tr>
<tr>
<td></td>
<td>HpGraph</td>
<td>9244</td>
</tr>
<tr>
<td>p4-osc</td>
<td>GetNext</td>
<td>13717</td>
</tr>
<tr>
<td></td>
<td>HpGraph</td>
<td>11700</td>
</tr>
<tr>
<td>ia64-pnl</td>
<td>GetNext</td>
<td>7928</td>
</tr>
<tr>
<td></td>
<td>HpGraph</td>
<td>7564</td>
</tr>
</tbody>
</table>

Table 1: Turnaround times for the CCD sub-computation

**Sequential Speedup on p4-osc : 14.7%**
Results - Execution Time Speed-ups

- GetNext-ia64-osc
- HpGraph-ia64-osc
- GetNext-p4-osc
- HpGraph-p4-osc
- GetNext-ia64-pnl
- HpGraph-ia64-pnl
- Linear
Results - Percentage of Computation overhead

![Graph showing computation overhead percentage vs number of processes for different tasks and platforms.](image-url)
Related Work

- Sparse or block-sparse matrices in parallel libraries supporting sparse linear algebra
  - Aztec, PETSc etc.
  - Load balancing

- Dense tiling - challenging in this context

- Application of hypergraph partitioning
  - Primarily used for parallelization

- Unaware of runtime support for:
  - Flexible global-shared abstractions for semi-structured data
  - Locality-aware scheduling of parallel tasks
  - Transparent memory hierarchy management
Conclusion

● Abstractions for locality-aware transparent memory hierarchy management
  - Data chunked for efficient access
  - Computation specification includes locality

● Modeled disk I/O minimization as hypergraph partitioning

● Partitioning to reduce disk I/O while attaining a feasible solution

● Demonstrated sequential speedup

● Demonstrated better scalability
Questions?