Multilevel Acyclic Partitioning of Directed Acyclic Graphs for Enhancing Data Locality

Julien Herrmann\textsuperscript{1}, Bora Uçar\textsuperscript{2}, Kamer Kaya\textsuperscript{3}, Aravind Sukumaran Rajam\textsuperscript{4}, Fabrice Rastello\textsuperscript{5}, P. Sadayappan\textsuperscript{4}, Ümit V. Çatalyürek\textsuperscript{1}

\textsuperscript{1}School of Computational Science and Engineering, Georgia Institute of Technology
\textsuperscript{2}CNRS and LIP, ENS Lyon, France
\textsuperscript{3}Computer Science and Engineering, Sabancı University, Turkey
\textsuperscript{4}Department of Computer Science and Engineering, The Ohio State University
\textsuperscript{5}INRIA Grenoble Rhone-Alpes, France

SIAM CSE
February 28th, 2017 – Atlanta, GA
Outline

1. Motivation
2. Acyclic Partitioning
3. Directed Multilevel Graph Partitioning
4. Experimental Results
Motivation

- Scheduling for task-based runtime systems by Ç et al.
- **Characterization of the Data Movement Complexity of Algorithms** by
  P. Sadayappan, A. Rountev, L-N. Pouchet, A. Sidiropoulos, N. Fauzia, V. Elango, and M. Ravishankar, The Ohio State University
  J. Ramanujam, Louisiana State University
  F. Rastello, INRIA-Grenoble
Motivation

- Data movement is much more expensive in computer systems than arithmetic operations (Flops)
  - Performance: latency as well as throughput
  - Energy
Motivation

- Data movement is much more expensive in computer systems than arithmetic operations (Flops)
  - Performance: latency as well as throughput
  - Energy
- Computational complexity alone (number of ops executed) cannot be sole (or even primary) criterion of algorithm choice
Motivation

- Data movement is much more expensive in computer systems than arithmetic operations (Flops)
  - Performance: latency as well as throughput
  - Energy
- Computational complexity alone (number of ops executed) cannot be sole (or even primary) criterion of algorithm choice
- But what is the inherent data movement complexity of an alg.?
  - Computational complexity well understood; invariant to transforms
  - Data access complexity is not well characterized today: cost is affected by code transformations and also capacity of registers/caches
Motivation

- Data movement is much more expensive in computer systems than arithmetic operations (Flops)
  - Performance: latency as well as throughput
  - Energy
- Computational complexity alone (number of ops executed) cannot be sole (or even primary) criterion of algorithm choice
- But what is the inherent data movement complexity of an alg.?
  - Computational complexity well understood; invariant to transforms
  - Data access complexity is not well characterized today: cost is affected by code transformations and also capacity of registers/caches
- Understanding data movement complexity is important:
  - Algorithm choice between alternatives e.g., will Krylov subspace solvers and FFTs continue to be as popular in the future?
  - Arch. parameters: minimum cache capacity and/or bus bw. needed to support inherent data movement needs of an alg.
  - Assessing manual/compiler optimizations: How much further improvement potential is there, beyond current optimizations?
FLOPs almost free; data movement cost is dominant

Minimizing amount of data movement increasingly critical

Source: Jim Demmel, John Shalf then P. Sadayappan
### Computational vs Data Move Complexity

**Untiled version**

```plaintext
for (i=1; i<N-1; i++)
    for (j=1; j<N-1; j++)
```

**Tiled Version**

```plaintext
for(it = 1; it<N-1; it +=B)
    for(jt = 1; jt<N-1; jt +=B)
        for(i = it; i < min(it+B, N-1); i++)
            for(j = jt; j < min(jt+B, N-1); j++)
```

### Questions

- Can we achieve lower cache misses than this tiled version?
- How can we know when much further improvement is not possible?
- What is the lowest achievable data movement cost among all possible equivalent versions of a computation?

Current performance tools and methodologies do not address this.
Computational vs Data Move Complexity

- Both have Comp. Complexity $(N - 1)^2$ OPs.
- Data movement cost different for two versions
- Also depends on cache size

Question: Can we achieve lower cache misses than this tiled version?

How can we know when much further improvement is not possible?

Question: What is the lowest achievable data movement cost among all possible equivalent versions of a computation?

Current performance tools and methodologies do not address this
Computational vs Data Move Complexity

- Both have Comp. Complexity \((N - 1)^2\) OPs.
  - Data movement cost different for two versions
  - Also depends on cache size
- Question: Can we achieve lower cache misses than this tiled version? How can we know when much further improvement is not possible?
Both have Comp. Complexity \((N - 1)^2\) OPs.
- Data movement cost different for two versions
- Also depends on cache size

Question: Can we achieve lower cache misses than this tiled version? How can we know when much further improvement is not possible?

Question: What is the lowest achievable data movement cost among all possible equivalent versions of a #computation?
Computational vs Data Move Complexity

- Both have Comp. Complexity $(N - 1)^2$ OPs.
  - Data movement cost different for two versions
  - Also depends on cache size
- Question: Can we achieve lower cache misses than this tiled version? How can we know when much further improvement is not possible?
- Question: What is the lowest achievable data movement cost among all possible equivalent versions of a computation?
- Current performance tools and methodologies do not address this
CDAG abstraction: Vertex = operation, edges = data dep.
**Modeling Data Move Complexity: CDAG**

CDAG abstraction: Vertex = operation, edges = data dep.

- 2-level memory hierarchy with $S$ fast mem locs. & infinite slow mem. locs.
  - To compute a vertex, predecessor must hold values in fast mem.
  - Limited fast memory $\Rightarrow$ computed values may need to be temporarily stored in slow memory and reloaded.

Untiled version

```plaintext
for (i=1; i<N-1; i++)
  for (j=1; j<N-1; j++)
```

Tiled Version

```plaintext
for(it = 1; it<N-1; it +=B)
  for(jt = 1; jt<N-1; jt +=B)
    for(i = it; i < min(it+B, N-1); i++)
      for(j = jt; j < min(jt+B, N-1); j++)
```

**CDAG for N=6**

- For $N=6$:
  - Inherent data movement complexity of
  - Computational vs. Data Movement Complexity
  - Model of Data Movement Complexity: CDAG
  - CDAG: Minimal #loads+#stores
  - Computation: Attempting to find a different order of execution of the operations that can improve the reuse-distance profile compared to that of the given program's sequential execution trace. If this analysis reveals a significantly improved reuse distance profile, it suggests that suitable source code transformations are possible.

**Computational vs. Data Movement Complexity**

- for (i=1; i<N-1; i++)
  - for (j=1; j<N-1; j++)

**Multilevel Acyclic Partitioning of Directed Acyclic Graphs for Enhancing Data Locality**

SIAM CSE

February 28th, 2017
Modeling Data Move Complexity: CDAG

- CDAG abstraction: Vertex = operation, edges = data dep.
- 2-level memory hierarchy with $S$ fast mem locs. & infinite slow mem. locs.
  - To compute a vertex, predecessor must hold values in fast mem.
  - Limited fast memory $\Rightarrow$ computed values may need to be temporarily stored in slow memory and reloaded
- Inherent data movement complexity of CDAG: Minimal $\#$loads+$\#$stores
In this section, we provide details on our algorithm for convex partitioning of CDAGs, which is at its heart a problem of deciding the correct execution order of the vertices in a CDAG.

The dynamic analysis involves the following steps:

1. Form a CDAG from the execution trace.
2. Generate a sequential execution trace of a program.
4. For each possible execution order of the vertices, perform convex partitioning to obtain a min-cost schedule.
5. Evaluate the reuse-distance profile for each possible schedule.
6. Choose the execution order of the vertices that yields the best reuse-distance profile.

Convex partitioning is analogous to tiling the iteration space of a regular nested loop computation. Multi-level convex partitioning involves creating multiple levels of tiling, each of which can further improve the data locality. The key idea behind the work presented in this article is to perform analysis on the CDAG of a computation, abstracting away the sequential schedule of operations and imposing instead a parallel schedule of operations.

Although a CDAG is derived from analysis of dependences between instances of statements executed in a program, it abstracts away that sequential schedule of operations and only imposes a parallel schedule of operations. This abstraction is crucial for improving data locality, as it allows us to focus on the dependencies between operations rather than their sequential order.

The key idea behind convex partitioning is to divide the CDAG into convex partitions, where each partition contains a set of operations that can be executed in parallel without violating any dependences. This partitioning can be performed at multiple levels of granularity, with higher levels of partitioning providing more opportunities for parallel execution.

Minimum possible data movement cost?

No known effective solution to problem

Develop upper bounds on min-cost

Develop lower bounds on min-cost

A version of Figure 2 for multilevel convex partitioning.

CDAG for N=6

Untiled version

Tiled Version

for (i=1; i<N-1; i++)
  for (j=1; j<N-1; j++)

for (it = 1; it<N-1; it +=B)
  for (jt = 1; jt<N-1; jt +=B)
    for (i = it; i < min(it+B, N-1); i++)
      for (j = jt; j < min(jt+B, N-1); j++)
Data Movement Upper Bounds

- Perform acyclic partitioning of the CDAG
- Assign each node in a single acyclic part
- Acyclic partitioning of a CDAG $\approx$ Tiling the iteration space
- Each part is acyclic
  - Can be executed atomically
  - No cyclic data dependence among parts
- Topologically sorted order of the acyclic parts $\Rightarrow$ a valid execution order
- **To Do:** Develop scalable distributed acyclic partitioning algorithm for CDAGs.
Outline

1 Motivation

2 Acyclic Partitioning

3 Directed Multilevel Graph Partitioning

4 Experimental Results
Balanced Acyclic Partitioning

Minimal edge cut:
Balanced Acyclic Partitioning

Minimal edge cut:

- Undirected graph: 2
Balanced Acyclic Partitioning

Minimal edge cut:

- Undirected graph: 2
Balanced Acyclic Partitioning

Minimal edge cut:

- Undirected graph: 2
Balanced Acyclic Partitioning

Minimal edge cut:
- Undirected graph: 2
- Directed graph: 3
Objective Function

**Objective 1**
Minimize the edge cut between parts

**Objective 2**
Minimize the total volume of communication between parts (edge cut counting edges coming from the same node only once)

**Objective 3**
At the application level:
- Maximize the performance
- Minimize the cache miss count
- ...
### Constraint 1
Upper bound on the weights of each part.

### Constraint 2
Upper bound on the weight of each part plus the sum of weights of the boundary vertices that are sources of the part’s incoming edges.

### Constraint 3
There should exist a traversal of the graph such that **alive** data fit into the cache at any moment.
Fauzia et al. Algorithm

- Vertices are traversed in a topological order with tunable depth and breadth priorities.
- Vertices are assigned to the current partition set until the maximum number of vertices that would be active during the computation of the partition set reaches a specified cache size.

- Partition sizes can be larger than the size of the cache (Constraint 3).
- This differs from our problem (Constraint 1).
Outline

1. Motivation
2. Acyclic Partitioning
3. Directed Multilevel Graph Partitioning
4. Experimental Results
Multilevel scheme

Three phases

- **Coarsening**: obtain smaller and similar graphs to the original, until either a minimum vertex count is reached or reduction on number of vertices is lower than a threshold.
- **Initial Partitioning**: find a solution for the smallest graph.
- **Uncoarsening**: Project the initial solution to the coarser graphs and refine it iteratively until a solution for the original graph obtained.
Coarsening

- Make sure not to create any cycle when matching
Coarsening

- Make sure not to create any cycle when matching
Coarsening

- Make sure not to create any cycle when matching.
Coarsening

- Make sure not to create any cycle when matching
- Find optimal matching \(\Rightarrow\) too costly.

Matching Restriction

Let \(G = (V, E)\) be a CDAG and \(M = \{(u_1, v_1), \ldots, (u_k, v_k)\}\) a matching such that:

\[(u_i, v_j) \in M, \text{top level}(v_i) = \text{top level}(u_i) + 1\]

any pair of \((u_i, v_i)\) and \((u_j, v_j)\) \(\in M\), either

\[(u_i, v_j) \text{ not in } E \text{ or } \text{top level}(u_i) \neq \text{top level}(v_j) + 1\]

Then, the coarse graph is acyclic.
Coarsening

- Make sure not to create any cycle when matching
- Find optimal matching \(\Rightarrow\) too costly.

Matching Restriction

Let \(G = (V, E)\) be a CDAG and \(M = \{(u_1, v_1), \ldots, (u_k, v_k)\}\) a matching such that:

- \((u_i, v_j) \in M, \quad \text{top} \_\text{level}(v_i) = \text{top} \_\text{level}(u_i) + 1\)
- any pair of \((u_i, v_i)\) and \((u_j, v_j) \in M\), either
  - \((u_i, v_j)\) not in \(E\) or
  - \(\text{top} \_\text{level}(u_i) \neq \text{top} \_\text{level}(v_j) + 1\)

Then, the coarse graph is acyclic.
Initial partitioning

Kernighan’s Algorithm (1971)
- Given a total order of vertices, vertex weights, edge costs, an upper bound on part weights,
- finds a cut so that part weights respect the upper bound and minimizes the edge cut, where the parts are contiguous.

We find a topological order with an attempt to reduce the maximum edge cut at a point (heuristically).

Then, feed this to Kernighan’s algorithm.
Kernighan’s Algorithm: Dynamic Programming

\[ T(x) = \min_y \{ T(y) + C(x, y) \} \]

- \( T(x) \): the best cost of cutting right before \( x \).
- \( C(x, y) \): the additional cut edges at \( x \), given the previous cut was at \( y \). Do not count twice.
- \( y_\ell \): the weight of the part \( y_\ell, \ldots, x - 1 \) is acceptable, but \( y_{\ell-1}, \ldots, x - 1 \) is not.
Uncoarsening

- Moving a node to another partition set can violate acyclicity

Refinement Restriction

- Define a topological order among parts.

- A node can only be moved to the part of its incoming nodes with the highest rank in the topological order or the part of its outgoing nodes with the smallest rank in the topological order.

Then, the refinement does not violate acyclicity.

- Nodes are moved as long as balance constraints are matched and edge cut is improving.
Uncoarsening

- Moving a node to another partition set can violate acyclicity

**Refinement Restriction**

- Define a topological order among parts.
- A node can only be moved to the part of its incoming nodes with the highest rank in the topological order or the part of its outgoing nodes with the smallest rank in the topological order.

Then, the refinement does not violate acyclicity.

- Nodes are moved as long as balance constraints are matched and edge cut is improving.
Uncoarsening

- Moving a node to another partition set can violate acyclicity

Refinement Restriction

- Define a topological order among parts.
- A node can only be moved to the part of its incoming nodes with the highest rank in the topological order or the part of its outgoing nodes with the smallest rank in the topological order.

Then, the refinement does not violate acyclicity.

- Nodes are moved as long as balance constraints are matched and edge cut is improving.
1 Motivation
2 Acyclic Partitioning
3 Directed Multilevel Graph Partitioning
4 Experimental Results
Coarsening ratios of Metis and dMLGP are very similar.

Directed coarsening does not seem to be too restrictive.
## Graphs Properties

Instances from the Polyhedral Benchmark Suite.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Parameters</th>
<th>#vertex</th>
<th>#edge</th>
<th>out-deg.</th>
<th>deg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2mm</td>
<td>P=10, Q=20, R=30, S=40</td>
<td>36,500</td>
<td>62,200</td>
<td>40</td>
<td>1.704</td>
</tr>
<tr>
<td>3mm</td>
<td>P=10, Q=20, R=30, S=40, T=50</td>
<td>111,900</td>
<td>214,600</td>
<td>40</td>
<td>1.918</td>
</tr>
<tr>
<td>adi</td>
<td>T=20, N=30</td>
<td>596,695</td>
<td>1,059,590</td>
<td>109,760</td>
<td>1.776</td>
</tr>
<tr>
<td>atax</td>
<td>M=210, N=230</td>
<td>241,730</td>
<td>385,960</td>
<td>230</td>
<td>1.597</td>
</tr>
<tr>
<td>covariance</td>
<td>M=50, N=70</td>
<td>191,600</td>
<td>368,775</td>
<td>70</td>
<td>1.925</td>
</tr>
<tr>
<td>doitgen</td>
<td>P=10, Q=15, R=20</td>
<td>123,400</td>
<td>237,000</td>
<td>150</td>
<td>1.921</td>
</tr>
<tr>
<td>durbin</td>
<td>N=250</td>
<td>126,246</td>
<td>250,993</td>
<td>252</td>
<td>1.988</td>
</tr>
<tr>
<td>fdtd-2d</td>
<td>T=20, X=30, Y=40</td>
<td>256,479</td>
<td>436,580</td>
<td>60</td>
<td>1.702</td>
</tr>
<tr>
<td>gemm</td>
<td>P=60, Q=70, R=80</td>
<td>1,026,800</td>
<td>1,684,200</td>
<td>70</td>
<td>1.640</td>
</tr>
<tr>
<td>gmm</td>
<td>N=120</td>
<td>159,480</td>
<td>259,440</td>
<td>120</td>
<td>1.627</td>
</tr>
<tr>
<td>gesummv</td>
<td>N=250</td>
<td>376,000</td>
<td>500,500</td>
<td>500</td>
<td>1.331</td>
</tr>
<tr>
<td>heat-3d</td>
<td>T=40, N=20</td>
<td>308,480</td>
<td>491,520</td>
<td>20</td>
<td>1.593</td>
</tr>
<tr>
<td>jacobi-1d</td>
<td>T=100, N=40</td>
<td>239,202</td>
<td>398,000</td>
<td>100</td>
<td>1.664</td>
</tr>
<tr>
<td>jacobi-2d</td>
<td>T=20, N=30</td>
<td>157,808</td>
<td>282,240</td>
<td>20</td>
<td>1.789</td>
</tr>
<tr>
<td>lu</td>
<td>N=80</td>
<td>344,520</td>
<td>676,240</td>
<td>79</td>
<td>1.963</td>
</tr>
<tr>
<td>ludcmp</td>
<td>N=80</td>
<td>357,320</td>
<td>701,680</td>
<td>80</td>
<td>1.964</td>
</tr>
<tr>
<td>mvt</td>
<td>N=200</td>
<td>200,800</td>
<td>320,000</td>
<td>200</td>
<td>1.594</td>
</tr>
<tr>
<td>seidel-2d</td>
<td>M=20, N=40</td>
<td>261,520</td>
<td>490,960</td>
<td>60</td>
<td>1.877</td>
</tr>
<tr>
<td>symm</td>
<td>M=40, N=60</td>
<td>254,020</td>
<td>440,400</td>
<td>120</td>
<td>1.734</td>
</tr>
<tr>
<td>syr2k</td>
<td>M=20, N=30</td>
<td>111,000</td>
<td>180,900</td>
<td>60</td>
<td>1.630</td>
</tr>
<tr>
<td>syrk</td>
<td>M=60, N=80</td>
<td>594,480</td>
<td>975,240</td>
<td>81</td>
<td>1.640</td>
</tr>
<tr>
<td>trisolv</td>
<td>N=400</td>
<td>240,600</td>
<td>320,000</td>
<td>399</td>
<td>1.330</td>
</tr>
<tr>
<td>trmm</td>
<td>M=60, N=80</td>
<td>294,570</td>
<td>571,200</td>
<td>80</td>
<td>1.939</td>
</tr>
</tbody>
</table>

Multilevel Acyclic Partitioning of Directed Acyclic Graphs for Enhancing Data Locality
SIAM CSE
February 28th, 2017
Experimental Results

- Average results on 100 runs.
- Imbalance ratio of 3%.

Multilevel Acyclic Partitioning of Directed Acyclic Graphs for Enhancing Data Locality
SIAM CSE
February 28th, 2017
Results

- Inter-partitions edges have a weight of 11 nanoseconds to model L3 cache latency.
- Intra-partitions edges have a weight of 1 nanoseconds to model L1 cache latency.
- Vertices have a latency of 1 nanoseconds to model task execution.
Data Movement

- Data movement costs will be increasingly dominant over computation costs, for both performance and energy/power.
  - Important to understand inherent constraints on minimal possible data movement for an algorithm as a function of storage capacity.
- Need advances in theory and software tools for modeling data movement complexity, and methodologies for application to algorithm analysis and algorithm-architecture co-design.
  - Significant benefit of lower bounds analysis: schedule-independent, unlike standard performance modeling; especially powerful for analysis of composite applications.
Summary and Ongoing/Future Work

Data Movement

- Data movement costs will be increasingly dominant over computation costs, for both performance and energy/power
  - Important to understand inherent constraints on minimal possible data movement for an algorithm as a function of storage capacity
- Need advances in theory and software tools for modeling data movement complexity, and methodologies for application to algorithm analysis and algorithm-architecture co-design
  - Significant benefit of lower bounds analysis: schedule-independent, unlike standard performance modeling; especially powerful for analysis of composite applications

Directed Graph Partitioning

- Implement agglomerative matching, i.e., clustering.
- Use directed graph partitioning to automatically improve data locality for compiler optimizations.
Thanks

To P. Sadayappan for sharing his motivation slides.

More information

contact: umit@gatech.edu
visit: http://cc.gatech.edu/~umit