Improving Locality for Irregular Codes

- Improving locality for better cache performance
- Irregular codes
  - run-time optimization based on access pattern
  - data/computation reordering
- Adaptive irregular codes
  - access pattern changes
  - periodically apply reordering

Irregular codes

\[
\text{do } i = 1, M \\
\quad \ldots = x(\text{idx}(i))
\]

Adaptive irregular codes

\[
\text{do } t = 1, \text{time} \\
\quad \text{if } (\text{change}) \text{ idx}(\ldots) = \ldots \\
\quad \text{do } i = 1, M \\
\quad \quad \ldots = x(\text{idx}(i))
\]
Applying Transformations

STEP 1: CLASSIFY BY ACCESS PATTERN

number of distinctive accesses

3 access(es)
do i = 1, M
  ... = x ( idx1 (i) )
  ... = x ( idx2 (i) )
  ... = x ( idx3 (i) )

STEP 2: COMPUTATION/DATA REORDERING

1 access:
  - computation reordering
2 or more accesses:
  - data reordering
  - computation reordering
Data Reordering

- Data reordering
  - consecutive packing (CPACK)
  - geometric partitioning (RCB)
    - coordinate information needed
- graph partitioning
  - multi-level graph partitioning (METIS)
  - graph clustering (GPART) : low overhead

![Diagram of data reordering process](image-url)
Computation Reordering

- Computation reordering
  sorting computation according to data location
  lexicographical sorting

\[(b,c),(d,e),(a,b),(c,d)\] => \[(a,b),(b,c),(c,d),(d,e)\]

data location:

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
3 & 1 & 5 & 2 & 4 \\
\end{array}
\]

location:

- after data reordering
- computation reordering
New Low-Overhead Graph Partitioning (GPART)

- Hierarchical clustering with randomly chosen neighbors
- Contiguously place nodes in a partition
- Preserve hierarchy in memory layout
Performance of Reordering

<table>
<thead>
<tr>
<th></th>
<th>Average Normalized Overhead</th>
<th>Exec-Time (at infinity)</th>
</tr>
</thead>
<tbody>
<tr>
<td>METIS</td>
<td>16.74</td>
<td>0.695</td>
</tr>
<tr>
<td>RCB</td>
<td>11.15</td>
<td>0.692</td>
</tr>
<tr>
<td>GPART</td>
<td>3.87</td>
<td>0.698</td>
</tr>
<tr>
<td>CPACK</td>
<td>1.00</td>
<td>0.799</td>
</tr>
</tbody>
</table>
Optimizing Adaptive Computation

- Reordering is not required after every access pattern change

- which transformation?
- how often to transform?

Sampling Iterations

Measured Information

- overhead
- benefit
- degradation rate

Input Program Characteristics

- number of iterations
- adaptation frequency

Cost Model

Running Parameters

- which transformation?
- how often to transform?
Adaptive Computation - Cost Model

- Net benefit: \( G(n) = A - n \times \text{Overhead} \)
- Find maximal point of \( G(n) \) (\( n > 0 \)), \( G'(n_0) = 0 \)

\[ m = \tan \theta \]

\( n \): number of transformations applied during total \( t \) iterations
\( m \): performance degradation rate
\( t \): total number of iterations
Adaptive Computation - Characteristics

- Apply transformation once: performance degrades
- Apply transformation after each change: high overhead
- How often? Which algorithm?
Experimental Validation of Cost Model

Vertical bars: predicted by cost model
number of times transformations applied
maximum gain for each algorithm