Locality-aware Dynamic Mapping for Multithreaded Applications

Betül Demiröz
betul.demiroz@marmara.edu.tr

Marmara University
Computer Engineering Department
Istanbul Turkey
Outline

- Locality-aware Mapping Algorithm
- Sparse vector-matrix Multiply
- Chunks
- Data-to-chunk Distribution
- Bins
- Similarity Calculation
- Experimental Study
Locality-aware mapping algorithm

- Locality analysis
  - Extracts data access patterns
  - Predicts runtime cache behaviour

- Our aim is to increase
  - Data reuse of on-chip shared cache
  - Performance of the application

- Can be applied to any multithreaded application running on CMPs
  - Especially for the ones with irregular data access patterns
Sparse vector-matrix multiply

- Used to validate the effectiveness of our algorithm
- Requires many indirect and irregular memory accesses
- Performs $A \times X = B$, where
  - $A$ and $B$ are matrices
  - $X$ is the vector shared by all application threads
- Data accessed on vector $X$ is used for locality analysis
Chunks

• Computations performed on consecutive blocks of data are represented with chunks
  ▫ OR operation is performed on consecutive computations
  ▫ The resulting bit vector is represented as a chunk

• Balance load among threads
Chunk Creation - I

- Assume that there are 8 computations
- Our aim is to create 4 chunks
- Each chunk should contain two consecutive computations

\[\begin{array}{c|c|c|c|c|c|c|c|c|c}
1^{st} \text{ computation} & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0  \\
2^{nd} \text{ computation} & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0  \\
\end{array}\]

OR

\[\begin{array}{c|c|c|c|c|c|c|c|c|c}
1^{st} \text{ chunk} & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1  \\
\end{array}\]
### Chunk Creation - II

<table>
<thead>
<tr>
<th>Application Data:</th>
<th>Bit Vectors:</th>
</tr>
</thead>
</table>

#### 1st chunk

<table>
<thead>
<tr>
<th>1st comp.</th>
<th>1 0 0 0 0 1 1 1 0 0 0 0 1 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd comp.</td>
<td>1 1 1 0 0 0 0 0 1 0 0 0 0 1 0</td>
</tr>
<tr>
<td>3rd comp.</td>
<td>0 0 0 0 1 1 0 0 1 1 0 1 0 0 0</td>
</tr>
<tr>
<td>4th comp.</td>
<td>0 1 0 0 0 0 0 1 1 0 0 0 0 1 1</td>
</tr>
<tr>
<td>5th comp.</td>
<td>0 1 0 1 0 1 0 0 0 1 0 1 0 0 0</td>
</tr>
<tr>
<td>6th comp.</td>
<td>0 0 0 1 1 0 0 0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>7th comp.</td>
<td>1 0 1 0 0 0 1 0 1 0 0 0 1 0 0</td>
</tr>
<tr>
<td>8th comp.</td>
<td>1 1 1 0 0 0 1 1 0 0 0 0 1 1 0</td>
</tr>
</tbody>
</table>

#### 2nd chunk

| 1st comp. | 1 1 0 0 1 1 1 1 0 0 0 0 1 1 0 |
| 2nd comp. | 0 1 0 0 1 1 0 0 1 1 0 1 0 0 0 1 |
| 3rd comp. | 0 1 0 1 1 1 0 0 0 1 0 1 0 0 0 1 |
| 4th comp. | 1 1 1 0 0 1 1 1 0 0 0 0 1 1 0 |

#### 3rd chunk

| 1st comp. | 1 1 0 0 1 1 1 1 0 0 0 0 1 1 0 |
| 2nd comp. | 0 1 0 0 1 1 0 0 1 1 0 1 0 0 0 1 |
| 3rd comp. | 0 1 0 1 1 1 0 0 0 1 0 1 0 0 0 1 |
| 4th comp. | 1 1 1 0 0 0 1 1 1 0 0 0 0 1 1 0 |

#### 4th chunk

| 1st comp. | 1 1 0 0 1 1 1 1 0 0 0 0 1 1 0 |
| 2nd comp. | 0 1 0 0 1 1 0 0 1 1 0 1 0 0 0 1 |
| 3rd comp. | 0 1 0 1 1 1 0 0 0 1 0 1 0 0 0 1 |
| 4th comp. | 1 1 1 0 0 0 1 1 1 0 0 0 0 1 1 0 |
Bins

- Calculate similarity among chunks
- Chunks with highest similarity are grouped into bins
  - To increase cache locality
- Assign bins containing computations with similar data access patterns to threads
- The number of bins created should at least be equal to the number of threads
  - bin-to-thread assignment (1-to-1)
Similarity Calculation for Assigning Chunks to Bins

- Calculate similarity among all chunk pairs

\[
\text{Similarity}_{i,j} = \frac{\text{totalShared}_{i,j}}{\text{totalShared}_{i,j} + \text{totalDistinct}_{i,j}}
\]

- \(\text{totalShared}_{i,j}\)
  - Stores number of shared data between chunk \(i\) and chunk \(j\)
  - Obtained by performing AND operation on bit vectors of chunk \(i\) and chunk \(j\)

- \(\text{totalDistinct}_{i,j}\)
  - Stores number of distinct data between chunk \(i\) and chunk \(j\)
  - Obtained by performing XOR operation on bit vectors of chunk \(i\) and chunk \(j\)
Similarity Calculation - 1

Bit Vectors:

1\textsuperscript{st} chunk

\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
\end{array}

\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
\end{array}

4\textsuperscript{th} chunk

\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
\end{array}

\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}

\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\end{array}

\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}

\text{AND}

\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}

Similarity Calculation:

- total\text{Shared}_{1,4}: 8
- total\text{Distinct}_{1,4}: 1
- Similarity_{1,4}: 8 / (8+1) = 0.88
Similarity Calculation - II

Bit Vectors:

Shared / Distinct Data:

<table>
<thead>
<tr>
<th></th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td></td>
<td>3/9</td>
<td>2/12</td>
<td>8/1</td>
</tr>
<tr>
<td>[3]</td>
<td>2/12</td>
<td>6/2</td>
<td></td>
<td>1/13</td>
</tr>
<tr>
<td>[4]</td>
<td>8/1</td>
<td>2/11</td>
<td>1/13</td>
<td></td>
</tr>
</tbody>
</table>

Similarity(1,2) = 3/12 = 0.25
Similarity(1,3) = 2/14 = 0.14
**Similarity(1,4) = 8/9 = 0.88**
Similarity(2,3) = 6/8 = 0.75
Similarity(2,4) = 2/13 = 0.15
Similarity(3,4) = 1/14 = 0.07
Experimental Study

- **Simics Simulation Environment**
  - 16 Cores with private L1 Cache and private L2 Cache
  - 16 Threads mapped to each core
- **Test Parameters**
  - Chunk Number: 2, 5, 10, 20, 50
- **Sparse vector-matrix multiply**
  - 5 Input Matrices with different shapes and characteristics from the University of Florida Sparse Matrix Collection(*)
- **Performance comparison with the Linux scheduler**
  - Consecutive blocks of chunks are assigned to bins
  - Bins are equally distributed to application threads
  - Data sharing is not considered by Linux scheduler

(*) [http://www.cise.ufl.edu/research/sparse/matrices](http://www.cise.ufl.edu/research/sparse/matrices)
Performance - Boddy4

Speedup:
Best: 6.6% (50 Chunks)  L2 Cache Hit Ratio: 8.6%
Worst: 3.3% (5 Chunks)  L2 Cache Hit Ratio: 4.2%
Average: 5.6%  L2 Cache Hit Ratio: 4.8%

# of rows: 17546
# of columns: 17546
# of nonzeros: 69742
Performance – Boddy5

# of rows: 18589
# of columns: 18589
# of nonzeros: 73935

Speedup:
Best: 9.9% (10 Chunks) L2 Cache Hit Ratio: 8.6%
Worst: 6.8% (50 Chunks) L2 Cache Hit Ratio: 2.2%
Average: 8.3% L2 Cache Hit Ratio: 7.0%
Performance - Boddy6

Speedup:
Best: 7.9% (20 Chunks)  
Worst: 1.4% (5 Chunks)  
Average: 5.3%

L2 Cache Hit Ratio:
Best: 2.9%  
Worst: 1.2%  
Average: 2.0%

# of rows: 19366  
# of columns: 19366  
# of nonzeros: 77057
Performance - llc1033

Speedup:
Best: 26.8% (20 Chunks)  
Worst: 19.8% (5 Chunks)  
Average: 24.5%  
L2 Cache Hit Ratio: 10.2%
L2 Cache Hit Ratio: 2.6%
L2 Cache Hit Ratio: 5.6%
Performance - Ncvxqpp9

<table>
<thead>
<tr>
<th># of Chunks per Bin</th>
<th>Execution Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8.00</td>
</tr>
<tr>
<td>5</td>
<td>6.50</td>
</tr>
<tr>
<td>10</td>
<td>6.00</td>
</tr>
<tr>
<td>20</td>
<td>5.50</td>
</tr>
<tr>
<td>50</td>
<td>5.00</td>
</tr>
</tbody>
</table>

- **Speedup:**
  - Best: 22.4% (50 Chunks)
  - Worst: 16.4% (10 Chunks)
  - Average: 18.3%

- L2 Cache Hit Ratio:
  - Best: 3.9%
  - Worst: 2.8%
  - Average: 3.7%

- # of rows: 16554
- # of columns: 16554
- # of nonzeros: 31547
Summary

• A novel locality-aware data mapping method is proposed
  ▫ Targets to assign computations with similar data access patterns to same cores
• In our algorithm:
  ▫ Computations of a given application are divided into chunks in order to provide load balancing
  ▫ A set of chunks are grouped into bins to provide data locality
• Our goal is to decrease cache contention by increasing data reuse on L2 cache
• Outperformed Linux scheduler with an average of 12.5% performance improvement
Thank You

Any Questions?
Speedup - 8K L1 Cache, 32K L2 Cache
Speedup - 32K L1 Cache, 128K L2 Cache
Inputs