Performance Analysis and Optimization of Sparse Matrix-Vector Multiplication on Intel Xeon Phi

Athena Elafrou  Georgios Goumas  Nectarios Koziris
National Technical University of Athens

iWAPT
International Workshop on Automatic Performance Tuning
Orlando, Florida
2 June 2017
Outline

1 Motivation & Background

2 Performance analysis on Intel Xeon Phi

3 Optimization auto-tuning

4 Performance evaluation

5 Summary – Future directions
Motivation & Background

Sparse Matrices

- Dominated by zeroes (> 90%)
- Application domains:
  - Scientific, discretization of PDEs
  - Graph analytics
  - Machine learning
  - Linear programming
- Compact representation
  - Non-zero values
  - Topological information
Motivation & Background

Storing a sparse matrix: the Compressed Sparse Row (CSR) format

\[
A = \begin{pmatrix}
  7.5 & 2.9 & 2.8 & 2.7 & 0 & 0 \\
  6.8 & 5.7 & 3.8 & 0 & 0 & 0 \\
  2.4 & 6.2 & 3.2 & 0 & 0 & 0 \\
  9.7 & 0 & 0 & 2.3 & 0 & 0 \\
  0 & 0 & 0 & 0 & 5.8 & 5.0 \\
  0 & 0 & 0 & 0 & 6.6 & 8.1
\end{pmatrix}
\]

- Stores the columns of the non-zero elements and “pointers” to the first non-zero element of each row
- Most commonly used format
  - Relatively compact representation ($\approx 12 \cdot NNZ$ bytes)
  - Straightforward implementation
The Sparse Matrix-Vector Multiplication (SpMV) kernel

\[ y = A \cdot x \]

- \( A \) is a sparse matrix
- \( x, y \) are dense vectors

- Ubiquitous in scientific and engineering applications
- Dominates the execution time of many iterative methods for the solution of large sparse linear systems (e.g., CG, GMRES)
Motivation & Background
SpMV using CSR

SpMV kernel

```
for (i = 0; i < N; i++)
  for (j = rowptr[i]; j < rowptr[i+1]; j++)
    y[i] += val[j] * x[colind[j]];
```

- SpMV is characterized by:
  - Extremely low flop:byte ratio (< 0.25)
    - inherently memory bound (according to the Roofline Model)
  - Irregular memory accesses to vector \( x \)
    - may cause excessive cache misses
  - Loop overheads in case of shorts rows
  - Workload imbalance in case of highly uneven row lengths
Performance analysis on Intel Xeon Phi

Which performance issues are more prominent?

- On traditional multicore systems
  - More compact formats (BCSR, SSS, CSX, CSB, ELL etc.)
  - Reordering techniques
  - Load balancing techniques

- On modern manycore systems?
Blindly applying/combining optimizations may hinder performance!

- We need to intelligently select the optimization(s) for every matrix
  - Based on its performance bottleneck(s)
Performance analysis on Intel Xeon Phi

How do we define and determine bottlenecks?

- We define four bottlenecks
  - \textit{MB}: Memory Bandwidth
  - \textit{ML}: Memory Latency
  - \textit{IMB}: Workload IMBalance
  - \textit{CMP}: CoMPutation

- We define a performance bound for every bottleneck
  - Perform a “bound and bottleneck” analysis
  - Estimate the performance that may be gained by eliminating each bottleneck

- We design heuristics that determine the bottleneck(s) of a matrix based on the estimated performance bounds
Performance analysis on Intel Xeon Phi

Per-bottleneck performance bounds (1/2)

\[ P_{MB} = \frac{2 \cdot NNZ}{M_{format, \text{min}} + M_{xy, \text{min}}} \]

where \( B_{\text{max}} \) is the maximum sustainable DRAM bandwidth of the system, \( M_{format, \text{min}} \) and \( M_{xy, \text{min}} \) is the minimum memory traffic that can be generated by the matrix stored in \textit{format} and the vectors respectively.
Performance analysis on Intel Xeon Phi

Per-bottleneck performance bounds (1/2)

\[ P_{MB} = \frac{2 \cdot NNZ}{M_{A_{\text{format, min}}} + M_{xy, \text{min}}} \frac{1}{B_{\text{max}}} \]

where \( B_{\text{max}} \) is the maximum sustainable DRAM bandwidth of the system, \( M_{A_{\text{format, min}}} \) and \( M_{xy, \text{min}} \) is the minimum memory traffic that can be generated by the matrix stored in \( \text{format} \) and the vectors respectively.

We run a modified SpMV kernel, where irregular accesses to the right-hand side vector are completely eliminated by setting the column indices of all nonzero elements to zero.
Performance analysis on Intel Xeon Phi
Per-bottleneck performance bounds (2/2)

\[ P_{IMB} = 2 \cdot \frac{NNZ}{t_{median}} \]

where \( t_{median} \) is the median execution time of all threads
IMB

\[ P_{IMB} = \frac{2 \cdot NNZ}{t_{median}} \]

where \( t_{median} \) is the median execution time of all threads

CMP

We run a modified SpMV kernel, where we completely eliminate indirect memory references, resulting in unit-stride accesses only.
Performance analysis on Intel Xeon Phi
Per-bottleneck performance bounds (2/2)

**IMB**

\[ P_{IMB} = \frac{2 \cdot NNZ}{t_{median}} \]

where \( t_{median} \) is the median execution time of all threads

**CMP**

We run a modified SpMV kernel, where we completely eliminate indirect memory references, resulting in unit-stride accesses only.

**Peak**

\[ P_{peak} = \frac{2 \cdot NNZ}{\frac{M_{A,min} + M_{xy,min}}{B_{max}}} \]

where \( M_{A,min} \) assumes we can only compress the indexing information of a sparse matrix (not the values).
Performance analysis on Intel Xeon Phi

“Bound and bottleneck” analysis
Performance analysis on Intel Xeon Phi

Heuristics for bottleneck detection

\begin{verbatim}
procedure CLASSIFY(PCS, P_M, P_ML, P_IMB, P_CMP, P_peak)
    class <- {}
    if (PCS \approx P_M \approx P_ML \approx P_IMB) then
        class <- class \cup \{MB\}
    end if
    if (\frac{P_IMB}{PCS} > T_{IMB}) then
        class <- class \cup \{IMB\}
    end if
    if (\frac{P_ML}{PCS} > T_{ML}) then
        class <- class \cup \{ML\}
    end if
    if (PCS > P_{peak} \text{ and } \frac{P_CMP}{P_{peak}} > T_{CMP1}) \text{ or }
        (P_CMP < P_{MB} \text{ and } \frac{P_CMP}{PCS} > T_{CMP2}) then
        class <- class \cup \{CMP\}
    end if
    return class
end procedure
\end{verbatim}

- We tune the hyperparameters $T_{CMP1/2}, T_{ML}$ and $T_{IMB}$ using grid search
Optimization auto-tuning

Formulation as a classification problem

- Classes represent performance bottlenecks
- A matrix is classified (multilabel classification)
  - We propose two classifiers
    - profiling-based (hand-tuned, more expensive)
    - feature-based (trained with machine learning, very cheap)
- Optimizations that target the detected performance bottlenecks are jointly applied
  - Runtime code generation
  - Focus on cheap CSR-based optimizations

<table>
<thead>
<tr>
<th>Class</th>
<th>Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>MB</td>
<td>column index compression through delta coding</td>
</tr>
<tr>
<td></td>
<td>software prefetching on vector $\times$</td>
</tr>
<tr>
<td>ML</td>
<td>matrix split or $auto$ scheduling (OpenMP)</td>
</tr>
<tr>
<td>IMB</td>
<td>inner loop unrolling $+$ vectorization</td>
</tr>
<tr>
<td>CMP</td>
<td></td>
</tr>
</tbody>
</table>
Optimization auto-tuning

Classifier A: profiling-based

- Uses the classification algorithm presented earlier
- Relies on micro-benchmarks to be run on-the-fly to estimate some of the per-class upper bounds
  ▶ hence “profiling-based”
  ▶ hence more expensive
Optimization auto-tuning

Classifier A: profiling-based

- Uses the classification algorithm presented earlier
- Relies on micro-benchmarks to be run on-the-fly to estimate some of the per-class upper bounds
  - hence “profiling-based”
  - hence more expensive

Can we do any better?
Optimization auto-tuning

Classifier B: feature-based (1/2)

- Uses real-valued structural features of the matrix
- Trained with supervised machine learning techniques (Decision Tree)
  - Has to be trained on the target hardware platform, offline
- Training data set
  - 215 matrices from the UF Sparse Matrix Collection
  - Not balanced in terms of class representation
- Labeling
  - profiling-based classifier
  - labels may not be accurate
- Machine-learning toolkit
  - scikit-learn
- Only performs feature extraction on-the-fly
  - hence “feature-based”
  - hence cheap
## Optimization auto-tuning

### Classifier B: feature-based (2/2)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Definition</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>0: exceeds or 1: fits in LLC</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>density</td>
<td>$\frac{NNZ}{N^2}$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>$nnz_{\text{min}}$</td>
<td>$\min{nnz_1, \ldots, nnz_N}$</td>
<td>$\Theta(N)$</td>
</tr>
<tr>
<td>$nnz_{\text{max}}$</td>
<td>$\max{nnz_1, \ldots, nnz_N}$</td>
<td>$\Theta(N)$</td>
</tr>
<tr>
<td>$nnz_{\text{avg}}$</td>
<td>$\frac{1}{N} \sum_{i=1}^{N} nnz_i$</td>
<td>$\Theta(N)$</td>
</tr>
<tr>
<td>$nnz_{\text{sd}}$</td>
<td>$\sqrt{\frac{1}{N} \sum_{i=1}^{N} (nnz_i - nnz_{\text{avg}})^2}$</td>
<td>$\Theta(2N)$</td>
</tr>
<tr>
<td>$bw_{\text{min}}$</td>
<td>$\min{bw_1, \ldots, bw_N}$</td>
<td>$\Theta(N)$</td>
</tr>
<tr>
<td>$bw_{\text{max}}$</td>
<td>$\max{bw_1, \ldots, bw_N}$</td>
<td>$\Theta(N)$</td>
</tr>
<tr>
<td>$bw_{\text{avg}}$</td>
<td>$\frac{1}{N} \sum_{i=1}^{N} bw_i$</td>
<td>$\Theta(N)$</td>
</tr>
<tr>
<td>$bw_{\text{sd}}$</td>
<td>$\sqrt{\frac{1}{N} \sum_{i=1}^{N} (bw_i - bw_{\text{avg}})^2}$</td>
<td>$\Theta(2N)$</td>
</tr>
<tr>
<td>$scatter_{\text{avg}}$</td>
<td>$\frac{1}{N} \sum_{i=1}^{N} scatter_i$</td>
<td>$\Theta(N)$</td>
</tr>
<tr>
<td>$scatter_{\text{sd}}$</td>
<td>$\sqrt{\frac{1}{N} \sum_{i=1}^{N} (scatter_i - scatter_{\text{avg}})^2}$</td>
<td>$\Theta(2N)$</td>
</tr>
<tr>
<td>$clustering_{\text{avg}}$</td>
<td>$\frac{1}{N} \sum_{i=1}^{N} clustering_i$</td>
<td>$\Theta(NNZ)$</td>
</tr>
<tr>
<td>$misses_{\text{avg}}$</td>
<td>$\frac{1}{N} \sum_{i=1}^{N} misses_i$</td>
<td>$\Theta(NNZ)$</td>
</tr>
</tbody>
</table>
Performance evaluation

Experimental setup

- Hardware platform
  - Intel Xeon Phi 3120P coprocessor
    - 56 cores, 224 threads
- 64-bit Linux OS
- ICC 15.0.0
- OpenMP parallel programming API
- double-precision floating-point
- 215 matrices from the UF Sparse Matrix Collection
Performance evaluation

Feature-based classifier accuracy

- Leave-One-Out cross validation
  - assuming labels generated from profiling-based classifier

- Exact Match Ratio
  - the percentage of samples for which the predicted set of classes is fully correct

- Partial Match Ratio
  - the percentage of samples for which at least one prediction is correct

<table>
<thead>
<tr>
<th>Features</th>
<th>Complexity</th>
<th>Accuracy Exact (%)</th>
<th>Accuracy Partial (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{nnz}{\text{min, max, sd}}$, $bw_{\text{avg}}$, $\text{dispersion}{\text{avg, sd}}$</td>
<td>O(N)</td>
<td>80</td>
<td>95</td>
</tr>
<tr>
<td>$\text{size, bw}{\text{avg, sd}}$, $\text{nnz}{\text{min, max, avg, sd}}$, $\text{misses}<em>{\text{avg}}$, $\text{dispersion}</em>{\text{sd}}$</td>
<td>O(\text{NNZ})</td>
<td>84</td>
<td>100</td>
</tr>
</tbody>
</table>
Performance evaluation

Raw performance

- We compare to Intel MKL’s `mkl_dcsrmv()`
- Significant speedups for matrices that belong to the ML/IMB classes
- Performance stability
  - No slowdowns!
Performance evaluation

Runtime overhead

Minimum number of solver iterations required to amortize cost

\[ N_{\text{iters}} \gg \frac{t_{\text{pre}}}{t_{\text{spmv}} - t'_{\text{spmv}}} \]

\( t_{\text{pre}} \): online preprocessing time
\( t_{\text{spmv}} \): the execution time of SpMV before optimization
\( t'_{\text{spmv}} \): the execution time of SpMV after optimization

<table>
<thead>
<tr>
<th>Optimizer</th>
<th>( N_{\text{iters, best}} )</th>
<th>( N_{\text{iters, avg}} )</th>
<th>( N_{\text{iters, worst}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>trivial-single</td>
<td>470</td>
<td>928</td>
<td>8460</td>
</tr>
<tr>
<td>trivial-combined</td>
<td>2062</td>
<td>3802</td>
<td>38400</td>
</tr>
<tr>
<td>profiling-based</td>
<td>149</td>
<td>297</td>
<td>3200</td>
</tr>
<tr>
<td>feature-based</td>
<td>26</td>
<td>62</td>
<td>601</td>
</tr>
</tbody>
</table>
Summary – Future directions

Bottleneck-oriented optimization tuning for SpMV offers

- Performance stability
  ▶ Successfully captures diversity in sparsity patterns and hardware platforms

- Low online overhead
  ▶ Using the feature-based classifier

Future directions

- Experiment on more platforms, e.g., GPUs
- Improve accuracy of feature-based classifier
- Expand the optimization pool
Thank you!

Questions – Discussion