Auto-tuning shared memory parallel Sparse BLAS operations in librsb-1.2

Michele MARTONE  (michele.martone@ipp.mpg.de)
Max Planck Institute for Plasma Physics, Garching bei München, Germany

Intro

 Sparce BLAS (Basic Linear Algebra Subroutines) [2] specifies main kernels for iterative methods:

- Sparse Multiply by Matrix: \texttt{MM}
- Sparse triangular Solve by Matrix: \texttt{SM}

Focus on \texttt{MM}: \( C \leftarrow C + \alpha \cdot op(A) \cdot B \), with

- \( \alpha \) has dimensions \( m \times k \) and is sparse (ooz nonzeros)
- \( op(A) \) can be either of \( A, A^T, A^H \) (parameter \texttt{transA})

- left hand side (LHS) \( B \) is \( k \times n \),
- right hand side (RHS) \( C \) is \( m \times n \) (\texttt{nrhs}), both dense (eventually with stride access \texttt{incB, incC, ...})

- \( \alpha \) is scalar
- either single or double precision, either real or complex

\texttt{librsb} implements the Sparse BLAS using the RSB (Recursive Sparse Blocks) data structure [3].

- Hand tuning for each operation variant is impossible.
- We propose empirical auto-tuning for \texttt{librsb-1.2}.

RSB: Recursive Sparse Blocks

- Sparse blocks in COO or CSR [1].
- ...eventually with 16-bit indices (‘HCOO’ or ‘HCSP’).
- Cache blocks suitable for thread parallelism.
- Recursive partitioning of submatrices results in Z-ordered blocks.

Instance of matrix bayer02 (ca. 14k x 14k, 64k nonzeros).

- The \texttt{black bordered} boxes are sparse blocks.
- Gray blocks have fewer nnz than average, redder have more.
- Blocks \texttt{rows (columns)} of LHS (RHS) range during \texttt{MM}.
- Larger submatrices like ‘9’/‘9’ can have fewer nonzeros than smaller ones like ‘4’/‘9’.

Merge / split based autotuning

- Optimal default blocking?
  - Irregular matrix patterns!
  - RSBs (especially \texttt{transA}, \texttt{nrhs}) change memory footprint!

Empirical auto-tuning:

- Given a Sparse BLAS operation, probe for a better performing blocking.
- Search among slightly coarser or finer ones

Experiment in MM tuning and comparison to MKL

\begin{center}
\begin{tabular}{c|c|c|c|c}
\textbf{Symmetric} & \textbf{General} & \textbf{General}\texttt{ transpose} & \textbf{General}\texttt{ untranspose} \\
\hline
\texttt{librsb} & \texttt{MKL} & \texttt{librsb} & \texttt{MKL}
\end{tabular}
\end{center}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{example.png}
\caption{Example figure}
\end{figure}

Setup

- \texttt{librsb (icc -O3 -ANXV v15) vs Intel MKL (v.11.2) CSR}.
- 2 × Intel Xeon E5-2680, 16 OpenMP threads.
- \texttt{MM} with \texttt{nrhs=12}, four BLAS numerical types.
- 27 matrices in total (as in [3]), including symmetric.

Results Summary

- Few dozen percent improvement over untuned, costing few thousand operations.
- Significantly faster than Intel MKL on symmetric and transposed operations with \texttt{nrhs=2}.
- Autotuning more effective on symmetric and unsymmetric untransposed with \texttt{nrhs=1}.
- Tuning mostly subdivided further for \texttt{nrhs=2}.

Highlight: symmetric MM vs MV performance

\texttt{RSB} Symmetric MM performance increases when \texttt{nrhs=1}, while Intel MKL CSR falls. See here for \texttt{nrhs=2} and \texttt{nrhs=4}.

Outlook

- One may improve via:
  - Reversible in-place merge and split: no need for copy while tuning.
  - Best merge/split choice not obvious: different merge and split rules.
  - Non-time efficiency criteria (e.g. use an energy measuring API when picking better performing).

References

