Intro

- Sparse BLAS (Basic Linear Algebra Subroutines) [2] specifies main kernels for iterative methods:
  - sparse Multiply by Matrix: "MM"
  - sparse triangular Solve by Matrix: "SM"
- Focus on MM: \( C = C + \alpha (A \cdot B) \) with
  - \( A \) has dimensions \( m \times k \) and is sparse (\( \omega \) nonzeros)
  - \( \alpha (A) \) can be either of \( \{ A_1, A_2, A_3 \} \) (parameter \( \text{transA} \))
  - left hand side (LHS) \( B \) is \( k \times n \), right hand side (RHS) \( C \) \( k \times m \) (\( \text{nnz(RHS)} \))
  - \( \omega \) is scalar
- either single or double precision, either real or complex
- \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 \librsb\.

**RSB: Recursive Sparse Blocks**

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

Experiment in MM tuning and comparison to MKL

**Setup**

- \librsb\ (\( \text{inc} = 0 \)) -AXIL\ v15) vs Intel MKL (v.11.2) CSR.
- \( 2 \times "\text{Intel Xeon E5-2680}" 16 OpenMP threads.
- \( \text{MM} \) with NRHS=\( 1-2 \). 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 operation with NRHS=\( \geq 2 \).
- **Auto-tuning** more effective on symmetric and unsymmetric untransposed with NRHS=1.
- **Tuning mostly subdivided further** for NRHS=2.

**Highlight: symmetric MM vs MV performance**

<table>
<thead>
<tr>
<th>Symmetric</th>
<th>MM tuning vs untuned RSB</th>
<th>avg. 1.2x</th>
<th>avg. 1.9x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsymmetric</td>
<td>MM tuning vs untuned RSB</td>
<td>avg. 2.6x</td>
<td>avg. 17.0x</td>
</tr>
<tr>
<td>Symmetric</td>
<td>untransposed vs transposed</td>
<td>avg. 1.62x</td>
<td>avg. 2.04x</td>
</tr>
<tr>
<td>Unsymmetric</td>
<td>untransposed vs transposed</td>
<td>avg. 1.0x</td>
<td>avg. 1.19x</td>
</tr>
</tbody>
</table>

**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 performance).

**References**


[3] Ratio: RSB blocks count (tuned to untuned)
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

1. Introduction, definitions

The Sparse BLAS (Basic Linear Algebra Subroutines) standard [2] specifies the programming interface to performance critical computing kernels on sparse matrices: multiply by dense matrix (‘MM’), triangular solve by dense matrix (‘SM’)

where:

- A has dimensions m x k and is sparse (m nonzeros)
  - op(A) can be either of (A, Aᵀ, Aᴴ) (parameter transA)
  - left hand side (LHS) B is k x n
  right hand side (RHS) C is m x n
  both dense (eventually with shared access to BLAS)

- n is a scalar

- either single or double precision, either real or complex

The matrix A can be general (A ≠ Aᵀ) or symmetric (A = Aᵀ), eventually Hermitian (A = Aᴴ). In the following, using the BLAS jargon we will refer to this as NRHS. If NRHS=1, MM is equivalent to sparse matrix-vector multiply ‘MV’.

SM is defined as B = copy(C)ᵀHB, being T triangular sparse.

Gearing a Sparse BLAS implementation to be efficient in all the above cases with all the different possible inputs is challenging. librsb is a library based on the RSB (Recursive Sparse Blocks) data structure [3] and implements Sparse BLAS with the aim of offering efficient multithreaded support. Our approach in seeking both simplicity and efficiency employs empirical autotuning. Namely, the user specifies a Sparse BLAS operation to be iterated many times (using the same fixed-pattern matrix). Then a tuning procedure attempts to rearrange the sparse matrix data structure to perform that operation faster. If it fails, time taken by the autotuning procedure is lost. But if it succeeds, the now faster iterations will ultimately lead to amortize the invested tuning time. This post introduces the idea of tuning with librsb for Sparse BLAS operations. Although the autotuning framework applies to SM as well, experiments presented herein pertain MM only.

2. RSB and merge / split autotuning

The RSB format represents a numerical matrix in memory by means of smaller sparse blocks, laid out in memory according to a Z-order. This ordering arises by repeated (recursive) partitioning of the matrix coordinate domain in quadrants. Each sparse block represents the given submatrix using one of the conventional COO or CSR sparse formats [1], eventually with shorter, 16-bit indices (‘HCOO’, ‘HCSR’). Figure 1 shows an RSB instance of a publicly available matrix.

Figure 1: Graphical representation of an RSB data structure instance for matrix bayer02 (ca. 14k x 14k, 64k nonzeros). The black-bordered boxes denote the sparse blocks, with the blue zigzag line following their succession in memory. Each sparse block is labeled with a number, its sparse format and contained nonzeroes. Inner colored boxes delimit the row/column ranges of contained nonzeroes. Greener boxes have fewer nonzeroes than average; redder ones have more.

The magenta (green) segments aside of each box signal the range of LHS (RHS) matrix that will be updated (read) during untransposed MM.

Notice how larger submatrices (like 9/9) do not necessarily contain more nonzeroes, and how the area containing nonzeroes can be much smaller than the blocks (like in A/4/9).

During a Sparse BLAS operation, multiple threads can operate on separate blocks in parallel. Namely, each block will be assigned to a thread for the time of the operation on it. Then, the thread will update the corresponding LHS and use both the block and the RHS data. It is well known that an opportune sizing of the data structure (i.e. cache blocking) favours efficiency. However, an optimal sparse blocks subdivision for most real problems is impossible to determine. librsb’s default is to arrange each matrix block plus its LHS and RHS (NRHS=1) occupation to be near to the (thread local) cache size

librsb 1.2 we attempt to improve this guess by means of an empirical auto tuning procedure, sketched in Algorithm 1. This procedure probes also for a better, matrix specific thread count (θ) choice.

• Call PROBE to get reference performance and if requested by the user, update θ.
  • A bit-to-bit copy of A, θ is made; then repeatedly:
    – Call PROBE on θ to get reference performance and eventually θ’,
    – Continue coarsening as long as performance improves.
    – If the new (X/θ/) pair performs better, go to the last step; otherwise continue.
    – A fresh bit-to-bit copy of A, θ’ is made; then repeatedly:
      – Call PROBE on θ’ to get reference performance and eventually θ’’,
      – Continue subdividing as long as performance improves.
      – If the new (X/θ’/) pair performs better, autotuning failed.
      – Otherwise, autotuning succeeded and (A/θ) is substituted with (X/θ’).

Algorithm 1: Sketch of AUTOTUNE procedure. It tunes an RSB matrix instance A for Sparse BLAS operation (M x (MM SM)). Uses θ threads specification and operates from θ.

• If θ is specified, that will be used to compute execution time of operation (t) on instance A andoperands from ω.
• If θ is not specified, it will be determined by repeating the above on a reasonably restricted range.

A minimum of 10 executions is required.

Algorithm 2: Sketch of PROBE procedure, determining reference performance of a specified Sparse BLAS operation (M x A and operands from θ). Thread count specification θ may request threads probing as well.

With Algorithm 1 it shall be possible to remedy to overly fine or coarse partitionings. Subdividing or coarsening is thread parallel and one full step takes the time of a few memory copies of the interested area. Coarsening converts and combines quadrants of possibly different formats; subdividing requires repeated searches to split a sparse matrix substructure.

Figure 2 shows the results of tuning for MV and MM with NRHS=3, leading to two different tuned partitionings.

3. Sparse BLAS autotuning extension

We propose a minimal extension that introduces the notion of autotuning in Sparse BLAS portably. Relying on the USP (unstructured sparse set property) mechanism, a user may request autotuning to be performed on the next Sparse BLAS operation. The example in Figure 3 illustrates its use.

• call USP(A, blas

4. Experimental results

In [3], we compared performance of librsb to Intel MKL CSRS for MV (MKL-CSR, equivalent to MM with NRHS=1), including symmetric and transposed variants, on a sample of 27 matrices. Here we extend that experiment to all of the four BLAS numerical types and consider MM (NRHS=2); as well. We measure effectiveness of the AUTOTUNE procedure (how much RSB improvements) and compare to the results of MKL CSR (using MKL-CSR, zero-based indexing).

- RSB autotuning is more effective on symmetric and unsymmetric transposed with NRHS=1.
- RSB has been slower than MKL in 1% of the symmetric cases, 1/3 of the general untransposed cases.
- Tuning mostly subdivided further for NRHS=2.

5. Outlook

We have shown that efficiency and ease of use are not mutually exclusive, and they can be achieved while respecting a standard specification. At the same time, we believe that our techniques can be further improved by:

- Reversible in-place merge and split. The current coarsening/subdividing procedures are not reversible. Consequently in order to preserve the original input matrix structure an extra copy is being used.
- Different merge and split policies. The best merge/split choice is not obvious. A combination of the following strategies may prove effective: a) choosing the largest/smallest ones first; b) using a large block splitting strategy; c) reducing the indexing usage.
- Non-time efficiency criteria. The PROBE procedure might be changed to use e.g. an energy based metric.

References

