RICSR: A Modified CSR Format for Storing Sparse Matrices

Roman Kuprii\textsuperscript{1,2}, Boris Krasnopolsky\textsuperscript{1} and Konstantin Zhukov\textsuperscript{2}

roman.kupry@gmail.com

\textsuperscript{1}Institute of Mechanics, Lomonosov Moscow State University,
\textsuperscript{2}Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University

Supported by Russian Science Foundation Grant № 18-71-10075
Motivation

- Solving sparse systems of linear algebraic equations (SLAEs) is among the common tasks when modeling mathematical physics problems.

- Solving SLAEs occupies a significant part of all calculations.

- Iterative methods are often used to solve SLAEs.

Introduction

• A significant part of the time is spent on the execution of the operation of multiplying a sparse matrix by a vector (SpMV)

• SpMV is characterized by low computational intensity

• The efficiency of the algorithm depends on the memory bandwidth of the compute system

• One of the options for improving the efficiency of calculations is to reduce data traffic

\[
\begin{align*}
\mathbf{r}_0 &:= \mathbf{b} - \mathbf{A}\mathbf{x}_0 \\
\mathbf{p}_0 &:= \mathbf{r}_0 \\
k &:= 0 \\
\text{repeat} \\
\alpha_k &:= \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A}\mathbf{p}_k} \\
\mathbf{x}_{k+1} &:= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\
\mathbf{r}_{k+1} &:= \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k \\
\text{if } \mathbf{r}_{k+1} \text{ is sufficiently small, then exit loop} \\
\beta_k &:= \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k} \\
\mathbf{p}_{k+1} &:= \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k \\
k &:= k + 1 \\
\text{end repeat} \\
\text{The result is } \mathbf{x}_{k+1}
\end{align*}
\]
Sparse matrix storage formats

• A sparse matrix is a matrix with only a few nonzero elements in each row

• To store sparse matrices, special formats are developed, where information about nonzero elements is stored in a special way

• Two widely used basic formats: **CSR** and **ELL**

• CSR format is simple, universal, but in many cases not optimal

• Lots of advanced modifications (e.g. CSR5, ESB, SELL-C-\(\sigma\) and many other)
  • difficult to implement
  • change the original matrix
  • take a long time to convert
Sparse matrix storage formats

• In many libraries of numerical methods, the CSR (Compressed Sparse Row) format is used as the main format

• Libraries: hypre, PETSc, AMGCL, Intel MKL (Math Kernel Library), etc

• Simple lightweight modification of the CSR format: RICSR (Row Incremental CSR)
  • aims to reduce the amount of data to store column numbers
  • easy to implement
  • does not require changes to the original matrix
  • does not take much time to convert
  • can be used together with CSR format
• Three arrays are used:

• DOUBLE Val[nnz] : \{a_{0,0}, a_{0,3}, a_{1,1}, a_{1,4}, a_{1,5}, a_{2,2},
    a_{3,3}, a_{4,0}, a_{4,2}, a_{4,4}, a_{5,2}, a_{5,5}\}

• INT Row[n+1] : \{0, 2, 5, 6, 7, 10, 12\} —
  information about the number of nonzero elements in rows

• INT Col[nnz] : \{0, 3 | 1, 4, 5 | 2 | 3 | 0, 2, 4 | 2, 5\}
  — column numbers of nonzero elements

• The sizeof\{INT\} in the Col and Row – determined
  by the matrix size and the number of nonzero elements
RICSR format

- The Col[nnz] array is split into two arrays: Col_0[n] and Col_i[nnz-n]

- DOUBLE Val[nnz] : \{a_{0,0}, a_{0,3}, a_{1,1}, a_{1,4}, a_{1,5}, a_{2,2}, a_{3,3}, a_{4,0}, a_{4,2}, a_{4,4}, a_{5,2}, a_{5,5}\}

- INT Row[n+1] : \{0, 2, 5, 6, 7, 10, 12\}

- INT Col_0[n] : \{0, 1, 2, 3, 0, 2\} - column numbers of first nonzero elements in rows

- INT Col_i[nnz-n] : \{3 | 3, 4 | | | 2, 4 | 3\} — offsets from the first element of the string
RICSR format

- Reducing memory consumption is possible because:
  - The size of integer data type used in \texttt{Col} array in CSR is determined by the matrix size.
  - The size of integer data type used in \texttt{Col}_i array in RICSR is determined by the maximum of the offsets between the first and last element in the row.

- Applicability criteria
  - 4 bytes for storing column numbers in \texttt{Col} array
  - 1 or 2 byte for storing offsets on \texttt{Col}_i array

- Since the SpMV operation is limited by memory bandwidth, reducing the memory consumption for the array \texttt{Col} gives a gain despite the additional arithmetic operation.
SpMV implementation

• The key feature of the proposed format is its simplicity and compatibility with the original CSR

• The algorithm for multiplying a sparse matrix by a vector does not undergo significant changes:

SpMV operation for CSR format:

```c
for (i = 0; i < n; i++) {
    y[i] = 0;
    for (j = Row[i]; j < Row[i+1]; j++)
        y[i] += x[Col[j]] * Val[j];
}
```

SpMV operation for RICSR format:

```c
for (i = 0; i < n; i++) {
    y[i] = 0;
    for (j = Row[i]; j < Row[i+1]; j++)
        y[i] += x[Col[j]] * Val[j];
}
```

SpMV implementation

- The key feature of the proposed format is its simplicity and compatibility with the original CSR
- The algorithm for multiplying a sparse matrix by a vector does not undergo significant changes:
Theoretical estimates

- The theoretical performance gain estimates for the matrix-vector multiplication are proposed based on the amount of memory traffic.

<table>
<thead>
<tr>
<th>Col_i array bitness</th>
<th>C</th>
<th>P_{32}</th>
<th>P_{64}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (int8)</td>
<td>15</td>
<td>1.28</td>
<td>1.16</td>
</tr>
<tr>
<td>2 (int16)</td>
<td>15</td>
<td>1.18</td>
<td>1.1</td>
</tr>
<tr>
<td>4 (int32)</td>
<td>15</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Data reduction when performing an SpMV operation with single (P_{32}) and double (P_{64}) precision floating point data:

- \( C = \text{nonzeros} / \text{nrows} \)
- \( P_{32} \) – CSR to RICSR memory consumption, single precision floating point data
- \( P_{64} \) – CSR to RICSR memory consumption, double precision floating point data

\[
P_{64} = \frac{10 \cdot C + 6}{9 \cdot C + 7} \quad \quad P_{32} = \frac{6 \cdot C + 4}{5 \cdot C + 5}
\]
Implementation in the XAMG library

- XAMG library – designed to solve large sparse SLAEs, including those with many right-hand sides

- It contains a set of numerical methods including the algebraic multigrid method, Krylov subspace methods and other.

- The library provides hierarchical three-level parallelization with a hybrid MPI+POSIX shared memory parallel programming model

- The library contains several specific optimizations like vectorization, data alignment, and other

- https://gitlab.com/xamg/xamg

Testing methodology

• A subset of matrices from the SuiteSparse Matrix Collection ranging from 500K to 2M rows was used

• Two computing systems:
  - Desktop with 6-core Intel Core i7-8700 and 2-channel DDR4, 2667 MHz
  - Cluster node with 14-core Intel Haswell-EP E5-2697v3 and 6-channel DDR4, 2400 MHz

• Testing scenario:
  - SpMV: XAMG vs Intel MKL
  - SpMV: XAMG, CSR vs RICSR
  - Linear Solvers: XAMG, CSR vs RICSR
Comparison with MKL

SpMV: XAMG CSR vs MKL CSR

Most of the cases demonstrate comparable results within the range of +/- 5%
SpMV: single and double precision

Average acceleration for matrices that meet the applicability criteria: 17% and 28% for double and single precision calculations, respectively.
PBiCGStab + Jacobi

BiCGStab solver with Jacobi preconditioner: XAMG CSR vs XAMG RICSR

Average acceleration for matrices that meet the applicability criteria: 15% and 25% for double and single precision calculations
PBiCGStab + Multigrid

BiCGStab solver with Multigrid preconditioner: XAMG CSR vs XAMG RICSR

Average acceleration for matrices that meet the applicability criteria: 14% and 24% for double and single precision calculations
Conclusions

- A lightweight modification of RICSR is proposed, aimed at reducing the amount of data for storing the matrix

- Theoretical estimates of the effectiveness of SpMV with the RICSR format are proposed

- A simple criterion for the applicability of the RICSR format is formulated based on the maximum distance between the extreme nonzero elements in each row of the matrix

- Proposed format is implemented in XAMG library and thoroughly tested

- For matrices that meet the applicability criteria, the RICSR format provides a speedup of 15% to 25% for both SpMV operation and linear solvers; for the rest provides performance comparable to CSR
Future plans

• To increase the scope of applicability of the RICSR format, it is expected to use graph algorithms for reducing matrix bandwidth

• Support for the use of graphics accelerators when using the proposed modification

• Improving the presented modification by using increments between successive elements in each line

• Cache-blocking optimizations