GENERALISED VECTORISATION FOR SPARSE MATRIX–VECTOR MULTIPLICATION

A. N. YZELMAN

This work explores the various ways in which a sparse matrix–vector (SpMV) multiplication may be vectorised. It arrives at a generalised data structure for sparse matrices that supports efficient vectorisation. This novel data structure generalises three earlier well-known data structures for sparse computations: the Blocked CRS format, the (sliced) ELLPACK format, and formats relying on segmented scans. All three formats were first explored in the 1990s, with Blocked CRS being used for CPU-based calculations, while ELLPACK and formats using segmented scans have seen renewed interest within GPU computing.

The new data structure generalises all three formats, and is relevant for sparse computations on modern architectures, since most new hardware supports vectorisation for increasingly wide vector registers. Normally, the use of vectorisation for sparse computations is limited due to bandwidth constraints. In cases where computations are limited by memory latencies instead of memory bandwidth, however, vectorisation can still help performance. Such an effort is made possible by a pair of vector instructions newly introduced with the Intel Xeon Phi instruction set: the gather and the scatter instructions.

The resulting strategy is consistent with the high-level requirements of sparse matrix computations on modern CPU-based hardware [YR14], and still allows for additional optimisation such as segmented scans and bitmasking.

Parallelisation. To illustrate the new approach, the vectorised SpMV is used within the one-dimensional method of Yzelman and Roose [YR14]. This parallel scheme uses a load-balanced row distribution so that the matrix rows of A are split in p contiguous parts, where p is the number of available hardware threads, such that each part contains roughly the same amount of nonzeroes. The output vector is divided in p parts as well, corresponding to the distribution of A. Local parts of A and y are stored separately in memory so to explicitly control data locality, while the input vector is stored in a single contiguous chunk.

Thread-local versions of A are subdivided into relatively small blocks. Blocks are ordered according to the Hilbert curve, while nonzeroes within each block retain a row-major order. This ensures a cache-oblivious traversal that benefits data reuse on the high-level caches, while minimising the amount of memory required for data storage.

The parallelisation is easily prototyped using MulticoreBSP for C. By nature of the 1D row-wise distribution, this parallel implementation of the SpMV multiplication does not require any explicit communication or synchronisation.

Generalised vectorisation. To use vector instructions, l elements must be loaded from main memory into vector registers first. Streaming loads, where l contiguous values are loaded into a register from a cache-aligned memory location, are the most efficient in terms of throughput and latency hiding. Streaming loads cannot be used in case of indirect addressing, however; this while indirect addressing is the norm for unstructured sparse computations.

The new *gather* operation provides a middle way: given a vector v and an index array $i = (i_0, \ldots, i_{l-1})$, a gather on v and i loads $(v_{i_0}, \ldots, v_{i_{l-1}})$ into a vector register. The scatter operates on an input register and an index array to perform the inverse of the gather operation.

These gather/scatters are employed within the following vectorised SpMV multiplication kernel, which operates on $p \times q$ blocks of nonzeroes, with $pq = l$:

```
• for each block do
•     load the relative position $(i, j)$ of the current block within $A$;
```

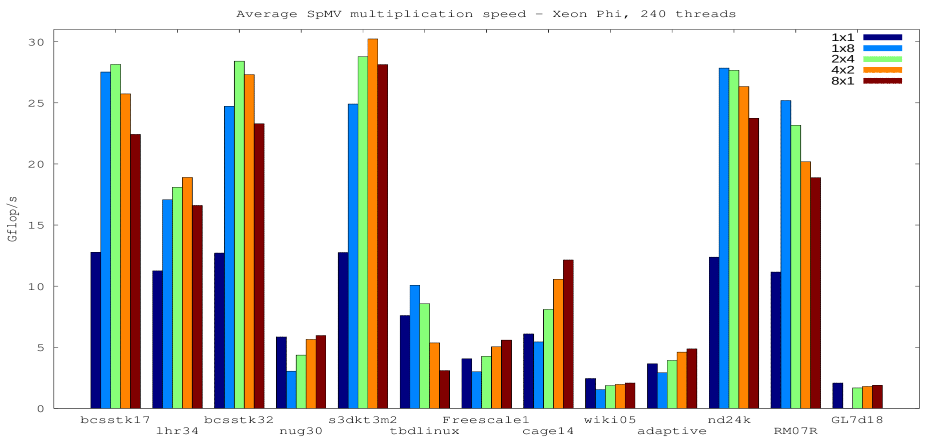

FIGURE 1. The results of the proposed 2D vectorisation of the SpMV multiplication using various matrices on the Intel Xeon Phi architecture.

- stream the l nonzeroes corresponding to this block into a vector register V, stream the row-wise offset array I of size p, and stream the column-wise offset array J of size q;
- gather the (possibly non-disjoint) elements from the output vector y using I into a vector register Y, and gather elements from x using J into a vector register X;
- do a vectorised $Y = Y + V \cdot X$, scatter Y according to I.

When $p > 1$ and $q > 1$, this multiplication scheme is, like Blocked CRS, a 2D vectorisation method. Otherwise, the resulting scheme is 1D and is equivalent to (sliced) ELLPACK (for $q = 1$) or SpMVs using segmented reductions (for $p = 1$). Unlike Blocked CRS and ELLPACK, the indices in I and J need not be contiguous thanks to the use of the gather and scatter primitives. Segmented scans do allow for non-contiguous J but require processing of an additional bitmasking array, which the proposed vectorised scheme does not require. The proposed vectorisation strategy hence generalises all three prior sparse matrix data structures.

The newly proposed method requires fill-in to ensure that the input nonzeroes in the given order fit precisely into successive $p \times q$ blocks; this was also required for Blocked CRS and ELLPACK, but was not required for methods based on segmented reductions. There, instead of filling in explicit zeroes, a bitmasking array in effect notes which nonzeroes are contained in a single block. Such bitmasking, however, was observed to be effective only on specific well-structured matrices; the proposed strategy avoids this issue entirely.

Results. The proposed 2D vectorised method for SpMV multiplication is highly effective on the Intel Xeon Phi. This relatively new architecture relies heavily on the use of many threads as well as the use of wide vector registers, supporting 240 hardware threads ($p = 240$) and storing 8 double-precision floating point values in a single register ($l = 8$). Figure 1 shows of the attained performance of the new SpMV strategy for different matrices and different block sizes, while the 1x1 category corresponds to same parallel SpMV method but without using vectorisation ($l = 1$).

The use of the proposed data structure is compared to the state-of-the-art performance on other architectures as well. Its use with other sparse matrix operations, such as the sparse matrix powers kernel, is also discussed.

REFERENCES

[YR14] A. N. Yzelman and D. Roose, High-level strategies for parallel shared-memory sparse matrix–vector multiplication, IEEE Transactions on Parallel and Distributed Systems, volume 25 (1); pp. 116–125, 2014.

FLANDERS EXASCIENCE LAB, DEPARTMENT OF COMPUTER SCIENCE, KU LEUVEN, CELESTIJNENLAAN 200A - BUS 2402, B-3001 HEVERLEE, BELGIUM., EMAIL: albert-jan.yzelman@cs.kuleuven.be