Memory access optimization for iterative tomography on many-core architectures

Wim van Aarle, Pieter Ghysels, Jan Sijbers and Wim Vanroose

***Abstract*—Iterative tomographic reconstruction methods, despite their virtues, are known to be slow compared to analytic reconstruction methods, mainly because of the computationally very intensive forward and backward projection operations. By relying on many-core architectures with large vector registers, modern high performance computing (HPC) systems can offer relief. However, to optimally benefit from such systems, the peak performance of the algorithms should not be bound by the memory bandwidth. In this work, a strategy is proposed that improves the performance of the tomographic forward projection by optimizing its memory accesses. Data locality is exploited to hide data access latency and knowledge of the cache architecture is used to optimally distribute the projection operation over many computing cores. Experiments performed on the recently introduced Intel® Xeon Phi™ architecture confirm a substantial boost in projection performance.**

***Index Terms*—Computed Tomography, High Performance Computing, vectorization, many-core computing, Xeon Phi.**

I. Introduction

ADVANCES in tomographic reconstruction techniques continue to lead to significant improvements in reconstruction quality with an ever decreasing radiation dose. Typically however, the price to pay for these improvements is a vastly increased computation time. GPU computing has already been widely applied to alleviate this downside [1], but tomographic algorithms are also ideal algorithms for implementation on general purpose *high performance computing (HPC)* systems. The performance of high-end HPC systems keeps on increasing exponentially; it is expected that by the early 2020s the first machines capable of performing one exaflop (=10^{18} floating point operations) per second will start appearing. To reach that goal, architecture manufacturers can no longer rely on ever increasing clock frequencies — power consumption has become a limiting factor — but are quickly introducing systems with an increasing number of computation cores, each with vector instructions for increasing vector lengths.

Unfortunately, memory performance is not keeping up with processor performance. Furthermore, as more cores are performing computations simultaneously, more data must be transferred from, to, and between these cores. Consequently, data intensive algorithms reach their optimal performance only if they are well adapted to the underlying system architecture.

A good measure to quantify algorithms is their associated *arithmetic intensity*. For each algorithm, this is typically a fixed number and is defined as the number of *floating point operations (flops)* executed per byte fetched from main memory.

Wim van Aarle, Pieter Ghysels and Wim Vanroose are with the Applied Mathematics and Numerical Analysis group at the University of Antwerp, Antwerp, Belgium; and with the Intel ExaScience Lab at IMEC, Leuven, Belgium. Wim van Aarle and Jan Sijbers are affiliated with iMinds-Visionlab, University of Antwerp, Antwerp, Belgium. Contact: wim.vanaarle@ua.ac.be

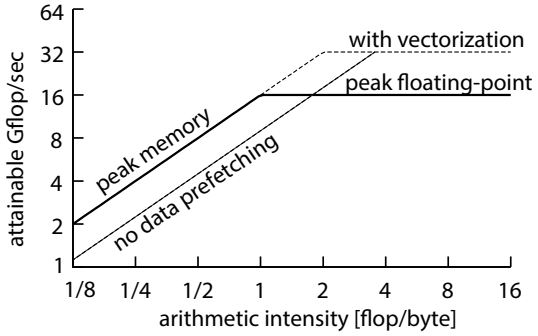


Fig. 1: Roofline model for the maximum attainable floating point performance for numerical algorithms as function of the arithmetic intensity [2]. The numerical values shown here are only an indication. In practice, they depend on the architecture.

The *roofline model*, illustrated in Fig. 1, predicts the maximum attainable performance of a computer algorithm, measured in flops per second, as a function of its arithmetic intensity [2]. The performance of algorithms with a low arithmetic intensity is memory bandwidth bound, while algorithms with a high arithmetic intensity are bound by the performance of the core processing unit.

Tomographic (back-)projection is typically memory bandwidth bound as the resulting code does not perform enough floating point operations per byte fetched from memory to hide the data access latency due to the limited memory bandwidth. The program is then often waiting for data to arrive from main memory. Each data access from main memory can take many processor cycles. With data prefetching, large chunks of data, called *cache lines*, are simultaneously brought closer to the computing processor before the data is even requested. By carefully exploiting data locality in the algorithm, data access latency can thus be hidden.

For algorithms with a sufficiently high arithmetic intensity, the attainable performance is limited by the number of flops each processing core can perform in a given time. As can be seen in Fig. 1, for such cases the peak performance can be substantially increased by vectorizing the program code, e.g. with *single instruction multiple data (simd)* instructions. For example, the recently introduced Intel® Xeon Phi™ architecture supports 512-bit vector registers, allowing one instruction to simultaneously process 16 single precision floating points.

In this paper, high performance computing optimizations are applied to the tomographic forward projection operation. Section II suggests an approach that exploits data locality to increase the benefit of code vectorization. In section III, an approach is suggested to distribute the algorithm on many-core systems in such a way that the bandwidth usage is minimal. Ultimately, Section IV concludes this work.

II. Vectorization of the forward projection

Let $\boldsymbol{v} = (v_j) \in \mathbb{R}^n$ denote a discretized square image of an object, stored in row-major form, with n, the number of pixels. In a 2D parallel beam geometry, projections of $\boldsymbol{v}$ are measured along the lines $x\cos\theta + y\sin\theta = t$, where $\theta \in [-45°, 135°)$ represents the angle between the line and the y-axis and t represents the displacement, or *detector offset*, of the line.

Let m denote the total number of measured detector values for all angles and let $\boldsymbol{p} = (p_i) \in \mathbb{R}^m$ denote the measured projection data. The forward projection can then be modelled as a linear operator $\boldsymbol{W} : \mathbb{R}^n \to \mathbb{R}^m$, that maps the volume $\boldsymbol{v}$ to the projection data $\boldsymbol{p}$, i.e. $\boldsymbol{p} := \boldsymbol{W}\boldsymbol{v}$. In this projection equation, $\boldsymbol{W}$ is an $m \times n$ matrix where w_{ij} represents the contribution of image pixel v_j to detector value p_i.

The projection weights, w_{ij}, can be modelled in various ways. In [3], an overview is given of different methods. It concludes that Joseph's linear interpolation kernel [4] has a sufficiently high accuracy and that it is well-suited for use in high performance computing. It will therefore be used in the remainder of this paper. In Joseph's method, a distinction has to be made between two types of ray. Define *vertical rays* as those for which $\theta \in [-45°, 45°)$ and define *horizontal rays* as those for which $\theta \in [45°, 135°)$.

A. Ray-driven projection

The ray-driven approach is a commonly used forward projection method in which each ray is cast through the volume, thereby summing the contributions of each pixel as the ray passes through. For a vertical ray i, at each row two pixels are hit, i.e. have a non-zero contribution to the ray. Let j denote the index of the left-most pixel. The weights w_{ij} and $w_{i,j+1}$ can then be computed and used to update p_i with the projection of pixels v_j and v_{j+1}. The order of the loops is thus: (1) direction θ; (2) detector offset t; and (3) volume row (for vertical rays) or column (for horizontal rays).

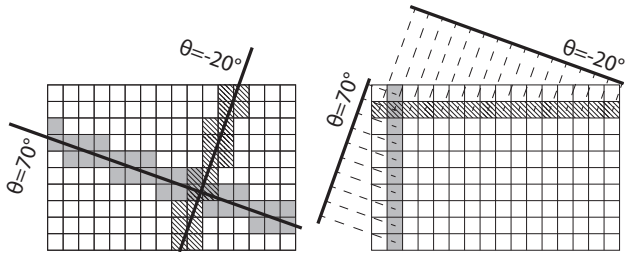
Listing 1: ray-driven projection

```
foreach ray i in [0,m):
    if direction θ of ray i is vertical:
        hitrows ← list of the rows that are hit
        foreach row in hitrows:
            j ← index of left hit pixel
            w_ij ← weight according to Joseph's model
            p_i ← p_i + w_ij v_j + (cos θ − w_ij) v_{j+1}
    else:
        analogue
```

Note that, in Listing 1, only rows are considered that are hit inside the volume window. That way, there are no conditional statements in the inner loop, which would otherwise have prevented its automatic vectorization by modern compilers.

In an optimized C++ implementation, for each byte of data accessed, only about 2 flops are performed. For optimal vectorization performance, it is therefore crucial that the required data reaches the processor as soon as possible, which can be achieved by making good use of data locality. This is accomplished if subsequent data accesses are part of the same cache line, i.e. if they are on the same row. That way, as large chunks of data are prefetched into cache, it is often already available upon request.

In Fig. 4a, the data accesses for a single ray are visualized. It is clear that horizontal rays (e.g. $\theta = 70°$) much more often require data on the same cache line than vertical rays (e.g. $\theta = -20°$). It can therefore be expected that the performance of the projection differs depending on the direction.



(a) ray-driven (b) rayrow-driven

Fig. 4: Visualization of volume data accesses. (a) For a ray-driven approach on row-major data, horizontal rays result in a data access pattern (grey pixels) much more accommodated to data prefetching than vertical rays (striped pixels). (b) For a rayrow-driven approach, vertical rays result in an optimal pattern, whereas horizontal rays result in a worst-case scenario.

B. Rayrow-driven projection

To improve the data access pattern for vertical rays, a reordering of the loops is suggested. In the *rayrow-driven* approach, the loop order becomes: (1) direction θ; (2) row (for vertical direction) or column (for horizontal direction); and (3) detector offset t. A row (vertical direction) or column (horizontal direction) is thus entirely projected in the direction θ before the next row or column is considered.

Listing 2: rayrow-driven projection

```
foreach vertical direction θ:
    foreach row:
        hitrays ← list of the rays that hit the row
        foreach ray i in hitrays:
            j ← index of hit pixel
            w_ij ← weight according to Joseph's model
            p_i ← p_i + w_ij v_j
foreach horizontal direction θ:
    analogue
```

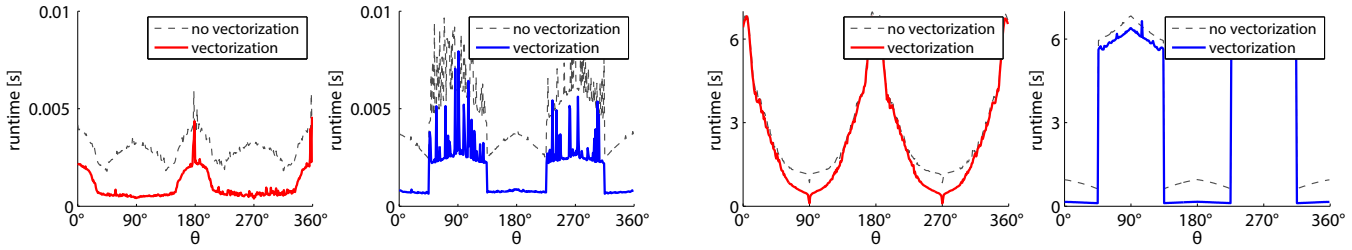
Listing 2 can be implemented such that, for each byte of data accessed only 1 flop is performed.

In Fig. 4b, the data accesses for a single row/column are visualized. For horizontal directions, the data accesses are in fact always optimal and high performance can thus be expected. Vertical directions, however, result in the worst-case scenario in which there is no data locality whatsoever.

C. Hybrid approach

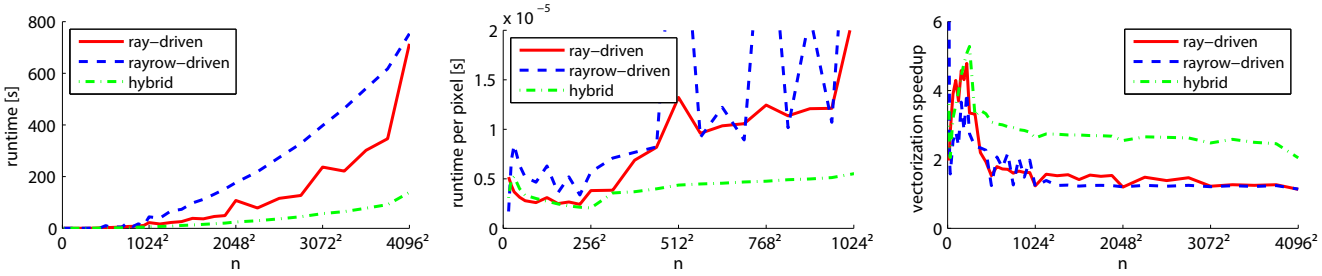
The ray-driven approach shines for horizontal rays, but is far from optimal for vertical rays. For the rayrow-driven approach, exactly the opposite is true. On their own, neither can fully utilize vectorization capabilities of modern architectures.

Fortunately, these two methods are complementary and a *hybrid approach* can easily be constructed by applying the ray-driven approach for horizontal rays, and the rayrow-driven approach for the vertical rays. By selecting the best of both worlds, a great performance improvement can be expected.



(a) ray-driven, $n = 256^2$ (b) rayrow-driven, $n = 256^2$ (c) ray-driven, $n = 4096^2$ (d) rayrow-driven, $n = 4096^2$

Fig. 2: The effect of vectorization for the ray- and rayrow-driven approaches, as a function of the projection angle. (a,b) For volumes that fit into the L2 cache, data accesses are fast enough to benefit from vectorization. (c,d) For larger volumes, vectorization is only useful if data locality can be exploited.



(a) runtime (b) runtime per pixel (c) vectorization speedup

Fig. 3: Experimental results of various projection implementations as a function of the volume size.

D. Experiments

To investigate the performance of the three previously mentioned projection approaches, a series of experiments was performed on a single core of an Intel® Xeon Phi™ "Knight's Corner (KNC)" co-processor. A KNC supports 60 cores, each of which can simultaneously run 4 threads. Combined with 512-bit vector instructions, the peak double precision performance is 1.2 TFlops/s. Each core has 512kb of L2 cache and a stream benchmark measures a bandwidth of 150GB/s. As KNC is an x86-architecture, C++ code can be easily compiled for it. All experiments were performed 5 times of which the median values are presented here.

In Fig. 2, the projection runtime for a single direction is plotted as a function of the angle θ, with vectorization enabled and disabled. Ray- and rayrow-driven approaches are compared for a small volume ($n = 256^2$), and a large volume ($n = 4096^2$). As predicted, both methods are complementary, with the ray-driven method performing well when the rayrow-driven method is slow, and vice versa. For small volumes, vectorization always results in a substantial speedup as the entire volume then fits into the L2 cache and there is little data latency. For larger volumes, a speedup can only be achieved for directions with good data locality.

In Fig. 3a, the projection time of the different methods is plotted as a function of the volume size n. In total, 240 projection directions were used. The hybrid approach clearly offers a substantial performance increase. Fig. 3b shows the projection time divided by the number of pixels in the volume size. It shows that the advantage of the hybrid method is larger for volumes larger than $n = 256^2$, which roughly coincides with the maximal volume size that fits in the L2 cache of a single Xeon Phi computing core. Fig. 3c clearly shows that the hybrid method benefits the most from vectorization.

III. Forward Projection on a Many-Core Architecture

As all rays can be handled independently, forward projection lends itself perfectly to parallel computation on modern many-core architectures. This section investigates the optimal way to distribute the workload over the different cores.

A. Direction-based parallelism

A simple approach to improve performance on many-core systems, is to distribute the outer loop of the algorithms presented in Section II. Each core then computes projection data for one projection direction.

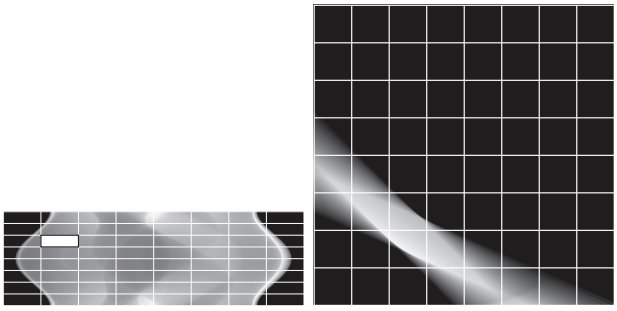
It should be noted that in such an approach, each core needs to access the entire volume data, leading to a vast increase of required memory bandwidth. As such, the conclusions drawn in section II can not be generalized to many-core systems and the performance is not likely to scale well.

B. Patch-based parallelism

In Section II and Fig. 2, it was demonstrated that for volumes that fit entirely into the L2 cache of a core, vectorization is always beneficial. Once the entire volume is loaded into the cache, all data accesses are very fast and do not contribute to the used memory bandwidth. With a *patch-based* projection strategy, this observation is used to improve projection performance, even for very large volumes.

With the patch-based strategy, the projection operation is split into many smaller sub-projections. The projection data is subdivided into a series of patches (Fig. 5a), which can be easily distributed over multiple cores. Furthermore, also the volume data is subdivided into patches, each with a size small enough to fit into the L2 cache (Fig. 5b). That way, the

projection of each small patch can be done very efficiently and substantial performance increases can be expected for large volumes. Also, as the use of memory bandwidth is limited, performance is likely to scale well over multiple cores.



(a) projection data $\boldsymbol{p}$ (b) volume data $\boldsymbol{v}$

Fig. 5: Projection and volume data split into patches. Only a few volume patches contribute to a certain projection patch.

From Fig. 5, it is clear that only a few volume patches actually contribute to a certain projection patch. Therefore, only those patches should be handled, which requires some extra logic at the start of each sub-projection. Also, to increase performance, the data should be stored in *patch-major* form instead of row- or column-major form.

C. Experiments

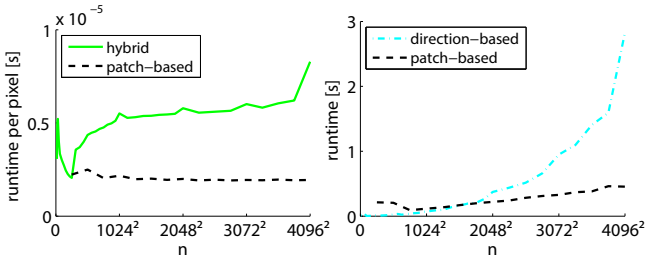
As in section II, a Xeon Phi co-processor is used to demonstrate the patch-based approach. The patch size for the projection data is chosen at 10×256 and that for the volume patches 256×256. The hybrid projection approach is used to project each patch.

Firstly, the hybrid approach is compared with the patch-based projection approach on a single core. Fig. 6a shows the runtime per pixel as a function of the volume size. It should be noted that the hybrid approach has a clear performance decrease as volumes become larger than $n = 256^2$. Many memory accesses are then required as the volume no longer fits into the L2 cache. With a patch-based approach, these memory accesses are limited in number and the runtime per pixel remains constant as the volume size increases.

Secondly, the same experiment is repeated using all 60 cores of the Xeon Phi. The workload is distributed over different cores using the Intel Thread Building Blocks (TBB) library. From Fig. 6b, the substantial performance increase of the patch-based method over the direction-based method, is clear. This can also be seen in Table I, where projection times are listed comparing all projection approaches discussed in this work.

projector type		serial	many-core	speedup
direction-based	ray-driven	714.21s	7.50s	95.2
	rayrow-driven	753.12s	7.57s	99.4
	hybrid	138.99s	2.83s	49.1
patch-based	ray-driven	79.55s	1.41s	56.4
	rayrow-driven	112.60s	2.11s	53.4
	hybrid	32.58s	**0.45s**	72.4

TABLE I: Forward projection times of a 4096×4096 volume with the different approaches discussed in this work.



(a) runtime per pixel, 1 core (b) runtime, 60 cores

Fig. 6: Experimental results of many-core adaptations of the forward projection as a function of the volume size.

IV. Conclusions

In this article, initial efforts were presented that optimize the memory accesses of iterative tomographic methods for use on modern many-core systems.

To reach optimal performance on such systems, vectorization is crucial. It was shown that vectorization is only beneficial for algorithms with a high arithmetic intensity or for algorithms where the data access latency can be hidden by exploiting data locality. For the tomographic projection operation, this can be accomplished by combining a common ray-driven approach, which has good data locality for some directions but bad data locality for others, with a rayrow-driven approach, where the data locality is perfectly complementary to the one of the ray-driven approach.

It was also shown that even with data locality, vectorization can not be achieved if the memory bandwidth is saturated. Therefore, a strategy was proposed to distribute the workload of a forward projection over multiple cores with a limited amount of data transfer. This patch-based method subdivides the projection problem into many sub-problems that are small enough to be stored in the L2 cache of a core.

Experiments performed on an Intel Xeon Phi co-processor, confirm that with the proposed strategies, the tomographic projection operation much more utilizes the capabilities of the architecture, resulting in a substantial performance increase.

Future work will focus on the back-projection, for which the same principles hold, and on algebraic reconstruction techniques.

Acknowledgment

This work is funded by Intel and by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT).

References

[1] W. J. Palenstijn, K. J. Batenburg, and J. Sijbers, "Performance improvements for iterative electron tomography reconstruction using graphics processing units (GPUs)," *Journal of structural biology*, vol. 176, pp. 250–253, 2011.

[2] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, Apr. 2009. [Online]. Available: http://doi.acm.org/10.1145/1498765.1498785

[3] F. Xu and K. Mueller, "A comparative study of popular interpolation and integration methods for use in computed tomography," in *Proceedings of the 2006 IEEE International Symposium on Biomedical Imaging: From Nano to Macro, 2006*. IEEE, 2006, pp. 1252–1255.

[4] P. M. Joseph, "An improved algorithm for reprojecting rays through pixel images," *IEEE Trans on Medical Imaging*, vol. 192, pp. 192–196, 1982.