Data Partitioning Strategies for Stencil Computations on NUMA Systems

Frank Feinbube, Max Plauth, Marius Knaust, and Andreas Polze

Operating Systems and Middleware Group
Hasso Plattner Institute for Software Systems Engineering
University of Potsdam, Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de

Abstract. Many scientific problems rely on the efficient execution of stencil computations, which are usually memory-bound. In this paper, stencils on two-dimensional data are executed on NUMA architectures. Each node of a NUMA system processes a distinct partition of the input data independent from other nodes. However, processors may need access to the memory of other nodes at the edges of the partitions. This paper demonstrates two techniques based on machine learning for identifying partitioning strategies that reduce the occurrence of remote memory access. One approach is generally applicable and is based on an *uninformed search*. The second approach caps the search space by employing *geometric decomposition*. The partitioning strategies obtained with these techniques are analyzed theoretically. Finally, an evaluation on a real NUMA machine is conducted, which demonstrates that the expected reduction of the remote memory accesses can be achieved.

Keywords: NUMA, stencil computation, data partitioning

1 Introduction

Stencils on two-dimensional data are a major field of research. [2, 6, 16, 19] Several scientific problems are solved with the help of stencils, ranging from image processing to fluid simulations. For instance, stencils are used to solve partial differential equations (PDEs) numerically [15] and linear equations with the *Jacobi method* [8]. Stencil computations iteratively update each cell of an input data matrix, using only a neighborhood of cells at a time to obtain the values. In real-world applications, this often leads to high computational intensities, which is why stencils are usually executed in a parallel fashion. Causing a high load on the memory channel, stencil computations are usually memory-bound.

Comprised of multiple processors and dedicated memory units, modern non-uniform memory access (NUMA) architectures facilitate massively data-parallel computations. With this set-up, processors can access their local memory fast and independently from other processors. Remote physical memory can still be accessed via inter-chip interconnects. However, *remote* memory access bears higher latencies and reduced bandwidth.

When executed on NUMA architectures, stencil computations require that the input data grid is *partitioned* such that each processor can perform a distinct portion of the computation in parallel to one another. When a processor updates cells at the border of its data partition, neighboring cells might not be located in the node's local memory. Hence, expensive access to remote data partitions is inevitable. The number of remote memory accesses is greatly influenced by the specific *shape* of the partitions, which raises the question which partitionings are most suitable for stencil computations on NUMA systems.

This work aims at finding partitionings that reduce the occurrence of remote memory access on modern NUMA systems. For this purpose, a technique based on evolutionary algorithms is devised to search for optimized partitionings. Building on this approach, a second technique is developed that solves the partitioning problem geometrically. Based on findings from experiments with the two techniques, the partitionings are elucidated further from a theoretical perspective. Finally, a practical evaluation on a real NUMA hardware shows that the number of remote memory accesses can indeed be decreased with the presented approaches.

The remainder of the paper is organized as follows: Section 2 presents related work. Section 3 describes two approaches how machine learning can be applied to acquire suitable data partitionings. Section 4 provides a theoretical analysis of the communication cost and compares the state of the art partitioning to the proposed partitioning for 5-point stencils. Section 5 assesses the performance on a four-node NUMA system. Finally, Section 6 summarizes this paper and highlights key results.

2 Related Work

Here, we provide a brief overview of preceding work dealing with performance optimization techniques for stencil computations and NUMA systems. At an abstract level, *vectorization* [6] and *blocking* [19] are the two general approaches for optimizing stencil computations. Nguyen el al. combined both spatial and temporal blocking to optimize stencil computations [11]. Dursun et al. concluded that the advantage of blocking is highly dependent on finding the right block size [4]. Strzodka et al. introduced an approach called CORALS, which combines temporal blocking with vectorization [18]. Shaheen et al. analyzed the effect of CORALS on NUMA systems [16]. However, the approach turns out not to be scalable on NUMA architectures.

Datta conducted research on stencil code optimizations and provided basic recommendations for NUMA systems [2]. Plauth et al. evaluated methods for introducing NUMA-awareness to the SIFT algorithm, which also employs stencil computations [13]. In a subsequent project, these findings resulted in a framework that assists C++ developers in maintaining NUMA-awareness, however the focus has shifted away from stencil computations [5].

Partition Shapes An alternative way to approach performance optimization is to focus on the communication cost by optimizing the shape of the input data partitions.

In 1986, Reed et al. [14] studied the characteristics of rectangular, triangular, and hexagonal spatial partitionings. The authors defined computation as a function of a partition's area and communication as a function of the partition's perimeter. They found that for 5-point stencils, hexagonal partitions yield the highest ratio of computation to communication. The authors evaluate the partitionings and show that good performance can only be achieved when considering the combination of stencil, partitioning, and system architecture. In 1991, Abraham et al. [1] extended this research by introducing algorithms to automatically partition the input data based on rectangular and hexagonal shapes.

In 2010, Orozco et al. [12] studied a number of different tilings for stencil computations. They provided a proof that a diamond shaped partition has the optimal ratio of computation to communication and describe how input data can be partitioned accordingly for a system with 64 processors. Their performance evaluation demonstrated the efficiency and performance of the diamond tiling in comparison to the other partitioning approaches.

In 2014, DeFlumere et al. [3] showed that – by the example of matrix multiplication algorithms – the optimal partitioning for large processor numbers is not optimal for smaller processor numbers or systems with heterogenous processors and systems with differing communication topologies.

Inspired by these findings, we studied the optimal partitionings for systems with a small number of computational nodes – such as processors or NUMA nodes. We show that while the optimal partitioning approach for large processor counts is known to be diamonds, a small number of processors or NUMA nodes require a different tiling.

3 Evolutionary Approaches

We discuss two approaches for applying machine learning to find suitable data partitioning strategies for stencil computations on NUMA systems. The first approach can be considered an *uninformed search*, which is applicable to a wide range of algorithms. The second approach takes the special characteristics of the 5-point stencil into account to produce a more efficient albeit less general solution by implementing a *geometric decomposition*.

Given *input matrices* of arbitrary type and a *latency matrix* indicating a NUMA topology, the evolutionary algorithms try to produce an optimal index range mapping matrix indices to NUMA nodes. Both approaches aim at minimizing the total occurrence of remote memory access in scenarios where the stencil computation is distributed across multiple NUMA nodes.

3.1 Uninformed Search

The *uninformed search* is suitable for various algorithms as it is provided with a function describing the *access pattern*. It is implemented as an evolutionary

algorithm that works by iteratively selecting the best individual and then creating multiple mutations of it yielding the individuals of the next generation. An individual represents a system instance, which holds a collection of nodes and a partitioning. The individuals are rated by a cost function that iterates over each cell and, depending on the partitioning, lets the according node apply the access pattern (for instance, a stencil function), and counts remote accesses. The accumulated cost of remote accesses during an individual's simulation step represents its fitness value. The cost for performing the access on the corresponding remote node is based on the *latency matrix* of the NUMA topology. In the mutation step, new individuals are created based on the parent allocation by randomly exchanging cells with different processor assignments. Parallelization is achieved using the *parallel mode* of `libstd++`, which is a parallel implementation of most of the algorithms found in the C++ Standard Library.

Optimization Strategies In order to speed up the evolutionary algorithm, some problem-specific optimizations are applied. Thereby, the resolution of the input data can be increased, which allows for the timely computation of partitionings with more nodes. In this implementation, an elitist selection is implemented by employing a Bernoulli distribution to decide whether the parent individuals should become part of the new generation or not.

With some states, performing single mutations always reduces the fitness value, while performing multiple mutations at once may result in a better partitioning. To overcome local minima, it is often necessary to mutate cells that are located close to each other. This is achieved by using a normal distribution around the first change to determine areas for the subsequent changes. In order to scale, the standard deviation of the distribution is inferred from the input data size and the number of nodes.

The algorithm sometimes moves away from the best partitioning instead of refining it, even when elitist selection is applied. While this is intended behavior to escape local minima, it might lead to a longer runtime, especially in the late phase when only refinements are required. A strategy to address this problem is to reset the population with mutations of the best known solution up to that point in time. These resets are performed based on the number of generations that have passed since the best known solution was replaced with a better one. Furthermore, the frequency of these resets is decreased when they do not lead to a successful outcome.

Results As illustrated in Figure 1, the results yielded by this approach corroborate the findings pointed out by Reed et al. [14], i.e. that diagonal partition borders are preferable since neighboring cells along diagonal borders share a common remote cell that both are accessing.

Unfortunately, the evolutionary technique reaches its limits soon with higher input data resolutions, which are necessary to find configurations with more nodes. Due to the search space explosion, partitionings with many nodes can

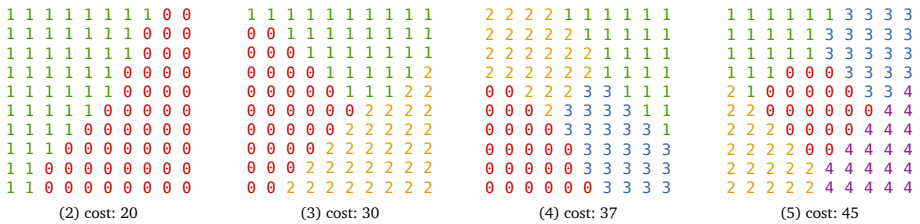


Fig. 1: Partitionings yielded using *uninformed search* for a 5-point stencil on two to five nodes with fully-connected topology and square-shaped input data.

hardly be represented without rasterization artifacts on smaller resolutions. Nevertheless, the experiments led to some interesting insights, which the *geometric decomposition* approach is taking advantage of.

3.2 Geometric Decomposition

To overcome the limitation imposed by the search space, the *geometric decomposition* works with geometric shapes instead of a raster of discrete cells. The idea is depicted in Figure 2. In order to partition a given outline shape into polygons, a configurable number of straight lines are randomly placed to subdivide the space. The resulting fragments (referred to as *atomic polygons*) are then combined to as many shapes as there are NUMA nodes in the system. For this purpose, all combinations are evaluated to determine the best candidate. More optimized partitionings can be found by performing these steps multiple times with different random lines. To further refine the partitionings obtained using this technique, the geometric approach employs a local search.

To find the best partitioning given the initial randomly generated lines, a cost function is developed that expresses remote access to other nodes in a continuous, two-dimensional space. A consequence of this approach is that the areas of the partition shapes are not necessarily the same size. Therefore, not only the remote communication cost have to be minimized but also the variation between the area sizes. To improve the efficiency of our approach, we refine the best results by applying random changes to the angle and distance of the lines following the principle of simulated annealing. [9]

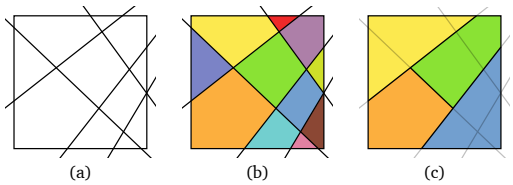


Fig. 2: Geometric approach: Random lines are generated (a) to fragment the space into *atomic polygons* (b), which are merged to match the node count (c).

To map the resulting shapes to an actual NUMA system, the shapes are converted to a discrete partitioning through rasterization.

Cost Function The cost of remote communication in the geometric approach are caused by the *edges* shared by adjacent polygons with different labels. Mapped to cell grids, polygon edges can be seen as rasterized line segments spanning Δx cells horizontally and Δy cells vertically. Without loss of generality, assume that $\Delta x > 0$, $\Delta y > 0$, and $\Delta x \geq \Delta y$. Seen from the node having the lower right partition, there are $\Delta x + \Delta y$ remote accesses to the node having the upper left partition – in general, $|\Delta x| + |\Delta y|$. However, some of the remote accesses are performed *twice* and can hence be fetched from the cache. This always amounts to $\min(|\Delta x|, |\Delta y|)$ cached remote cost. Thus, the cost for remote access is:

$$|\Delta x| + |\Delta y| - \min(|\Delta x|, |\Delta y|) = \max(|\Delta x|, |\Delta y|) \tag{1}$$

When considering polygon edges as rasterized line segments of infinite resolution, the cost of a polygon edge $e = (p_1, p_2)$ becomes:

$$\text{cost}(e) = \text{cost}((p_1, p_2)) = \max(|p_{1,x} - p_{2,x}|, |p_{1,y} - p_{2,y}|) \tag{2}$$

Abraham et al. come to similar conclusions [1]. For a single atomic polygon $A = (e_1, e_2, \ldots)$, the communication cost caused by remote accesses from *inside* the polygon is:

$$\text{cost}(A) = \sum_{e \in A} \text{cost}(e) \tag{3}$$

In principle, the remote communication cost of the *entire* geometric partitioning would be the sum of the cost of all atomic polygons. However, some edges could have been subject to merging polygons in order to match the number of available nodes. Figure 3a indicates merged edges using the same color for adjacent polygons. The cost of these merged edges $m_1, m_2, \ldots$ have to be subtracted from the total remote communication cost – once for each of both adjacent atomic polygons. Additionally, the border edges of the outline shape do not contribute any cost, as accesses are neither performed from inside the outline shape to the outside nor the other way around. For this reason, the cost of the outline shape's border edges is subtracted as well.

With atomic polygons $\underline{A} = (A_1, A_2, \ldots)$, merged edges $M = (m_1, m_2, \ldots)$, and outline edges $O = (o_1, o_2, \ldots)$, the total cost are then:

$$\text{cost}(\underline{A}, M, O) = \sum_{A \in \underline{A}} \text{cost}(A) - 2 * \sum_{m \in M} \text{cost}(m) - \sum_{o \in O} \text{cost}(o) \tag{4}$$

Since the geometric approach does *not* guarantee that all partition shapes have the same area, the presented implementation uses a *score* function to simultaneously minimize the cost function as well as the variation between the partition shapes' areas. Assuming the areas of the smallest and biggest shape are $\text{area}_{\min}(\underline{A}, M)$ and $\text{area}_{\max}(\underline{A}, M)$, respectively, the total score is defined as:

$$\text{score}(\underline{A}, M, O) = \text{cost}(\underline{A}, M, O) * \frac{\text{area}_{\max}(\underline{A}, M)}{\text{area}_{\min}(\underline{A}, M)} \tag{5}$$

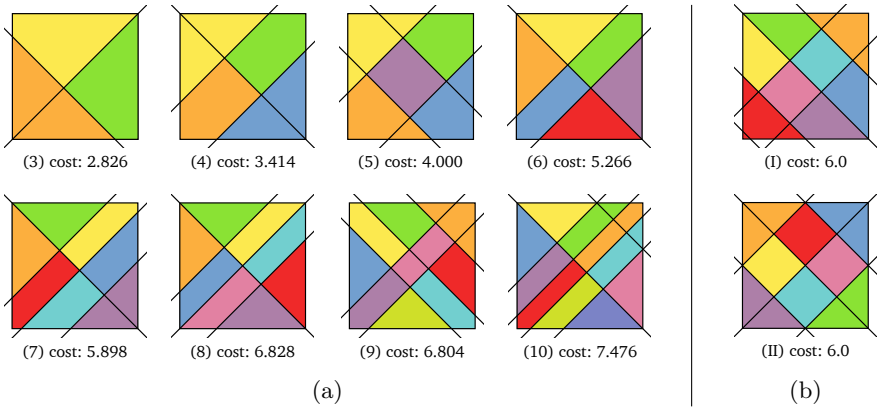


Fig. 3: The best partitionings found by the geometric approach for three to ten nodes (**a**). Two partitionings for eight nodes with diamond shapes (**b**).

Results Figure 3a shows the best partitionings found for three to ten nodes on square-shaped data when *only diagonal lines* were used. Diagonal lines were used since they have lowest remote communication cost – while the restriction improves the efficiency of the geometric approach.

Notably, the results for three to five nodes almost match the results obtained with the uninformed search. In the experiments, two patterns recurred frequently: a two-part stripe pattern and a diamond pattern.

Two-Part Diagonal Stripe Pattern An interesting finding is that all partitionings with an even number of nodes seem to follow a similar pattern. The partition shapes form diagonal stripes, each cut in half by a single diagonal line that ranges from one corner to the opposite one.

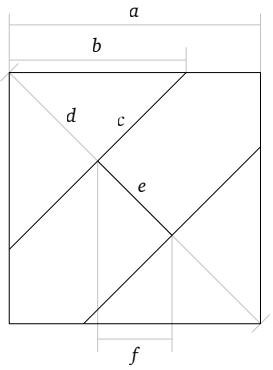
Diamond Pattern For configurations with eight nodes, two better results exist. When allowing the algorithm to generate more random lines, the patterns in Figure 3b emerge. It is interesting to observe that in these cases, triangular shapes in the corners and diamond-like structures in the middle seem to be preferable.

Hierarchical Application The geometric technique can be applied to hierarchical NUMA topologies. In such cases, each partition is further divided into sub-partitions, using the same approach. Using this technique, large NUMA systems with hierarchical topologies (such as the SGI UV300H [17]) can be handled.

4 Theoretical Analysis

In this section, we provide a theoretical analysis of the partitionings observed in Section 3. Optimal communication cost is obtained when maximizing the area of the partitions in relation to the perimeter. When rectangular partition shapes are assumed, squares are the the optimal rectangular partitioning for 5-point stencils [14]. However, partitioning a given outline with just squares is only possible for n NUMA nodes with $n = k^2$ for some $k \in \mathbb{N}^+$.

For our four-node test system, a square-based partitioning is trivial: Each shape is circumvented by four edges of length $\frac{a}{2}$, where a is the side length of the two-dimensional input matrix. Each edge amounts to the cost of a ($\frac{a}{2}$ once for both of the adjacent squares). The total cost are $\text{cost} = 4\left(2 * \frac{a}{2}\right) = 4a$.



$$
\begin{aligned}
b &= \sqrt{\frac{a^2}{2}} &&= a\sqrt{\frac{1}{2}} \\
c &= \sqrt{b^2 + b^2} &&= a \\
d &= \sqrt{a^2 + a^2} &&= a\sqrt{2} \\
e &= d - 2\frac{c}{2} &&= a\left(\sqrt{2} - 1\right) \\
f &= \sqrt{\frac{e^2}{2}} &&= e\sqrt{\frac{1}{2}}
\end{aligned}
$$

Fig. 4: Pattern yielded by the geometric approach, labeled with length ratios.

Next, we consider the non-rectangular four-node pattern with the lowest cost yielded by the geometric approach (see Figure 4). As shown in Equation 6, the communication cost amount to $3.414a$, which is less than the cost of the *rectangular partitioning* ($4a$). Notice that this calculation expects caching to be present. As only diagonal lines occur in the pattern, both projections would have the same length.

$$\text{cost} = 2 * b + 2 * (b + f) = \left(\sqrt{2} + 2\right) a = 3.414a \tag{6}$$

5 Evaluation

Here, we evaluate the approaches presented in Section 3 using a fully connected four-node NUMA system (see Table 1) with a 5-point cross-type stencil being applied to square-shaped data. This evaluation is based on the partitionings discussed in Section 4. Based on the theoretical analysis, we expect that the remote access cost of the partitioning pattern roughly amount to about 85 % (based on the ratio $\frac{3.414a}{4.0a}$) of the cost of the rectangular reference partitioning.

Table 1: Detailed specifications of the reference system.

	HPE ProLiant DL580 G9 Server
CPU	4 × Intel Xeon E7-8890 v3 (Haswell), 18C/36T, 2.5 GHz
Memory	16 × 8GB DDR4-1600 reg. ECC DIMM
Topology	4 Sockets, Fully interconnected, RMA Penalty: ca. 1.4 [7]

To verify this hypothesis, the number of remote accesses is counted while performing the stencil computation. In order to reduce noise in the measurement, the simulation of the stencil operation only performs the memory accesses without executing the actual computation.

5.1 Method of Measurement

While the stencil operation is not required to be computed for evaluation purposes, it is important to make sure that memory accesses are still performed. In particular, the compiler must not optimize any memory access away. To have full control over the implementation inline assembler is used.

To evaluate the effect of the refined partitioning, it necessary to count how many remote memory accesses are performed by the nodes. For this purpose, off-core hardware performance counters are used. To filter off-core response events, certain fields in the *model-specific registers* (MSR) have to be configured. With this mechanism at hand, it is possible to tailor a special filter suited to measuring memory accesses on remote nodes regarding data and not instructions. Linux `perf` is used to access the off-core performance counters.

In order to measure only the impact of the introduced partitioning pattern, it is necessary to eliminate all other factors on the number of remote memory accesses. For this reason, prefetchers are disabled in the evaluation, even though they would have an effect in practice. Furthermore, automatic NUMA balancing is disabled to make sure the Linux kernel does not interfere with the memory placement. Lastly, functions have been implemented that verify that thread bindings and memory allocations are performed as expected.

5.2 Results

In our measurements, a 5-point cross-type stencil is applied to a two-dimensional input matrix. The geometric partitioning and the rectangular partitioning from Section 4 are both tested and the impact of input data grid size, cell size and kernel size are investigated.

Dimension of the Input Data Grid The cell size was fixed at 10 kB, and the side length ranged from 10 to 1000 cells. The anticipated value of 85 % is reached starting at resolutions of 100×100 (see Figure 5a). This value cannot be obtained with smaller side lengths, presumably because the resolution is not high enough and introduces aliasing artifacts.

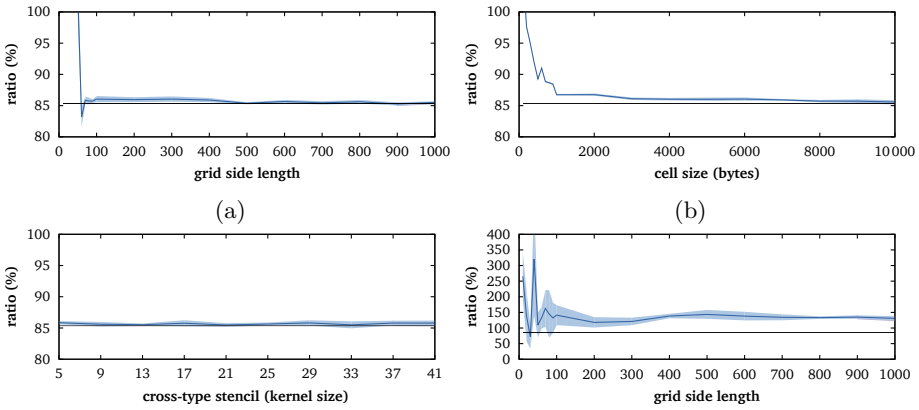


Fig. 5: For larger input grid sizes (**a**) and larger cell sizes (**b**), the theoretically computed improvement of the remote communication cost is achieved. Increasing the kernel size has no effect (**c**). However, when data is written locally, the cache coherence protocol introduces additional remote accesses (**d**).

Variable Cell Size The grid dimension is fixed to 1000×1000, and the cell size is varied from 10 to 10 000 bytes. At around 3 kB, the expected improvement regarding the number of remote memory accesses is almost reached (see Figure 5b). Additional investigations are necessary to identify why the improvements do not affect smaller cell sizes.

Kernel Size of the Cross-Type Stencil A grid of 1000×1000 cells of size 5 kB is used and the stencil size ranges from 5 to 41 to evaluate the impact of the stencil size. The result shows that the kernel size does not influence the obtained improvement concerning the number of remote memory accesses (see Figure 5c).

Additional Observations When actual computations are performed, data is written locally in addition to reading remote data. This results in increased node-interconnect utilization, likely to be caused by cache coherency traffic among nodes (see Figure 5d). To represent real world scenarios more accurately, the theoretical model needs to be extended to incorporate the remote accesses introduced by the cache coherency protocol.

Furthermore, the findings of this evaluation lead to the question which actual systems could benefit from the newly found partitioning patterns. Even with a well-suited system, the positive effects of the partitionings highly depend on the exact configuration, as Reed et al. [14] noticed as well. Partitioning patterns need to be tailored to the exact number of NUMA nodes and the caching behavior. Otherwise, applying the partitioning patterns can be counterproductive. While the *geometric decomposition* approach is more efficient, the flexibility of

the *uninformed search* approach allows the consideration of further system characteristics, such as cache line sizes, in its fitness function.

6 Conclusions

With NUMA architectures, accesses to the memory of remote nodes bear higher latencies than local accesses. Multiple solutions were developed with the objective of finding data partitionings that reduce the demand of remote memory accesses during the execution of stencil operations.

First, an uninformed search technique based on evolutionary algorithms was developed. This approach was conceived to make as little assumptions about the data, memory access patterns, and the system configuration as possible. Even though the evolutionary technique is limited to small data grid resolutions due to the large search space, multiple recurring partitioning patterns could be observed. Building on these findings, a second, geometric approach was devised. This technique is based on the temporary assumption that the input data grid is a resolutionless, two-dimensional space. In this space, partitionings are searched for by fragmenting the space into simple, polygonal shapes. A new cost function was introduced in order to translate the concept of remote memory demands to this continuous representation. The implementations of each approach are available online[1]. They are discussed in detail in [10].

A novel partitioning pattern for a four-node NUMA system is identified, analysed and evaluated – showing the projected communication cost reduction to 85%. Furthermore the impact of various scaling factors was evaluated. To provide consistent performance improvements in real world scenarios, the influence of the cache coherency protocol has to be further investigated.

Acknowledgement & Disclaimer This paper has received funding from the European Union's Horizon 2020 research and innovation programme 2014-2018 under grant agreement No. 644866. This paper reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

References

1. Abraham, S.G., Hudak, D.E.: Compile-time partitioning of iterative parallel loops to reduce cache coherency traffic. Parallel and Distributed Systems, IEEE Transactions on 2(3), 318–328 (1991)
2. Datta, K.: Auto-tuning Stencil Codes for Cache-Based Multicore Platforms. Ph.D. thesis, University of California, Berkeley (2009)
3. DeFlumere, A.: Optimal Partitioning for Parallel Matrix Computation on a Small Number of Abstract Heterogeneous Processors. Ph.D. thesis, University College Dublin (2014)

[1] `https://gitlab.com/hpi-osm/stencil-partitioning`

4. Dursun, H., Nomura, K.i., Wang, W., Kunaseth, M., Peng, L., Seymour, R., Kalia, R.K., Nakano, A., Vashishta, P.: In-Core Optimization of High-Order Stencil Computations. In: PDPTA. pp. 533–538 (2009)
5. Hagen, W., Plauth, M., Eberhardt, F., Polze, A.: PGASUS: A Framework for C++ Application Development on NUMA Architectures. In: 2016 Fourth International Symposium on Computing and Networking (CANDAR). pp. 368–374. IEEE, Hiroshima, Japan (Nov 2016)
6. Henretty, T., Veras, R., Franchetti, F., Pouchet, L.N., Ramanujam, J., Sadayappan, P.: A stencil compiler for short-vector SIMD architectures. In: Proceedings of the 27th international ACM conference on International conference on supercomputing. pp. 13–24. ACM (2013)
7. Hewlett-Packard Development Company: Red Hat Linux NUMA Support for HP ProLiant Servers. Tech. rep. (2013), [Online; accessed 1-February-2017]
8. Jacobi, C.G.J.: Über ein leichtes Verfahren die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen. Journal für die reine und angewandte Mathematik 30, 51–94 (1846)
9. Kirkpatrick, S., Vecchi, M.P., et al.: Optimization by simulated annealing. science 220(4598), 671–680 (1983)
10. Knaust, M.: Partitioning 2D Data for Stencil Computations on NUMA Systems. Master's thesis, Hasso Plattner Institute, University of Potsdam (2016)
11. Nguyen, A., Satish, N., Chhugani, J., Kim, C., Dubey, P.: 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–13. IEEE Computer Society (2010)
12. Orozco, D., Garcia, E., Gao, G.: Locality Optimization of Stencil Applications Using Data Dependency Graphs. In: Proceedings of the 23rd International Conference on Languages and Compilers for Parallel Computing. pp. 77–91. LCPC'10, Springer-Verlag, Berlin, Heidelberg (2011)
13. Plauth, M., Hagen, W., Feinbube, F., Eberhardt, F., Feinbube, L., Polze, A.: Parallel Implementation Strategies for Hierarchical Non-Uniform Memory Access Systems by Example of the Scale-Invariant Feature Transform Algorithm. In: IEEE International Parallel and Distributed Processing Symposium Workshops. pp. 1351–1359. IEEE, Chicago, Illinois, USA (May 2016)
14. Reed, D.A., Adams, L.M., Patrick, M.L.: Stencils and problem partitionings: Their influence on the performance of multiple processor systems. Computers, IEEE Transactions on 100(7), 845–858 (1987)
15. Roth, G., Roth, G., Mellor-crummey, J., Mellor-crummey, J., Kennedy, K., Kennedy, K., Brickner, R.G., Brickner, R.G.: Compiling Stencils in High Performance Fortran. In: In Supercomputing 97: Proceedings of the 1997 ACM/IEEE conference on Supercomputing. pp. 1–20. ACM Press (1997)
16. Shaheen, M., Strzodka, R.: Numa aware iterative stencil computations on many-core systems. In: Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International. pp. 461–473. IEEE (2012)
17. Silicon Graphics International Corp: SGI UV 300H for SAP HANA (2015)
18. Strzodka, R., Shaheen, M., Pajak, D., Seidel, H.P.: Cache oblivious parallelograms in iterative stencil computations. In: Proceedings of the 24th ACM International Conference on Supercomputing. pp. 49–59. ACM (2010)
19. Wellein, G., Hager, G., Zeiser, T., Wittmann, M., Fehske, H.: Efficient temporal blocking for stencil computations by multicore-aware wavefront parallelization. In: Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International. vol. 1, pp. 579–586. IEEE (2009)