NUMA Aware Iterative Stencil Computations on Many-Core Systems

Mohammed Shaheen and Robert Strzodka
Integrative Scientific Computing Group
Max Planck Institut Informatik
Saarbrücken, Germany
Email: {mshaheen, strzodka} @mpi-inf.mpg.de

***Abstract*—Temporal blocking in iterative stencil computations allows to surpass the performance of peak system bandwidth that holds for a single stencil computation. However, the effectiveness of temporal blocking depends strongly on the tiling scheme, which must account for the contradicting goals of spatio-temporal data locality, regular memory access patterns, parallelization into many independent tasks, and data-to-core affinity for NUMA-aware data distribution. Despite the prevalence of cache coherent non-uniform memory access (ccNUMA) in todays many-core systems, this latter aspect has been largely ignored in the development of temporal blocking algorithms. Building upon previous cache-aware [1] and cache-oblivious [2] schemes, this paper develops their NUMA-aware variants, explaining why the incorporation of data-to-core affinity as an equally important goal necessitates also new tiling and parallelization strategies. Results are presented on an 8 socket dual-core and a 4 socket oct-core systems and compared against an optimized naive scheme, various peak performance characteristics, and related schemes from literature.**

***Keywords*-stencil computation; temporal blocking; NUMA-aware data distribution; affinity; cache-oblivious; cache-aware; parallelism and locality**

I. Introduction

Any single stencil computation of low arithmetic intensity, e.g., a 3D Laplace kernel, is memory-bound and its performance is limited by the peak system bandwidth. However, the performance of iterative stencil computations, e.g., a 3D Jacobi solver, can surpass the peak system bandwidth by exploiting temporal data locality between iterations.

If we assemble multiple iterations of a spatial stencil computation along a time axis, we obtain a space-time domain in which each point belongs to a certain spatial location and iteration number. An effective method to increase the performance of an iterative stencil application is to tile the space-time such that the spatial and temporal locality is increased, and thus more in-cache processing occurs during the computation. However, on modern computers the tiling scheme must also regard the ccNUMA (cache-coherent non-uniform memory access wherein the memory is physically distributed but logically shared) nature of the memory system, extract many-fold parallelism for all cores in the machine, and use regular memory access patterns for best bandwidth utilization. These contradicting requirements lead to many tiling schemes with many parameters and different emphasis on the requirements. The parameter space is explored manually [3], by auto-tuning [4]–[6], by heuristics [7], [8], by general loop transformation frameworks [9]–[11], or the algorithm is cache oblivious and does not require parameters [12], [13].

So far the ccNUMA nature of today's machines has been largely ignored in tiling schemes despite its crucial importance for scalability and the fact that the related problem of minimizing communication in a distributed memory system has been already analyzed for one of the first temporal blocking schemes by Wonnacott [14]. To systematically devise an algorithm that delivers scalable high performance results, we include the NUMA aspect as an equally important goal in our list of four key requirements for efficient temporal blocking schemes on ccNUMA machines:

- spatio-temporal data locality,
- parallelization,
- regular memory access,
- data-to-core affinity.

In this paper, we build upon our previous cache-aware CATS [1] and cache-oblivious CORALS [2] schemes that perform well on uniform memory systems but exhibit unsatisfactory scalability on ccNUMA machines. Adding data-to-core affinity to these schemes is a challenge because the requirements are in conflict, e.g., parallelization conflicts with data-to-core affinity when an idle processor could process data that has been allocated by threads running on a different core. In case of CATS these conflicts can be resolved more easily than for CORALS, which requires a new tiling and parallelization strategy and becomes a significantly different scheme than the original. In summary, our main contributions are:

- A scalable cache aware scheme for iterative stencil computations, called nuCATS (Section II).
- A scalable cache oblivious scheme for iterative stencil computations, called nuCORALS (Section III).
- The schemes can operate with stencils of any order and size on multi-dimensional arrays (Section IV-F).
- The stencil coefficients may vary across the domain, i.e., the schemes support also a product with a sparse banded matrix (Section IV-E).

With nuCATS and nuCORALS we achieve excellent performance with default parameters. Weak and strong scalabil-

ity on 16- and 32-core ccNUMA machines is demonstrated for multiple problem sizes in Section IV. We compare our results against an optimized naive implementation, various benchmarks that expose the compute and bandwidth limits of the hardware across all levels of the memory hierarchy, and results from literature. We conclude that the idea of fulfilling all requirements equally leads to highly successful temporal blocking schemes (Section V).

II. NUMA-aware CATS Scheme (nuCATS)

The cache-aware CATS (cache accurate time skewing) scheme [1] divides the space-time into large tiles, much larger than the cache. However, the tiles have a carefully chosen cross-section that allows a cache efficient wavefront traversal of them. The processing within the tile, i.e., the wavefront traversal, does not change in nuCATS, however, the tiling and the scheduling of the tiles changes.

CATS assigns threads to tiles in a round robin fashion, such as to reduce synchronization and obtain automatic load balancing, because tiles at the domain boundary are smaller than inside the domain. However, such an assignment violates the data-to-core affinity requirement, because a thread may be assigned a tile that resides anywhere in the domain; nuCATS performs a domain decomposition so that each thread owns a subdomain. Then it assigns tiles to threads based on which subdomain contains most of the tile. For simplicity nuCATS enforces a particularly regular pattern of how tiles and subdomains match.

Formulas inherited from CATS deliver the recommended wavefront size of tiles based on cache parameters. From this nuCATS computes the number of tiles that could fit side by side along the dimension designated for tiling. We distinguish two cases; the first case when the number of tiles is greater than but does not divide the number of threads, then we reduce the wavefront size and thus the number of tiles is enlarged until it divides the number of threads. The second case when the number of threads is greater than but does not divide the number of tiles, then similar to the first case the wavefront size is reduced and thus number of tiles is enlarged until it equals the number of threads. However, often for the second case when the number of threads is huge, this could result in a wavefront size smaller than a heuristic value computed from the cache parameters, we stop reducing the wavefront size when it is equal to half the number of threads. The number of tiles is then doubled by cutting the dimension of the wavefront traversal in half. This reduces locality, however, is still better than cutting the unit-stride dimension which would also affect the utilization of the system bandwidth. At this stage the number of tiles is equal to or a multiple of the number of threads and each thread is assigned to one or multiple tiles that lie within its subdomain.

III. NUMA-aware CORALS Scheme (nuCORALS)

The original cache-oblivious CORALS (cache oblivious parallelograms) scheme [2] creates a regular hierarchical space-time decomposition into parallelograms, which serves both for cache oblivious data locality and parallelization. Data-to-core affinity cannot be directly incorporated into this decomposition, therefore, the new NUMA-aware CORALS scheme (nuCORALS) inherits only the single-threaded treatment of data locality from its predecessor and creates a second level of tiling with different parallelization and synchronization. Overall, there are more differences than similarities so we describe the entire scheme in the following.

A. Bidirectional Tiling

The spatial dimensions form a tensor product and each of them relates to the time dimension in the same fashion. Therefore, to explain most aspects of nuCORALS it suffices to discuss the relation between one spatial dimension and the time dimension in a 2D space-time, see Figure 1 and 2. Section III-D discusses those properties that need additional consideration in multiple dimensions.

The scheme runs in three phases:

Phase I: *NUMA-aware spatial domain decomposition and data-to-core affinity maximization*

We tile the *spatial* dimensions such that the overall number of *spatial tiles* is equal to the number of threads executing the scheme. In Figure 1 the spatial tiles are one dimensional with width b. We use affinity routines to pin each thread to one core before it allocates and initializes one spatial tile. As such, each spatial tile is allocated in the memory attached to the processor running the thread (first touch strategy) and by allowing each thread to process the spatial tile it has allocated, we ensure the data-to core affinity requirement is satisfied.

Phase II: *Parallelization*

We tile the temporal dimension according to the parameter τ (see Figure 1) into a certain number of *temporal tiles*. Section III-C discusses the selection of the parameter τ in detail. The tensor product of the spatial tiles with the temporal tiles results in layers of *space-time slices*.

To allow multiple threads to start in parallel, space-time slices are skewed to the right with a slope equal to the stencil order, resulting in parallelograms which we refer to as *thread parallelograms*. Thread parallelograms (depicted in different colors in Figure 1) at the left boundary of each spatial dimension are wrapped around to support periodic boundary conditions.

Phase III: *Cache oblivious decomposition and stencil kernel computation*

Each thread proceeds by covering its thread parallelogram by a single space-time parallelogram, which we call *root parallelogram*. Root parallelograms are skewed to the left with a slope equal to the stencil order to respect the stencil

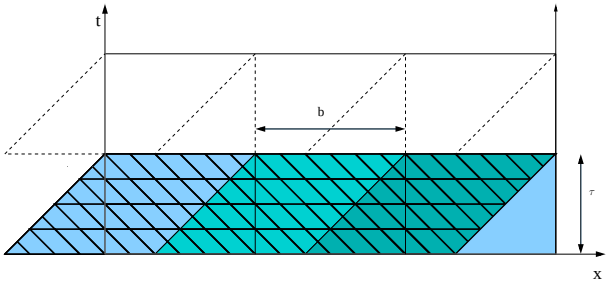

Figure (1): Bidirectional tiling. Large thread parallelograms are skewed to the right and depicted with different colors. Small base parallograms are skewed to the left. Vertical lines separate special tiles of width b. Each spatial tile is allocated and initialized by a different thread to assure data-to-core affinity. τ denotes the height of thread parallelograms, it parameterizes the trade-off between data-to-core affinity and temporal locality. Dotted parallelograms depict the next layer of thread parallelograms.

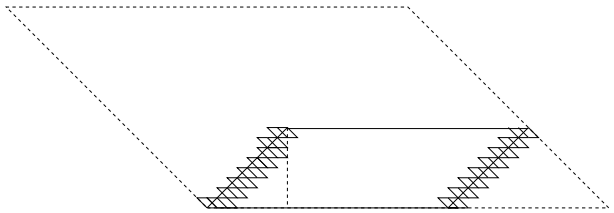

Figure (2): One thread parallelogram covered by a root parallelogram (dotted). The small base parallelograms are formed be subdivision of the root parallelogram. Threads must synchronize at base parallelograms that intersect the thread parallelogram boundary; the lower part of each intersecting base parallelogram must be computed first, before a different thread in a neighboring thread parallelogram may compute the upper part. The part to the left of the dotted vertical line has been allocated by the neighboring thread parallelogram.

dependencies. We skew thread parallelograms to the right and root parallelograms to the left and not vice versa, because the alternative would require to process thread parallelograms from right to left, which works against the prefetcher.

nuCORALS recursively subdivides the root parallelogram into *intermediate parallelograms* striving to maximize their volume-to-surface area ratio. To this end, always the longest dimension (including time) of the intermediate parallelograms is subdivided. The subdivision is stopped when all dimensions of the current intermediate parallelograms have reached a certain size, we call the resultant parallelograms which are not subdivided further *base parallelograms*. A single-threaded kernel is then applied on the data covered by the base parallelograms. Once all threads have finished executing the kernels on the data covered by their thread parallelograms, they synchronize before they proceed to the next layer of space-time slices and execute phase III repeatedly until all layers are processed.

B. Synchronization

Threads are synchronized in two places, between each pair of thread parallelograms and at the boundary of each layer of space-time slices. For the latter, one could synchronize each thread parallelogram with the two thread parallelograms beneath it. Since this synchronization does not happen very often due to the relatively small number of thread parallelograms, we use barriers in pthreads to synchronize all threads at the boundary of each layer of space time slices. We call this *global* synchronization, since all threads are involved.

Base parallelograms that intersect the boundary of any thread parallelogram (Figure 2) must be processed by multiple threads in a certain order. Therefore, synchronization is needed between these threads. We attach a structure of synchronization flags to each thread. Each flag represents the index of a base parallelogram within the root parallelogram space. We distinguish two checks, the first is the intersection with the right boundary of the thread parallelogram, and the second is the intersection with the left boundary. If a base parallelogram intersects the right boundary of a thread parallelogram, then the thread enters a spin-wait loop waiting for the flag of that base parallelogram to be set. If a base parallelogram intersects the left boundary of a thread parallelogram, then the thread processes all data that belongs to its thread parallelogram, i.e., the lower part of the base parallelogram, and then sets the corresponding flag in the synchronization structure of the adjacent thread whose right boundary intersects with this base parallelogram. We call this *local* synchronization.

C. Internal Parameters

nuCORALS has several internal parameters which are hidden from the user. Tuning these parameters can yield higher performance on some machines, however, we fix them for easier code portability of our schemes.

As is the case for most practical implementations of cache oblivious algorithms, we stop the recursive subdivision of the space-time domain once the tile is sufficiently small because going deeper in the recursion tree, until single space-time points are reached, would produce more control logic overhead than the actual computation. Furthermore, tiles with single space-time points limit the optimization opportunities for the computation inside the tiles such as innermost loop unrolling and vectorization, see [15]. We compute the dimensions of the base parallelogram in the same way as in CORALS [2].

The internal parameter τ is the height of a thread parallelogram, it represents a trade-off between temporal locality and data-to-core affinity. For stencil order $s = 1$, the ratio of data items processed by one thread but allocated by another to the overall number of items

computed by this particular thread is $\tau/2b$, where $b = spatial\_dimension\_size/no.\_of\_threads$ is the width of the thread parallograms. We can obtain more temporal locality by increasing τ at the expense of less data-to-core affinity, because larger τ results in bigger fractions of data being processed by one thread but allocated by another. The same effect appears when b becomes small, e.g., due to a high number of threads, however, we solve this problem by parallelizing across multiple dimensions, see Section III-D. With some experiments, we have found that setting $\tau = b/2$ to be half the width of the thread parallelograms results in a good trade-off between these two conflicting requirements: 75% of the overall processed data are local.

D. Multidimensional Properties

This section explains the properties of our scheme for an arbitrary dimension m of the space-time. We explain the differences to the 2D iteration space and refer for analogy to the previous 2D figures.

Domain decomposition. In an m dimensional space-time and n threads, we create n tiles by dividing all dimensions except for the unit-stride since this reduces the bandwidth utilization [4], [16]. Each dimension is subdivided into approximately $n^{1/(m-2)}$ tiles where $m-2$ results from excluding the time and the unit-stride dimensions. If $n^{1/(m-2)}$ is not an integer, we favor dimensions with a higher stride, e.g., for $m = 4$D space-time domain and $n = 4$, only two dimensions are subdivided, each dimension is subdivided into 2 tiles; for $n = 8$, the dimension with highest stride is subdivided into 4 tiles and the other is subdivided into 2 tiles.

Synchronization. Synchronization is almost the same as in the 2D case, the only difference is that local synchronization is now needed between each adjacent pair of thread parallelograms in each dimension. This results in more checks for intersection with the left or the right boundaries of the thread parallelogram in each dimension. However, these checks are cheap and hardly impact the running time of the scheme.

Internal parameters. In 2D we have $\tau = b/2$, where b is the width of the thread parallelograms. For higher dimensional space-time, we use the same formula only b is now the smallest spatial dimension of the thread parallelograms. The domain decomposition tries to tile the spatial dimensions equally so that τ becomes as large as possible without degrading data-to-core affinity.

IV. Results

A. Schemes

In the figures the following schemes are compared:

- **PeakDP**: Measured computational peak in double precision. We obtain this value by performing a sequence of independent multiply-add operations in registers. PeakDP models the absolute upper bound for any computation on a machine. It is clear that no optimization of stencil codes will reach this upper bound since stencil operations are not independent. The goal is to achieve a high fraction of this peak.
- **LL1Band0C**: Last-level cache bandwidth with zero further caching. It models the performance of a stencil code in case the domain could entirely fit into the last-level cache, but no higher level caches are present. Accordingly, for the case of 7-point constant stencil of order $s = 1$, 7 read and 1 write operations are performed from the last-level cache for each kernel execution. For the variable stencil case (banded-matrix), 14 reads (7 vector elements plus 7 matrix coefficients) and 1 write operations are counted. LL1Band0C represents the achievable performance in case of an enormous last level cache that could hold all data on-chip.
- **nuCATS**: Our NUMA-aware, cache-aware scheme from Section II; nuCATS is parallelized with pthreads and the kernel is vectorized using SSE2 intrinsics to prevent it from becoming compute-bound.
- **nuCORALS**: Our NUMA-aware, cache oblivious scheme from Section III; nuCORALS is parallelized with pthreads and the kernel is vectorized using SSE2 intrinsics to prevent it from becoming compute-bound.
- **CATS**: Our original cache aware time skewing scheme [1].
- **CORALS**: Our original cache oblivious parallelograms scheme [2].
- **Pochoir**: Code compiled using Phase II compilation of the Pochoir compiler and run-time system for implementing stencil computations on multicore processors [17]. We modify the kernel function of the 3D 7-point stencil example provided in the examples folder of the Pochoir package to implement (1) and use Pochoir's latest version v0.5 to compile it with the -O3 -ipo -xHost flags. Other flags suggested in the makefile either do not affect or worsen the performance.
- **PLuTo** code transformed by the automatic parallelizer and locality optimizer for multicores PLuTo version 0.7.0 [18]. We have tuned the tile sizes for our machines individually and use the transformation flags that yield the best performance. The transformed code is compiled with intel icc compiler version 12.1.2 with the -O3 -ipo -openmp -parallel flags and it reports successful vectorization of the loops.
- **SysBandIC**: System bandwidth with ideal caching. A performance estimate derived from the measured peak system bandwidth, see Table I. It assumes a sufficiently large cache that can hold at least 2 slices of the 3D domain or 2 lines of the 2D domain; therefore, for the case of 7-point constant stencil of order $s = 1$, 1 read and 1 write operations are performed from main memory for each kernel execution. For the variable

stencil case (banded-matrix), 8 reads (7 vector elements plus 7 matrix coefficients) and 1 write operations are counted. SysBandIC models the absolute upper bound for the performance of a naive implementation of stencil codes when the domain is too big to fit entirely into the cache.

- **NaiveSSE**: A Naive implementation with the following optimizations employed; parallelization using pthreads, kernel vectorization using SSE2 intrinsics, and NUMA-aware data allocation. We expect that the NaiveSSE curve will lie between SysBand0C and SysBandIC.
- **SysBand0C**: System bandwidth with zero-caching. In contrast to SysBandIC, it assumes there is no cache and thus all data accesses go to main memory. For the case of 7-point constant stencil of order $s = 1$, 7 read and 1 write operations are performed from main memory for each kernel execution. For the variable stencil case (banded-matrix), 14 reads (7 vector elements plus 7 matrix coefficients) and 1 write operations are counted. SysBand0C represents the lower bound for the performance of an efficient naive implementation of stencil codes.

The LL1Band0C, SysBandIC, and SysBand0C schemes assume that the bandwidth is the sole limiting factor, and all other factors (memory access latency, access to higher level memories, computation, etc.) are hidden behind it. Due to layout restrictions we refer in the figures to the suffix 'Band' with only the letter 'B'.

B. Testbed

Our testbed comprises constant and variable (banded matrix) 7-point stencils with order $s = 1$. Each stencil execution performs 7 multiplications and 6 additions amounting to 13 flops. A single stencil point update in 3D is described by

$$\begin{aligned} X^{t+1}_{i,j,k} = c_1 \cdot X^t_{i-1,j,k} + c_2 \cdot X^t_{i,j-1,k} + c_3 \cdot X^t_{i,j,k-1} \\ + c_4 \cdot X^t_{i+1,j,k} + c_5 \cdot X^t_{i,j+1,k} + c_6 \cdot X^t_{i,j,k+1} \\ + c_0 \cdot X^t_{i,j,k} \end{aligned} \quad (1)$$

where c_i, for $0 \leq i \leq 6$ are the stencil coefficients.

We show both weak and strong scalability of nuCATS, nuCORALS and the other schemes on the two machines whose specifications are listed in Table I. To prevent the early exploitation of another socket's system bandwidth before all cores on one socket are in use, we use the affinity routines to pin the thread contexts to cores on one socket, before occupying a new socket.

We demonstrate the weak scalability of nuCATS and nuCORALS on a 200^3 domain per core configuration, whereby the domain on which we compute in case of n threads is not an agglomeration of n separate 200^3 cubes, but one cube of volume $n \cdot 200^3$. Thus, with growing thread number, the weak scalability is not trivial, as it becomes more and more difficult to exploit data locality in the large data cubes. The strong scalability is presented for 160^3 and 500^3 domains. In the 160^3 case, the challenge is the shrinking working size for each thread that makes the inter-core communication become a bigger relative overhead. Not surprisingly it is therefore easier to obtain good scalability on the large 500^3 domain.

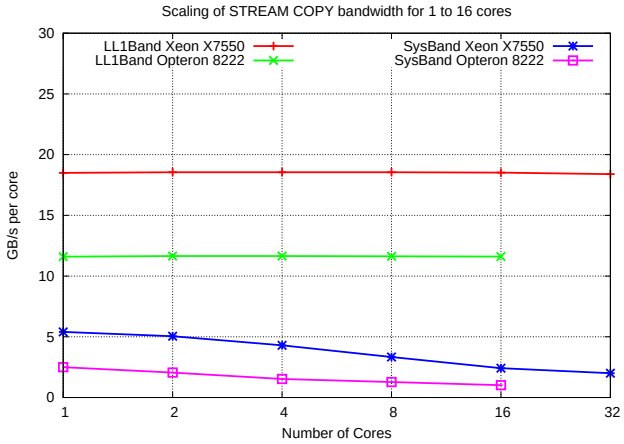

Figure (3): Scalability of last-level cache and system bandwidths for 1 to 16 threads on Opteron 8222 machine and for 1 to 32 threads on Xeon X7550 machine.

We run 100 iterations with two copies of X instead of in-place updates of Gauss-Seidel type with one one copy of X, since the two copy scenario is more general and challenging for temporal blocking. Temporal blocking is also beneficial for fewer iterations, e.g., to accelerate multiple smoother applications on each level of a multigrid solver, however, for a general performance comparison of temporal blocking schemes 100 iterations are more suitable.

All figures show the number of cores involved in executing the schemes on the x-axis. Figure 4 to Figure 15 have two y-axes; the left one shows how many giga updates of X^{t+1} can be executed per second (Gupdates/s) per core, and the right one shows the achieved GFLOPS per core with stencil (1), i.e., Gupdates/s times 13 in this case. We show Gupdates/s since it is a more informative measure when the performance of different stencils is compared, e.g., Gupdates/s hardly changes if we add another stencil point to (1) because the problem is still memory bound, however, the GLOPS number would change immediately. Since all graphs show results per core, a straight horizontal line means linear scaling with the number of cores.

C. Memory Bandwidth

Figure 3 shows how memory and cache bandwidths scale with the number of cores. For both machines, the cache bandwidth scales linearly with the number of cores, because each core has its own connection to the caches.

One thread does not saturate the memory bus. For the Opteron 8222 machine, the single-threaded memory bandwidth grows by a factor of 1.6x when 2 cores are used. The

Table I: Hardware configurations. The measured bandwidth numbers come from the STREAM COPY benchmark tool running with 16 and 32 threads and SSE reads. The measured peak double precision (DP) FLOPS come from our SSE benchmark consisting of independent multiply-add operations on registers. LL1 denotes last-level cache, LL2 denotes last but one level cache.

Brand	AMD	Intel
Processor	Opteron 8222	Xeon X7550
Code-named	Santa Rosa	Beckton
Frequency	3.0 GHz	2.0 GHz
Number of sockets	8	4
Cores per socket	2	8
L1 Cache per core	64 KiB	32 KiB
L2 Cache per core	1 MiB	256 KiB
L3 Cache per core	-	2.25 MiB
Operating system	Linux 64 bit	Linux 64 bit
Parallelization	1..16 pthreads	1..32 pthreads
Vectorization	SSE2	SSE2
Number of NUMA Nodes	8	4
Compiler	g++ 4.3.2	icpc 12.1.2
Measured L1 Bandwidth	675.3 GB/s	819.1 GB/s
Measured L2 Bandwidth	185.7 GB/s	642.8 GB/s
Measured L3 Bandwidth	-	588.6 GB/s
Measured Sys. Bandwidth	11.9 GB/s	63.0 GB/s
Measured Peak DP FLOPS	95.3 G	202.5 G
LL1 Band./Sys. Bandwidth	15.6	9.3
LL2 Band./LL1. Band.	3.6	1.1
Peak DP/(Sys. Band./8B) Arith. intensity for Sys.	64.1	25.7
Peak DP/(LL1 Band./8B) Arith. intensity for LL1	4.1	2.8

bandwidth increases on average by a factor of 1.5x when one additional socket is used up to 8 sockets (number of sockets in the machine). Overall, Opteron's single threaded bandwidth grows by 6.5x when all 16 cores on all sockets are employed.

For the the Xeon X7550 machine, memory bandwidth scales almost linearly from 1 to 2 cores; from 2 to 4 cores, bandwidth grows by 1.7x. Using all 8 cores on one socket saturates the bus since bandwidth increases by only 1.5x. Bandwidth grows by a factor of 1.4x when another socket is used. Overall, Xeon's single threaded bandwidth grows by a factor of 13.7x when all cores on the four sockets are engaged.

The cache and bandwidth performance numbers displayed in Figure 3 are used to define the benchmarks LL1Band0C, SysBandIC and SysBand0C. Clearly all schemes and benchmarks will achieve higher performance on the Xeon than on the Opteron due to the higher cache and memory bandwidths. However, the system bandwidth per core goes down significantly in both cases. So to obtain linear scalability with a temporal blocking scheme, the scheme has to create so much temporal locality and so few cache misses, that its scalability starts depending mostly on the linear scalability of the cache bandwidth rather than the degrading scalability of the system bandwidth. We will see that nuCATS and nuCORALS cannot decouple completely from the degrading scalability of the system bandwidth, however, in most case the scalability is much better.

D. Scalability for Constant Stencils

Figures 4 to 9 have the following common features:

NaiveSSE, SysBandIC and SysBand0C on the Xeon are faster than their counterparts on the Opteron.
The performance of these schemes depends on the system bandwidth. With 16 threads (2 sockets), the Xeon has 38.7 GB/s system bandwidth while the Opteron has only 11.9 GB/s, a ratio of 3.3, see Table I, and in fact NaiveSSE on the Xeon achieves a similar speedup factor of 2.7x over NaiveSSE on the Opteron.

LL1Band0C on the Xeon is faster than on the Opteron.
The performance of this benchmark depends on the last level cache bandwidth. LL1 cache bandwidth on the Xeon is faster than on the Opteron, see Table I and Figure 3.

The Xeon is much faster than the Opteron on nuCORALS and nuCATS schemes.
The performance of nuCATS and nuCORALS depends primarily on the cache bandwidth. The Xeon X7550 features larger and faster caches than the Opteron 2218. Both schemes exploit them effectively to reduce the impact of the slow accesses to the main memory.

The performance graph of NaiveSSE lies between SysBandIC and SysBand0C on both machines.
NaiveSSE scheme performs better than SysBand0C since SysBand0C assumes that 7 vector elements are fetched from main memory for each kernel execution, whereas in reality some of them are cached. SysBandIC on the other hand performs better than NaiveSSE since SysBandIC assumes ideal caching wherein only 2 memory transactions per update are necessary and additional overhead in the real execution is not considered.

Although LL1Band0C transfers 4x more data than SysBand0C, it shows higher performance.
This is not an inherent property of LL1Band0C vs. SysBandIC. For the 7-point stencil of order $s = 1$, SysBandIC, which assumes ideal spatial blocking, reads 1 double and writes 1 double, LL1Band0C, which assumes zero further caching, reads 7 doubles and writes 1 double. The ratio of transferred data by LL1Band0C to transferred data by SysBandIC is 4 which is far less than the ratios of last-level cache bandwidth to system bandwidth 15.6 and 9.3 for the Opteron and the Xeon, respectively. However, for high order stencils, the ratio of transferred data becomes larger and may yield that SysBandIC surpasses LL1Band0C.

nuCATS is better on the large domains, nuCORALS is better on the small domain.

nuCATS diverts almost all effort towards the maximal cache reuse in the last level cache at the expense of all other optimizations. If the domain is much bigger than the last level cache this strategy pays off as it minimizes main memory traffic which is 15.6x or 9.3x slower than the last level cache bandwidth on the Opteron and the Xeon, respectively. However, on smaller domains less aggressive last level cache optimization in nuCORALS also reduces main memory traffic to a small amount, and then its additional higher level cache optimization leads to better performance.

1) Opteron Results: Figures 4, 6, and 8 show that the performance curves of nuCATS and nuCORALS lie between SysBandIC and LL1Band0C on the Opteron. Being faster than SysBandIC means that both schemes transfer on average less than 2 doubles from main memory per stencil update due to the created space-time data locality. Despite the degrading system bandwidth both schemes show very good scalability up to 8 cores (nearly horizontal lines). When using all 16 cores of the machine, they become more affected by the system bandwidth limit and the 8 core performance of nuCORALS and nuCATS grows only by a factor of 1.6x and 1.7x.

Overall, Opteron's single-core performance on nuCORALS grows by a factor of 10.4x when using all 16 cores of the machine in Figure 4 (200^3 per core domain), 11.1x in Figure 6 (160^3 domain), and 10.7x in Figure 8 (500^3 domain). The highest fraction of the computational peak on 16 cores is reached in Figure 6, namely 26%. Opteron's single-core performance on nuCATS grows by a factor of 11.2x when using all 16 cores of the machine in Figure 4 (200^3 per core domain), 9.4x in Figure 6 (160^3 domain), and 11.2x in Figure 8 (500^3 domain). nuCATS achieves the highest fraction of the computational peak (28%) using all 16 cores of the machine.

2) Xeon Results: On the Xeon, nuCORALS not only surpasses SysBandIC, but also it beats the performance of LL1Band0C up to 4 cores. This means that even if gigabyte large domains could fit into the last level cache and would be processed completely on-chip, the already available performance of nuCORALS is still superior. This is a remarkable result, as it shows that a cache oblivious algorithm can draw so much benefit from higher level caches that it overcompensates for the remaining slow data accesses to main memory and performs on average better than the last level cache alone. However, for higher core counts than 4, the sublinear scaling of the main memory bandwidth renders it more and more difficult to beat LL1Band0C. Only on the 160^3 domain nuCORALS is still better than LL1Band0C with 8 cores. This is due to the big fraction of data cached in higher level caches and less main memory traffic compared to big domains, which compensates for the increasingly slower transfers from main memory.

nuCATS optimizes for the last level cache exclusively and in fact on the large domains it shows very similar performance to LL1Band0C up to 16 cores, only for 32 cores it falls off a bit. This is a big achievement, demonstrating an algorithmic decoupling from the slow main memory bandwidth, which is already severely degrading up to 16 cores, see Figure 3. At first it appears very surprising that nuCATS can even beat LL1Band0C in some cases, as it has some overheads and does not optimize for anything else than the last level cache. However, similar to the processing pattern of the naive scheme, there is some natural data reuse in higher level caches.

As already discussed nuCORALS is clearly better than nuCATS on the 160^3 domain, because the small domain allows high data reuse in higher level caches and the cache-oblivious nuCORALS automatically takes advantage of that.

Figures 5, 7, and 9 show that nuCORALS's per core performance falls off from 2 to 8 cores, because more and more threads compete for the shared last level cache. Despite the decreasing cache capacity per thread and the decreasing main memory bandwidth available to each thread (Figure 3) the drop in per-core performance is moderate. When additional sockets come into use, i.e., the transitions from 8 to $2 \cdot 8$ to $4 \cdot 8$ cores, nuCORALS maintains a near linear scalability. When all cores on all sockets are in use, nuCORALS achieves 52% of the measured computational peak performance. Overall, the Xeon's single-core performance grows on average by a factor of 22.0x when nuCORALS uses all 32 cores of the machine and by a factor of 22.7x when nuCATS uses all 32 cores of the machine.

To summarize, nuCATS and nuCORALS perform very well on both the Xeon and the Opteron. They show a near linear scalability where the system bandwidth scales almost linearly and still good scalability where system bandwidth scales only sublinearly.

E. Scalability for Banded Matrices

Common to all banded matrix figures is the omission of PeakDP, because its inclusion would severely compress all other graphs at the bottom. However, it is important to keep in mind that PeakDP is much higher than the displayed LL1Band0C and represents the real extent of the memory wall problem.

When the stencil coefficients are not constant, they must be stored in main memory. This corresponds to a banded matrix vector product. In this case, to exploit temporal locality, not only vector elements, but also the coefficients must reside in cache. Therefore, another 7 components along with each vector value must be fetched from main memory. This makes the problem even more memory-bound. When all 16 cores are used, nuCORALS's and nuCATS's aggregate performances drop by a factor of 6.6x and 7.6x, respectively, on both the 200^3 per core and 500^3 domains compared to the constant stencil case on the Opteron. Xeon's big L3 cache and relatively high system memory bandwidth (Table I) are able to mitigate the problem to some extent and therefore,

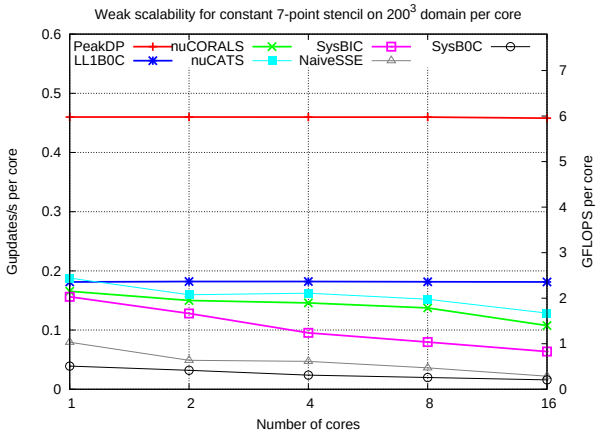

Figure (4): Constant stencil weak scalability for 1 to 16 threads with 200^3 doubles per thread and 100 timesteps on the Opteron 8222. GFLOPS achieved with 16 cores, PeakDP 95.3, LL1Band0C 37.7, nuCORALS 22.4, nuCATS 26.8, SysBandIC 13.2, NaiveSSE 4.6, SysBand0C 3.3

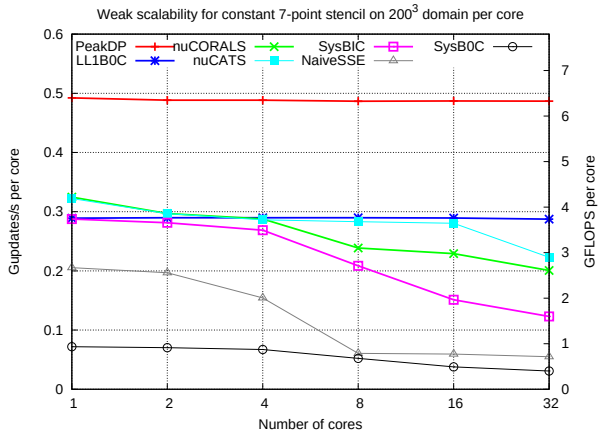

Figure (5): Constant stencil weak scalability for 1 to 32 threads with 200^3 doubles per thread and 100 timesteps on the Xeon X7550. GFLOPS achieved with 32 cores, PeakDP 202.5, LL1Band0C 119.6, nuCORALS 83.4, nuCATS 92.7, SysBandIC 51.2, NaiveSSE 22.9, SysBand0C 12.7

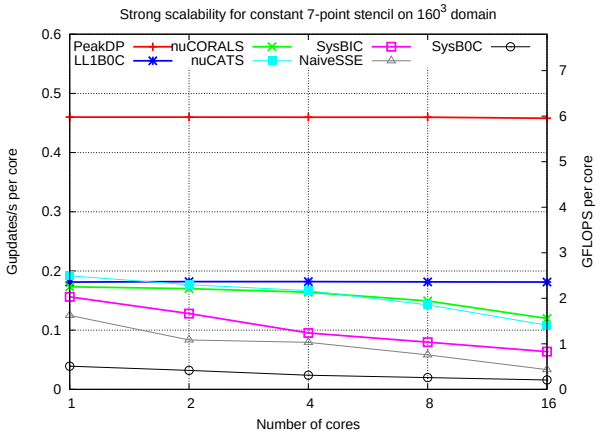

Figure (6): Constant stencil strong scalability for 1 to 16 threads on a 160^3 domain of doubles and 100 timesteps on the Opteron 8222. GFLOPS achieved with 16 cores, PeakDP 95.3, LL1Band0C 37.7, nuCORALS 24.9, nuCATS 22.5, SysBandIC 13.2, NaiveSSE 6.9, SysBand0C 3.3

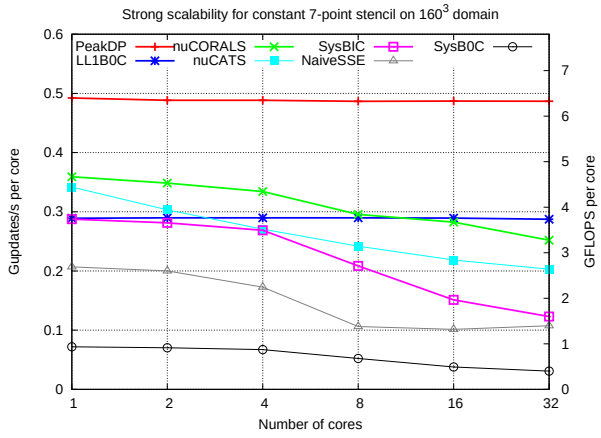

Figure (7): Constant stencil strong scalability for 1 to 32 threads on a 160^3 domain of doubles and 100 timesteps on the Xeon X7550. GFLOPS achieved with 32 cores, PeakDP 202.5, LL1Band0C 119.6, nuCORALS 104.8, nuCATS 84.5, SysBandIC 51.2, NaiveSSE 44.7, SysBand0C 12.7

its aggregate performance drops by a factor of only 3x for nuCORALS and 5x for nuCATS.

On the Opteron, the additional data transfers create a large gap between nuCORALS and nuCATS on the one side and LL1Band0C on the other side. Both schemes maintain a clear advantage over SysBandIC, however, the additional main memory traffic makes them also inherit its sublinear scalability. The single-threaded performance of nuCORALS and nuCATS accelerates by around 6x on the 160^3 (Figure 12) and 500^3 (Figure 14) domains and around 5x on the 200^3 per core problem (Figure 10), when all 16 cores are engaged in the computation. The latter is particularly difficult to accelerate because the dependence on the system bus grows super-linearly (linear in volume plus more tile boundaries in large volume), while system bandwidth per thread decreases.

On the Xeon, the additional transfer of matrix coefficients prevents nuCORALS from surpassing the performance of LL1Band0C as in the constant stencil case. Benefiting from the large shared last-level cache, the single-threaded performance of nuCORALS and nuCATS is much closer to LL1Band0C than on the Opteron. However, it falls off rapidly when more cores are engaged in the computation, because the advantage of the shared cache disappears when it has to be divided among all cores on the same socket (There is almost no data reuse between tiles of different threads). The corresponding reduction in per-core performance is particularly strong for nuCATS and the 4 to 8 core transition, because the available last-level cache capacity per thread is halved and system bandwidth scales particularly poorly for

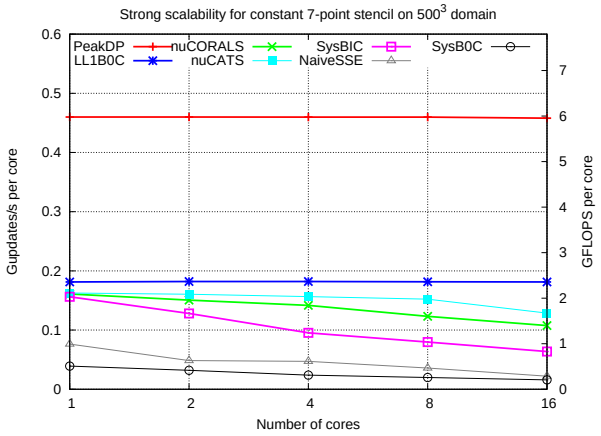

Figure (8): Constant stencil strong scalability for 1 to 16 threads on a 500^3 domain of doubles and 100 timesteps on the Opteron 8222. GFLOPS achieved with 16 cores, PeakDP 95.3, LL1Band0C 37.7, nuCORALS 22.4, nuCATS 26.8, SysBandIC 13.2, NaiveSSE 4.6, SysBand0C 3.3

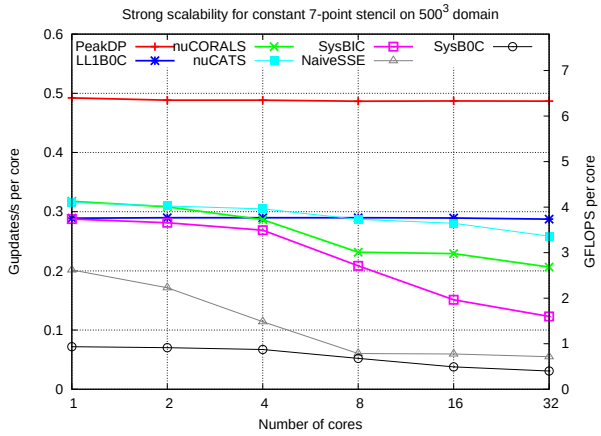

Figure (9): Constant stencil strong scalability for 1 to 32 threads on a 500^3 domain of doubles and 100 timesteps on the Xeon X7550. GFLOPS achieved with 32 cores, PeakDP 202.5, LL1Band0C 119.6, nuCORALS 85.9, nuCATS 107.6, SysBandIC 51.2, NaiveSSE 22.9, SysBand0C 12.7

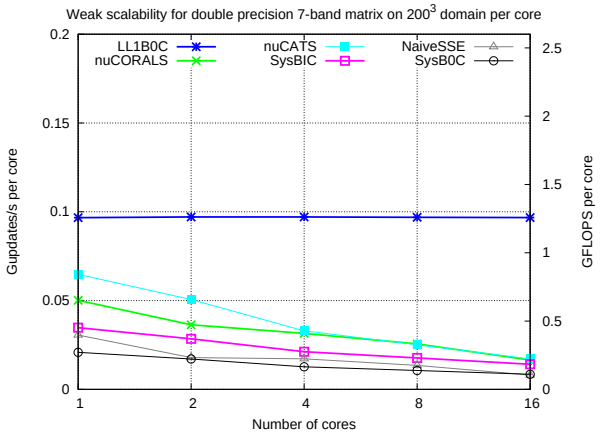

Figure (10): Banded matrix weak scalability for 1 to 16 threads with 200^3 doubles per thread and 100 timesteps on the Opteron 8222. GFLOPS achieved with 16 cores, PeakDP 95.3, LL1Band0C 20.1, nuCORALS 3.4, nuCATS 3.6, SysBandIC 2.9, NaiveSSE 1.7, SysBand0C 1.8

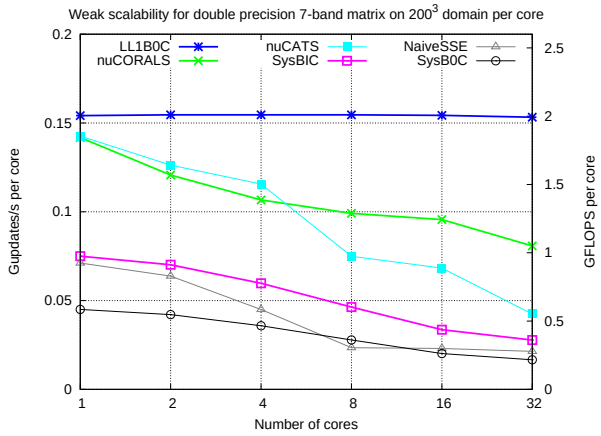

Figure (11): Banded matrix weak scalability for 1 to 32 threads with 200^3 doubles per thread and 100 timesteps on the Xeon X7550. GFLOPS achieved with 32 cores, PeakDP 202.5, LL1Band0C 63.8, nuCORALS 33.6, nuCATS 17.7, SysBandIC 11.3, NaiveSSE 8.9, SysBand0C 6.8

this transition, see Figure 3. Although the reduction from 4 to 8 cores is disproportionally large, on average nuCATS's performance correlates with SysBandIC. nuCORALS suffers a similar reduction in per-core performance on the 160^3 domain (Figures 13), however, maintains per-core performance on the bigger domains in Figures 13 and 15. The cache oblivious nature of the algorithm with the automatic exploitation of the entire cache hierarchy is of great help in these cases.

So nuCORALS is the clear winner against nuCATS for the banded matrix multiplication. It maintains more than 50% parallel efficiency on all domains, achieving speedups of 18.7x on the 200^3 per core domain, 16.3x on the 160^3 domain, and 22.5x on the 500^3 domain with 32 threads. nuCATS's per core performance is 9.3x higher than its single-threaded performance on the 200^3 per core domain, 11.3x on the 160^3 domain, and 14.4x on the 500^3 domain.

F. Scalability for High Order Stencils

Skewing thread and root parallelograms with a slope equal to the stencil order s makes it more challenging to achieve high performance and scalable results. We have more control overhead from additional boundary intersections and synchronizations, the tiles' surface to volume ratios increase and more surface layers must be kept on-chip, and a larger fraction of data is processed by one thread but owned by another in case of a fixed thread parallelogram height (τ in Figure 1). The last effect can be alleviated by setting $\tau = b/(2 \cdot s)$, which recovers the previous compromise between data-to-core affinity and temporal blocking.

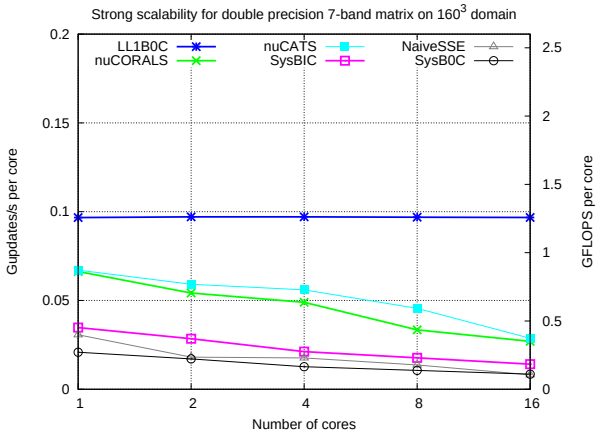

Figure (12): Banded matrix strong scalability for 1 to 16 threads on a 160^3 domain of doubles and 100 timesteps on the Opteron 8222. GFLOPS achieved with 16 cores, PeakDP 95.3, LL1Band0C 20.1, nuCORALS 5.6, nuCATS 6.0, SysBandIC 2.9, NaiveSSE 1.7, SysBand0C 1.8

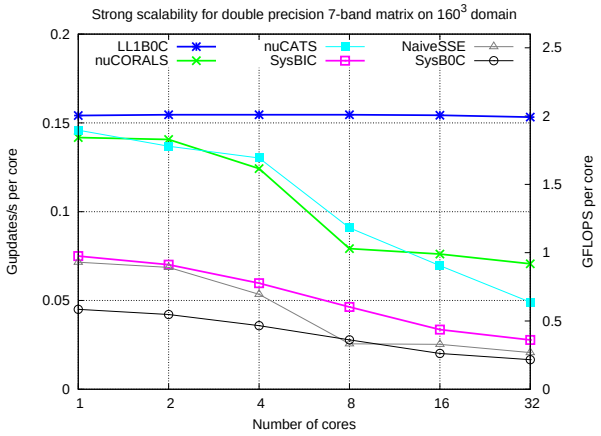

Figure (13): Banded matrix strong scalability for 1 to 32 threads on a 160^3 domain of doubles and 100 timesteps on the Xeon X7550. GFLOPS achieved with 32 cores, PeakDP 202.5, LL1Band0C 63.8, nuCORALS 29.4, nuCATS 20.4, SysBandIC 11.3, NaiveSSE 8.6, SysBand0C 6.8

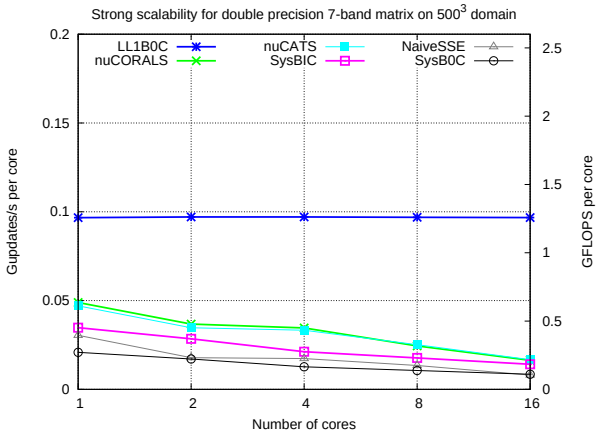

Figure (14): Banded matrix strong scalability for 1 to 16 threads on a 500^3 domain of doubles and 100 timesteps on the Opteron 8222. GFLOPS achieved with 16 cores, PeakDP 95.3, LL1Band0C 20.1, nuCORALS 3.4, nuCATS 3.5, SysBandIC 2.9, NaiveSSE 1.7, SysBand0C 1.8

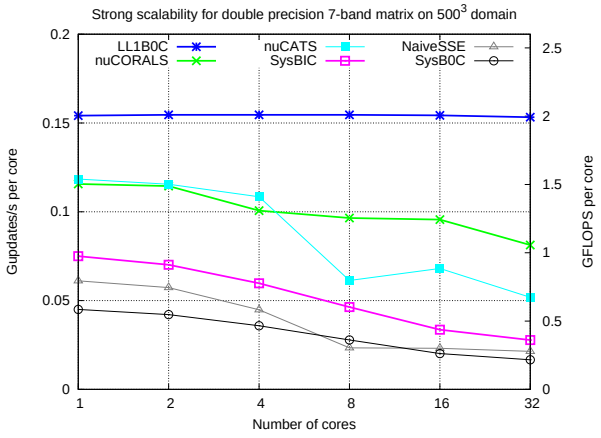

Figure (15): Banded matrix strong scalability for 1 to 32 threads on a 500^3 domain of doubles and 100 timesteps on the Xeon X7550. GFLOPS achieved with 32 cores, PeakDP 202.5, LL1Band0C 63.8, nuCORALS 33.8, nuCATS 21.6, SysBandIC 11.3, NaiveSSE 8.9, SysBand0C 6.8

Figures 16 to 19 show the scalability of nuCORALS for stencil orders $s = 1$, $s = 2$, and $s = 3$. Our model problem has 25 flops for $s = 2$ (13 multiplications and 12 additions), and 37 flops for $s = 3$ (19 multiplications and 18 additions). The scalability behavior of nuCORALS and nuCATS for $s = 2$ and 3 is not much different compared to the $s = 1$ case discussed above. The absolute performance clearly decreases, however, as a very positive result we observe that the decrease from $s = 1$ to $s = 2$ is less than 2x, and from $s = 1$ to $s = 3$ less than 3x, although the convex hull of the stencil required for spatial locality on-chip grows cubically.

G. Performance Comparison

This section highlights the importance of data-to-core affinity by comparing the performance of nuCORALS and nuCATS with other recent temporal blocking schemes from literature: CATS [1], CORALS [2], PLuTo 0.7.0 [18], and Pochoir 0.5 [17]. All schemes but nuCORALS, nuCATS, and NaiveSSE do not explicitly pay attention to this requirement; therefore, we anticipate that they will exhibit worse scalability beyond one NUMA node.

Figures 20, 21, and 22 show that the performances of CORALS and CATS are on par with their NUMA-aware counterparts using one core since each scheme is similar to its NUMA-aware counterpart. However, when up to 8 cores (one socket) are engaged in the computation, the graphs of CORALS vs. nuCORALS and CATS vs. nuCATS already drift apart, although both are still running on the same NUMA node. The per-thread local data allocation in nuCATS and nuCORALS helps also the efficient utilization

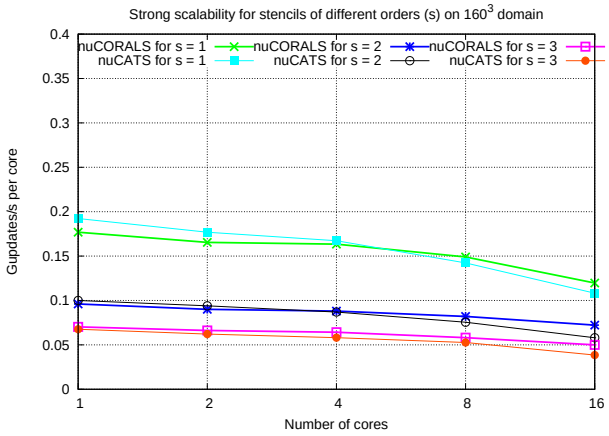

Figure (16): Strong scalability for high order stencils on a 160^3 domain of doubles and 100 timesteps on the Opteron 8222. GFLOPS achieved for $s = 1$ nuCORALS 24.9, nuCATS 22.5. For $s = 2$ nuCORALS 28.9, nuCATS 23.2. For $s = 3$ nuCORALS 29.6, nuCATS 22.8

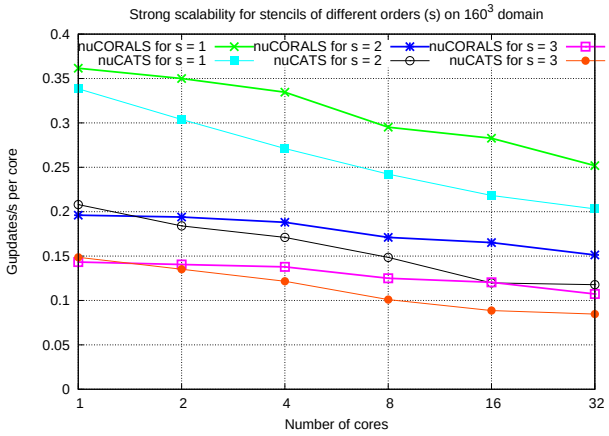

Figure (17): Strong scalability for high order stencils on a 160^3 domain of doubles and 100 timesteps on the Xeon X7550. GFLOPS achieved with 32 cores for $s = 1$ nuCORALS 104.8, nuCATS 84.5. For $s = 2$ nuCORALS 121, nuCATS 94.2. For $s = 3$ nuCORALS 127, nuCATS 100.3

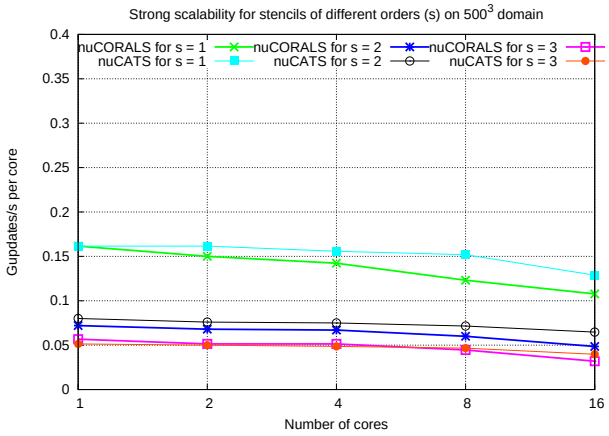

Figure (18): Strong scalability for high order stencils on a 500^3 domain of doubles and 100 timesteps on the Opteron 8222. GFLOPS achieved for $s = 1$ nuCORALS 22.4, nuCATS 26.8. For $s = 2$ nuCORALS 19.4, nuCATS 25.9. For $s = 3$ nuCORALS 18.9, nuCATS 23.5

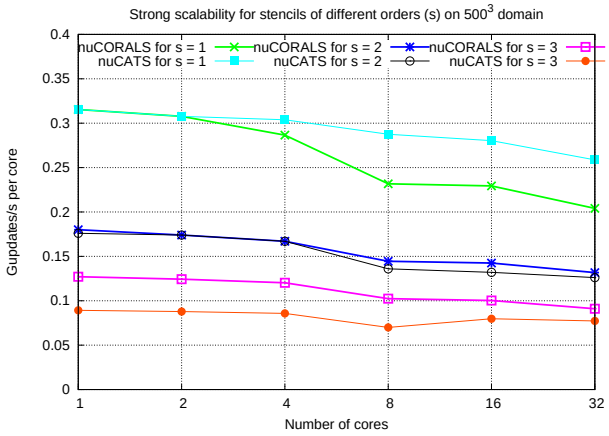

Figure (19): Strong scalability for high order stencils on a 5003 domain of doubles and 100 timesteps on the Xeon X7550. GFLOPS achieved with 32 cores for $s = 1$ nuCORALS 85.9, nuCATS 107.6. For $s = 2$ nuCORALS 105.4, nuCATS 100.9. For $s = 3$ nuCORALS 107.7, nuCATS 91.5

of multi-channel memory buses. The difference between CATS and nuCATS is smaller than between CORALS and nuCORALS, because nuCORALS underwent more significant changes including a second level tiling for more coarse-granular parallelization and synchronization.

The NUMA importance is underlined when more than 8 cores are engaged in the computation. All non-NUMA-aware schemes suffer a big slowdown in the per-core metric as the computation goes beyond one NUMA node; nuCATS and nuCORALS on the other hand maintain a high, rather stable per-core performance level. Pochoir is quite stable up to 8 cores and then drops off sharply, while PLuTo's per-core performance degrades gradually with the number of cores.

Figure 22 reports strong scaling on a rather small domain and shows particularly dramatic performance degradation on all schemes that do not observe the data-to-core affinity. For 32 cores the naive scheme is clearly faster (more than 2.5x) than all non-NUMA-aware temporal blocking schemes apart from CATS, which is only slightly worse; nuCATS and nuCORALS maintain a clear advantage of around 2x over the naive scheme.

The overall performance of nuCATS and nuCORALS grows favorably when more sockets are engaged, while NUMA ignorance can even lead to a drop in the overall performance: for Pochoir from 16 to 32 cores on all domains, for CORALS from 8 to 16 to 32 cores on the 160^3 domain and from 16 to 32 cores on the 500^3 domain. The drop in the overall performance of the NaiveSSE scheme occurs already for 8 cores, because the partitioned caches offer less opportunity for data reuse. But since it observes the data-

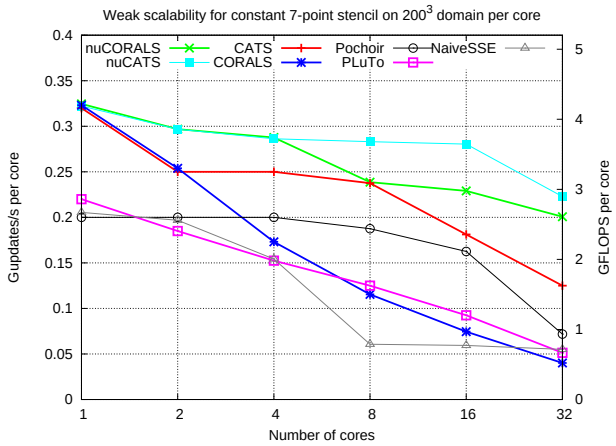

Figure (20): Constant stencil weak scalability for 1 to 32 threads with 200^3 doubles per thread and 100 timesteps on the Xeon X7550. GFLOPS achieved with 32 cores, nuCORALS 83.4, nuCATS 92.7, CATS 52, CORALS 16.7, Pochoir 29.9, PLuTo 21.3, NaiveSSE 22.9

to-core affinity requirement, it scales linearly beyond one NUMA node.

In summary, we see that data-to-core affinity is critical for maintaining performance beyond one NUMA node and also helps on a single socket with a multi-channel bus. Ignorance of the NUMA aspect in today's memory systems can even lead to the situation that a naive scheme which observes this aspect outperforms more sophisticated schemes that ignore it.

V. Conclusions

Spatio-temporal locality, parallelism, regular memory access and data-to-core affinity are all key requirements to achieve high performance on iterative stencil computations. We have shown that a systematic treatment of these requirements brings forth schemes that deliver high absolute performance and overall good scalability on many-core systems.

Analysis of our previous schemes CATS and CORALS, and other temporal blocking algorithms that do not take data-to-core affinity into account demonstrates a huge per-core slowdown when scaling beyond one NUMA node; sometimes this even results in a drop of overall performance. Our new schemes nuCORALS and nuCATS on the other hand continue to benefit from additional cores even in the case of strong scaling on a small domain.

Acknowledgment

We would like to thank Yuan Tang and the Pochoir stencil compiler team for granting us an early access to their code for testing.

References

[1] R. Strzodka, M. Shaheen, D. Pajak, and H.-P. Seidel, "Cache accurate time skewing in iterative stencil computations," in *Proceedings of the International Conference on Parallel Processing (ICPP)*, Sep. 2011.

[2] R. Strzodka, M. Shaheen, D. Pajak, and H.-P. Seidel, "Cache oblivious parallelograms in iterative stencil computations," in *ICS '10: Proceedings of the 24th ACM International Conference on Supercomputing*. ACM, 2010, pp. 49–59.

[3] M. Wittmann, G. Hager, and G. Wellein, "Multicore-aware parallel temporal blocking of stencil codes for shared and distributed memory," in *Proc. Workshop on Large-Scale Parallel Processing (LSPP'10) at IPDPS'10*, 2010.

[4] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick, "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, 2008, pp. 1–12.

[5] S. Kamil, C. Chan, L. Oliker, J. Shalf, and S. Williams, "An auto-tuning framework for parallel multicore stencil computations," in *International Parallel & Distributed Processing Symposium (IPDPS)*, 2010.

[6] M. Christen, O. Schenk, and H. Burkhart, "Patus: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures," in *IPDPS '11: Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium*, 2011.

[7] S. Krishnamoorthy, M. Baskaran, U. Bondhugula, J. Ramanujam, A. Rountev, and P. Sadayappan, "Effective automatic parallelization of stencil computations," *SIGPLAN Not.*, vol. 42, no. 6, pp. 235–244, 2007.

[8] L. Renganarayana, M. Harthikote-Matha, R. Dewri, and S. Rajopadhye, "Towards optimal multi-level tiling for stencil computations," in *Proceedings of International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2007.

[9] M. Griebl, "Automatic parallelization of loop programs for distributed memory architectures," University of Passau, Jun. 2004, habilitation thesis.

[10] D. Kim, L. Renganarayanan, D. Rostron, S. V. Rajopadhye, and M. M. Strout, "Multi-level tiling: M for the price of one," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2007, p. 51.

[11] M. M. Baskaran, A. Hartono, S. Tavarageri, T. Henretty, J. Ramanujam, and P. Sadayappan, "Parametrized tiling revisited," in *Proc. of the International Symposium on Code Generation and Optimization (CGO'10)*, 2010.

[12] M. Frigo and V. Strumpen, "The cache complexity of multithreaded cache oblivious algorithms," in *SPAA '06: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*. New York, NY, USA: ACM, 2006, pp. 271–280.

[13] G. E. Blelloch, P. B. Gibbons, and H. V. Simhadri, "Low depth cache-oblivious algorithms," Carnegie Mellon University, Tech. Rep., 2009.

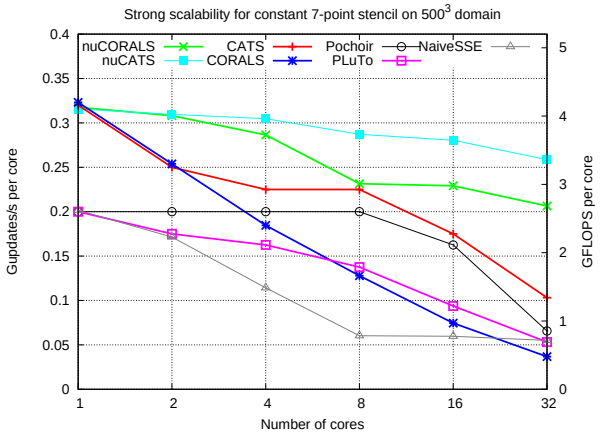

Figure (21): Constant stencil strong scalability for 1 to 32 threads on a 500^3 domain of doubles and 100 timesteps on the Xeon X7550. GFLOPS achieved with 32 cores, nuCORALS 85.9, nuCATS 107.6, CATS 42.9, CORALS 15.3, Pochoir 27.3, PLuTo 22.1, NaiveSSE 22.9

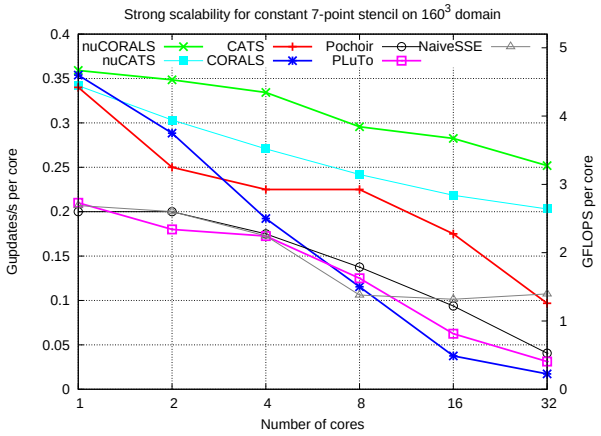

Figure (22): Constant stencil strong scalability for 1 to 32 threads on a 160^3 domain of doubles and 100 timesteps on the Xeon X7550. GFLOPS achieved with 32 cores, nuCORALS 104.8, nuCATS 84.5, CATS 40.3, CORALS 7.2, Pochoir 16.9, PLuTo 13, NaiveSSE 44.7

[14] D. Wonnacott, "Using time skewing to eliminate idle time due to memory bandwidth and network limitations," in *Proceedings of International Parallel and Distributed Processing Symposium*, 2000.

[15] V. Strumpen and M. Frigo, "Software engineering aspects of cache oblivious stencil computations," IBM Research, Tech. Rep., 2006.

[16] S. Kamil, P. Husbands, L. Oliker, J. Shalf, and K. Yelick, "Impact of modern memory subsystems on cache optimizations for stencil computations," in *MEMORY SYSTEM PERFORMANCE*. ACM, 2005, pp. 36–43.

[17] Y. Tang, R. A. Chowdhury, B. C. Kuszmaul, C.-K. Luk, and C. E. Leiserson, "The pochoir stencil compiler," in *SPAA*. ACM, 2011, pp. 117–128.

[18] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A practical automatic polyhedral parallelizer and locality optimizer," *SIGPLAN Not.*, vol. 43, no. 6, pp. 101–113, 2008.