Overcoming Bandwidth Limitations in Visual Computing

Robert Strzodka Mohammed Shaheen
Max Planck Institut Informatik, Saarbrücken

Dawid Pajak
West Pomeranian University of Technology, Szczecin

***Abstract*—Because visual computations are very data intensive they are often limited by the bandwidth of the system rather than its peak computational performance. The trend towards many-core architectures exacerbates the problem because the parallel cores let the compute capability grow exponentially while the system bandwidth increases only linearly. At the core of the bandwidth problem in visual computing are iterative loops over discrete local operators. A typical representative is a stencil computation with constants weights or a sparse matrix vector product in case of variable weights. This computations are bandwidth limited because of their low arithmetic intensity. We have developed techniques that accelerate these loops beyond the peak bandwidth limit.**

While different cache friendly approaches have been tried in the past, they have never been so successful. For example our quad-core Xeon X5482 3.2GHz system reaches 25.1 GFLOPS in double precision on a stencil computation in registers. On a large 3D domain of 500^3 doubles and 100 iterations a hand-vectorized single-threaded naive stencil implementation achieves 1.6 GFLOPS and there is no improvement in the multi-threaded version because the system memory bandwidth limits the performance. A state-of-art automatic loop transformation framework Pluto [1] achieves 1.9 GFLOPS for this stencil computation with four threads. In comparison, our scheme performs already at 5.3 GFLOPS with a single thread and soars to 13.0 GFLOPS with four threads.

I. Introduction

The real-time nature of visual computing requires the processing of large amounts of data at interactive rates, thus driving the trend towards high performance, many-core computing. However, many-core parallelism alone is not sufficient as the huge amounts of data first have to be transferred to the processing elements, which dramatically increases the off-chip bandwidth requirements. A large fraction of visual simulations is already memory bound and due to the growing number of parallel cores in CPUs the discrepancy between system bandwidth requirements and system bandwidth performance threatens to widen even further.

In response to this threat of the bandwidth wall problem dual-, triple- and quad-channel memory interfaces have been introduced. They alleviate the problem temporarily but their scaling is too expensive to keep up with the exponentially growing number of cores. So for data intensive applications the situations is already bad and is bound to become worse and worse.

In contrast to system bandwidth, the aggregate cache bandwidth scales naturally with the number of cores if each core has a dedicated connection to its cache. Then doubling the number of cores also doubles the number of connections and thus the aggregate cache bandwidth. So ideally we would like data intensive applications to scale with the cache bandwidth rather than the system bandwidth.

II. Related Work

To exploit temporal locality in stencil computations so called *time skewing* schemes have been introduced ten years ago by Wolf [2], Song et al, [3] and Wonnacott [4]. However, despite very satisfactory theoretical results, practical benefits have been limited. When comparing against a naive scheme with simple compilation, good speedups can be obtained through automatic code transformation in many cases [1]. But automatic code transformation is far less competitive against hand-optimized naive scheme [5]. It turns out that the theoretically best solutions are not well suited for all characteristics of current hardware architectures. A very delicate balance between the theory and hardware requirements must be found. Our algorithm is not a general loop transformation framework but rather specific to iterative stencil computations, and there it demonstrates truly high performance benefits against a fully optimized naive scheme by performing well above 10 GFLOPS in 3D on a quad-core processor.

III. Algorithm

Clearly, an algorithm operating repeatedly on gigabyte large domains cannot become completely independent of the system bandwidth as the data must be read again and again from system memory. But for certain iterative stencil computations we have developed a cache accurate time skewing (CATS) algorithm that scales very favorably with increased cache bandwidth. The algorithm works both for constant and variable stencils and is efficient for 1D, 2D and 3D spatial dimensions, i.e. iterative applications of sparse banded matrices are supported. Moreover, the stencil computation may involve additional data from other vectors, so typical iterative linear equation solvers like Jacobi or Gauss-Seidel solver can be accelerated in this fashion. In the field of visual computing this means the applicability to many iterative algorithms in image denoising, segmentation, registration or physical simulations. The main current limitation of CATS is the restriction to 3-wide stencils in every dimesion, i.e. a 27-point stencil in 3D is supported, but a 125-point stencil would fail. We plan on extending our algorithm to more general stencils in future.

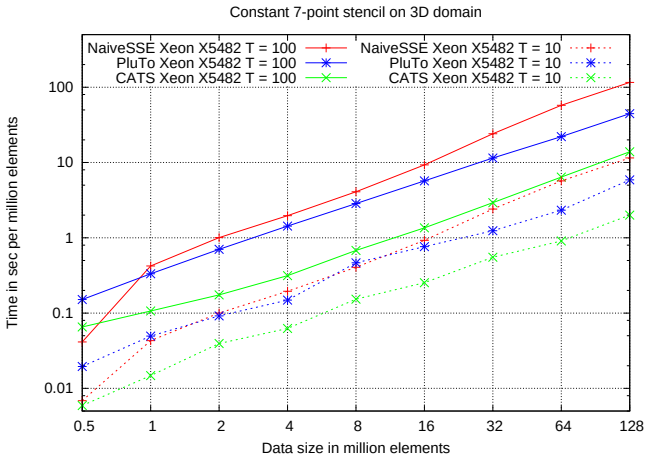

Fig. 1. Timings of the Xeon X5482 with constant stencils in 3D. GFLOPS for 128 million elements with $T = 100$: NaiveSSE 1.6, Pluto 1.9, CATS 13 .

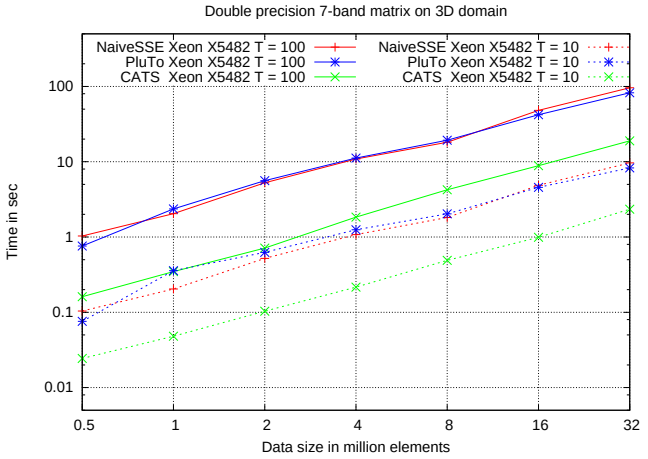

Fig. 2. Timings of the Xeon X5482 with a banded matrix in 3D. GFLOPS for 64 million elements with $T = 100$: NaiveSSE 0.6, Pluto 0.5, CATS 2.5 .

IV. Results

All shown result have been obtained on a quad-core Xeon X5482 (Harpertown) 3.2 GHz. The syntheticly benchmarked system bandwidth delivered 6.2 GB/s, a synthetic stencil computation on registers delivered 25.1 GFLOPS. All code was compiled with g++-4.3 and executed on a 64-bit Linux operating system.

We compare the execution times on 3D domains of varying sizes from 0.5 to 128 million elements and either $T = 10$ or $T = 100$ iterations. The compared algorithms are:

- NaiveSSE: Hand parallelized and vectorized naive code.
- Pluto: An automatic parallelizer and locality optimizer for multicores [1]
- CATS: Our cache accurate time skewing scheme.

Figure 1 shows the astounding result, that the execution time of our CATS algorithm (green) scales with the data size linearly as though everything would be always in the cache, whereas the naive algorithm (red) suffers a big slow down when data starts to exceed the cache size at the transition from 0.5 to 1 million elements (logarithmic axes). The Pluto results (blue) perform similar to the NaiveSSE scheme being slightly worse for small domain sizes and slightly better for large domain sizes. Consequently, 10 iterations of the other schemes require approximately the same time as 100 iterations of our CATS scheme. These results are so strong that at first they appear impossible because we operate on gigabytes of data at speeds that suggest that all data resides in the cache, which means our CPU would require 2 GiB of on-chip cache. Such performance cannot be obtained with traditional methods of cache blocking or prefetching because they are still limited by the system bandwidth. The key ingredient is a very complex space-time index reordering ensuring that almost all data accesses are resolved in the cache, despite its small size (12 MiB) in comparison to the overall data amount (up to 2 GiB). Accordingly, the obtained speedup depends mainly on the ratio of the cache to system bandwidth. Figure 2 shows 3D results for the same domains but this time the stencil varies across the domain, so we have a matrix vector product with a sparse banded matrix. Again our scheme clearly outperforms the competing schemes. Note that all schemes compute exactly the same result, they only traverse the space-time of indices in a different order.

V. Conclusions

The exponential growth of cores on CPUs leads to a bandwidth wall problem where limited off-chip bandwidth capabilities severely restrict the performance of visual computations. We suggest to overcome these problems by computational schemes that scale mainly with the aggregate cache bandwidth rather than the system bandwidth. While at first this seems impossible for gigabyte large domains that can never fit into caches, our cache accurate time skewing schemes do deliver this type of strong scalability for certain stencil computations. Our results clearly outperform hand-optimized naive and state-of-art automatic loop transformation code by factors 4 to 6 on the quad-core Xeon X5482. In general the acceleration potential depends mainly on the ratio of the cache to system bandwidth. In future we will work on providing a convenient application interface for accessing these algorithms and removing the current limitations on the stencil sizes.

References

[1] 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.

[2] M. Wolf, "More iteration space tiling," in *Proceedings of Supercomputing '89*, 1989.

[3] Y. Song and Z. Li, "New tiling techniques to improve cache temporal locality," in *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, 1999.

[4] 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.

[5] S. Kamil, K. Datta, S. Williams, L. Oliker, J. Shalf, and K. Yelick, "Implicit and explicit optimizations for stencil computations," in *MSPC '06: Proceedings of the 2006 workshop on Memory system performance and correctness*. ACM, 2006, pp. 51–60.