Advanced tiling techniques for memory-starved streaming numerical kernels

Tareq Malas[1], Georg Hager[2], Hatem Ltaief[1], and David Keyes[1]

[1]Extreme Computing Research Center – King Abdullah University of Science and Technology

[2]Erlangen Regional Computing Center (RRZE)

ABSTRACT Many temporal blocking techniques for stencil algorithms have been suggested for speeding up memory-bound code via improved temporal locality. Most of the established work concentrates on updating separate cache blocks per thread, which works on all types of shared memory systems, regardless of whether there is a shared cache. The downside of this approach is that the cache space for each thread can become too small for accommodating a sufficient number of updates and eventually decouple from memory bandwidth. In this poster we introduce a generalized multi-dimensional intra-tile parallelization scheme for shared-cache multicore processors that results in a significant reduction of cache size requirements. It ensures data access patterns that allow efficient hardware prefetching and TLB utilization. We describe the approach and some implementation details, and we show that our solution is consistently faster than the state-of-the-art stencil frameworks PLUTO and Pochoir. We also prove the superiority of cache block sharing using accurate cache block size and code balance models.

STENCIL COMPUTATIONS

- Bottleneck and time-dominant in many scientific codes
- Appear in finite difference, element, and volume discretizations of PDEs

PDE Example: $\frac{\partial^2 u}{\partial t^2} = \nabla \cdot (\alpha(x) \nabla u)$

Fig.1: 7-point stencil

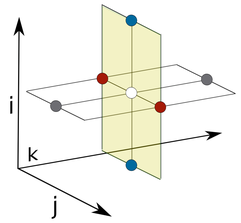
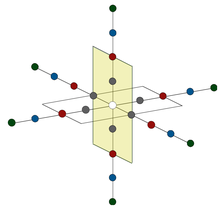


Fig.2: 25-point stencil



CHALLENGES IN CONTEMPORARY AND FUTURE ARCHITECTURES

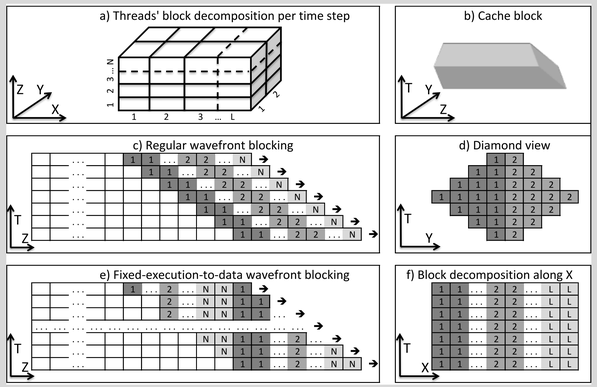
- Increasing gap between compute and memory performance
  → **low utilization** of the compute resources
- Small cache per core
  → **insufficient data reuse** to decouple from main memory
- No hardware synchronization facilities
  → **expensive** global **synchronization**

CHALLENGES FOR OPTIMIZING STENCIL ALGORITHMS

- **High-order** (long-range) stencils
  - steep tile slope → **large cache block size** required
  - large intra-cache traffic → **low memory pressure**
  - register shortage → **low-level code** is a challenge
- **Variable coefficient** stencils
  - coefficient data → **large cache block size** required
  - **low** computational **intensity**

APPROACH: MULTI-DIMENSIONAL INTRA-TILE PARALLELIZATION

Fig.3: Multi-dimensional parallelization of wavefront diamond blocking, showing the threads assignment in space-time dimensions



SOFTWARE INFRASTRUCTURE

Fig.4: Girih system components

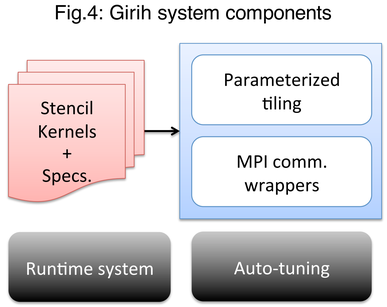
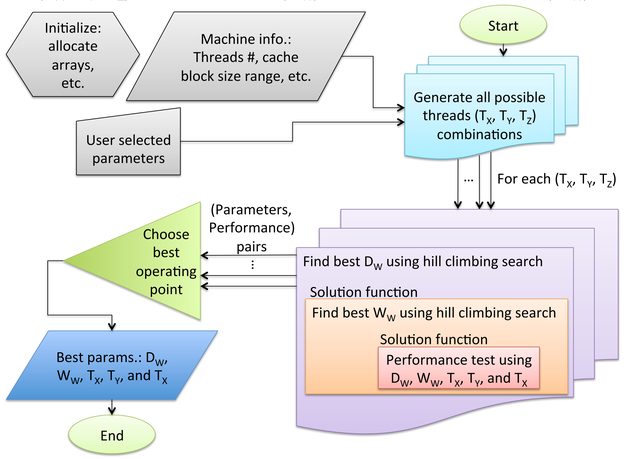


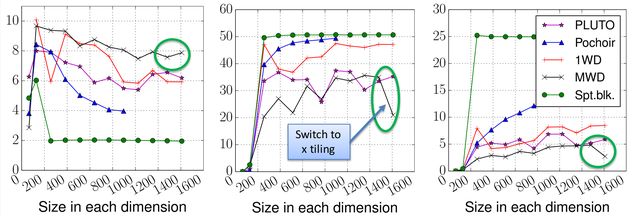
Fig.5: Auto-tuning: finds best configuration of threads decomposition (T_X, T_Y, T_Z), Diamond width (D_W), and wavefront tile width (W_W)



RESULTS: INCREASING GRID SIZE*

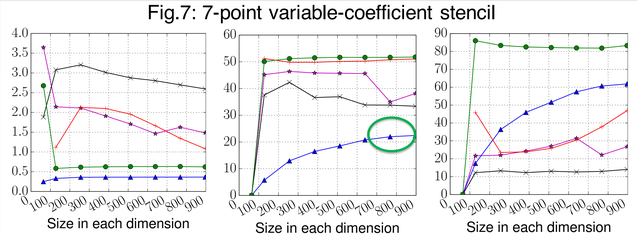
Performance GLUP/s** | **Memory bandwidth Gbytes/s** | **Memory transfers Bytes/LUP**

Fig.6: 7-point constant-coefficient stencil



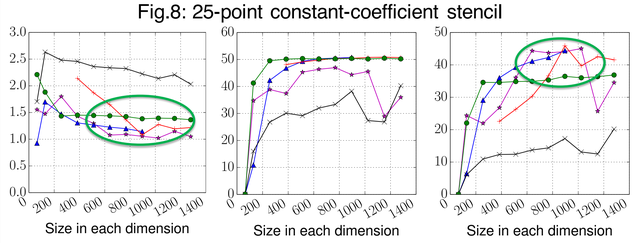
- MWD gets highest performance (GLUP/s)
- ≥ 4x vs. optimal spatial blocking
- Lowest memory traffic per LUP
- Lowest memory bandwidth
- Thread group size 2 or 6 for most sizes

Fig.7: 7-point variable-coefficient stencil



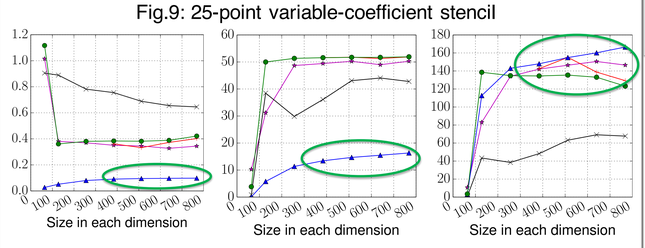
- MWD gets highest performance (GLUP/s)
- 4-5x vs. optimal spatial blocking
- High mem BW pressure due to low intensity
- Lowest memory traffic per LUP
- Low mem BW does not guarantee high performance

Fig.8: 25-point constant-coefficient stencil



- MWD gets highest performance (GLUP/s)
- Speedup limited due to intra-cache traffic
- MWD benefits from lower cache size needs
- Lowest memory traffic per LUP
- PLUTO, Pochoir fail at reducing memory pressure

Fig.9: 25-point variable-coefficient stencil



- Only MWD faster than optimal spt. blocking
- Speedup limited due to intra-cache traffic
- High mem BW pressure due to low intensity
- Low memory BW does not guarantee high performance
- PLUTO, Pochoir fail at reducing memory pressure

COMPUTATIONAL INTENSITY AND CACHE BLOCK SIZE MODEL

Fig.10: 7-point constant-coefficient $N=960^3$

- To achieve 5x reduction in code balance, 30 MiB cache block size is required (10 cores * 3MiB cache/core)

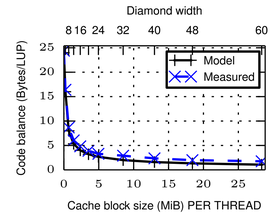


Fig.11: 7-point variable-coefficient $N=680^3$

- To achieve 4x reduction in code balance, 40 MiB cache block size is required (10 cores * 4MiB cache/core)

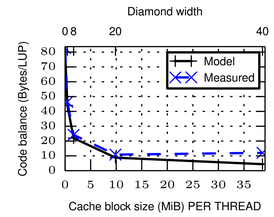
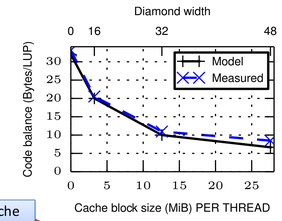


Fig.12: 25-point constant-coefficient $N=960^3$

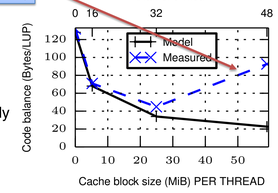
- Smallest cache block for a single core already occupies the cache memory



Measurements deviate when cache block exceeds available cache memory

Fig.13: 25-point variable-coefficient $N=480^3$

- Smallest cache block for a single core already occupies the cache memory



SUMMARY OF SIGNIFICANCE

- **Our approach is consistently faster than the state-of-the-art stencil frameworks**
- **Significant reduction in cache and memory BW requirements**
- **Architecture-friendly intra-tile cache block sharing**
- **Efficient hierarchical cache blocking**
- **Reduced data traffic in cache hierarchy through fixed-execution to data**

REFERENCES

[1] T. Malas, G. Hager, H. Ltaief, H. Stengel, G. Wellein, & D. Keyes. *Multicore-optimized wavefront diamond blocking for optimizing stencil updates*. SIAM J. on Sci. Comput., 37(4):439-464, 2015. DOI. 10.1137/140991133

[2] G. Wellein, G. Hager, T. Zeiser, M. Wittmann, & H. Fehske *Efficient temporal blocking for stencil computations by multicore-aware wavefront parallelization.* Proc. COMPSAC, 1:579-586, 2009. DOI: 10.1109/COMPSAC.2009.82

Acknowledgments: KAUST Extreme Computing Research Center for support of T. Malas

*Using 1-socket 18-core Intel Haswell processor (E5-2699-v3) @ 2.3GHz, with 45 MiB L3 cache, using Intel C compiler version 15

**Giga Lattice-site Updates per second

LEGEND - MWD: Our approach, Multi-core wavefront diamond blocking. **1WD:** 1-core wavefront diamond blocking. **Spt.blk.:** Spatial blocking

Code available at github.com/tareqmalas/girih