Optimizing Stencil Computations: Multicore-optimized Wavefront Diamond Blocking on Shared and Distributed Memory Systems

Tareq Malas[1], Georg Hager[2], Hatem Ltaief[1], Holger Stengel[2], Gerhard Wellein[2], and David Keyes[1]

[1]Extreme Computing Research Center - King Abdullah University of Science and Technology [2]Erlangen Regional Computing Center (RRZE)

ABSTRACT The importance of stencil-based algorithms in computational science has focused attention on optimized parallel implementations for multilevel cache-based processors. Temporal blocking schemes leverage the large bandwidth and low latency of caches to accelerate stencil updates and approach theoretical peak performance. A key ingredient is the reduction of data traffic across slow data paths, especially the main memory interface. In this work we combine the ideas of multi-core wavefront temporal blocking and diamond tiling to arrive at stencil update schemes that show large reductions in memory pressure compared to existing approaches. The resulting schemes show performance advantages in bandwidth-starved situations, which are exacerbated by the high bytes per lattice update case of variable coefficients. We present performance results on a contemporary Intel processor.

STENCIL COMPUTATIONS

- Bottleneck and time-dominant in widely used scientific codes
- Appear in finite difference, element, and volume discretizations of PDEs
  - Example:

$$\frac{\partial^2 u}{\partial t^2} = \nabla \cdot \left(\alpha(x) \nabla u\right)$$

Fig. 1: 7-point variable-coefficient stencil code

```
for t in Nt

  for i in Nz
    for j in Ny
      for k in Nx
        U(i,j,k)= C0(i,j,k)*V(i  , j  , k  )
                + C1(i,j,k)*V(i+1, j  , k  )
                + C1(i,j,k)*V(i-1, j  , k  )
                + C2(i,j,k)*V(i  , j+1, k  )
                + C2(i,j,k)*V(i  , j-1, k  )
                + C3(i,j,k)*V(i  , j  , k+1)
                + C3(i,j,k)*V(i  , j  , k-1)
  V = U
```

Fig. 2: 7-point stencil

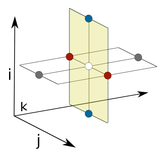
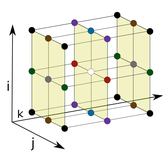


Fig. 3: 27-point stencil



CHALLENGES IN FUTURE ARCHITECTURES

Nodes may have hundreds of shared-memory cores with:

- small cache size per core
- expensive global synchronization
- small memory bandwidth per core
- complex cache sharing among cores
- interaction between heterogeneous processors

Fig. 4.1: 25-point constant-coefficient stencil thread scaling (grid: 980³)

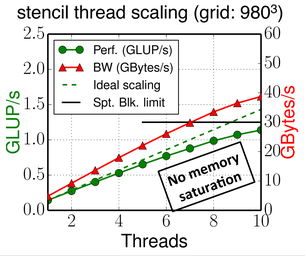
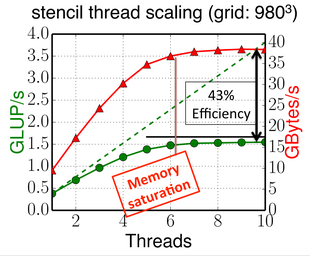
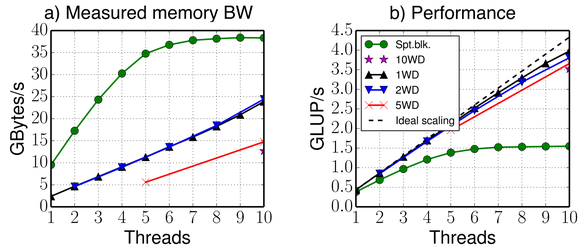


Fig. 4.2: 7-point constant-coefficient stencil thread scaling (grid: 980³)



BREAKING THE MEMORY WALL THROUGH ADVANCED TEMPORAL BLOCKING TECHNIQUES*

Fig. 5: 7-point constant-coefficient stencil thread scaling using grid size N=9803



c) Measured energy

		Power [W]			Energy [pJ/LUP]		
Method	Threads	CPU	DRAM	Total	CPU	DRAM	Total
Spt. Blk.	6	44.1	39.8	83.9	29.6	26.8	56.4
1WD	10	58.5	32.6	91.1	14.7	8.2	22.9
2WD	10	59.0	33.5	92.4	15.5	8.8	24.4
5WD	10	57.4	24.4	81.7	15.8	6.7	22.5
10WD	10	56.6	22.4	79.0	16.2	6.4	22.7

Fig. 6: 25-point variable-coefficient stencil thread scaling using grid size N=480³

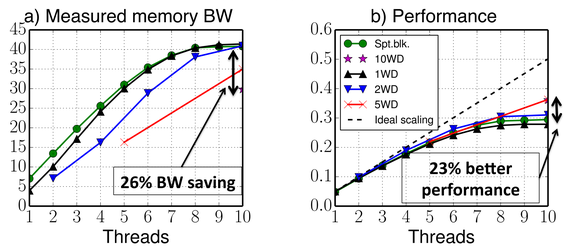
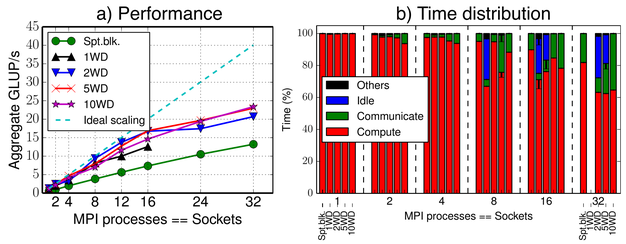


Fig. 7: 7-point variable-coefficient stencil distributed memory strong scaling using grid size N=768³ §



* Experiments are in double precision using 10-core Intel Ivy Bridge socket (Xeon E5-2660v2) @2.2 GHz

§ Nodes are connected by a full non-blocking fat-tree QDR InfiniBand network, with 2 sockets/node. Using one MPI process per socket

DIAMOND TILING + WAVEFRONT TEMPORAL BLOCKING

- Diamond tiling and domain decomposition across the Y-axis
- Wavefront temporal blocking along the Z-axis
  - 1-core wavefront [1]
  - Multi-core wavefront [2]
- No decomposition across the X-axis

Fig. 8: 1-core wavefront diamond [1]

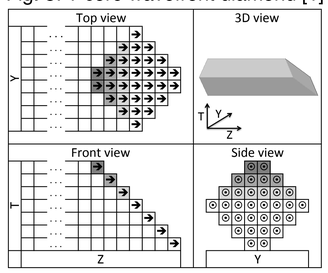
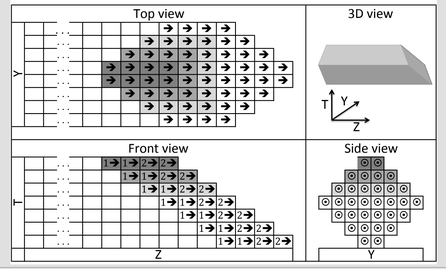


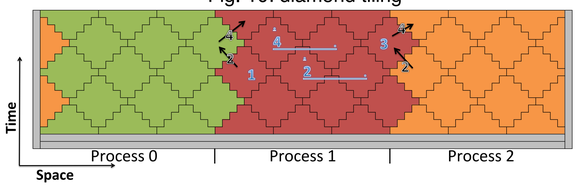
Fig. 9: APPROACH - Multi-core wavefront diamond "MWD"



DIAMOND TILING FOR SHARED AND DISTRIBUTED MEMORY SYSTEMS

- High data reuse, reducing memory accesses
- Provides independent space-time blocks to
  - reduce synchronization between threads and nodes
  - overlap computation with communication
- Assignment of diamond tiles to Thread Group (TG):
  - Within diamond: tight synchronization and large cache blocks
  - Across diamonds: loose synchronization between TGs
- Can provide efficient data migration for neighbor processes load-balancing
- Tessellation can reduce the overhead of handling the boundaries of subdomains, sockets, and heterogeneous processors

Fig. 10: diamond tiling

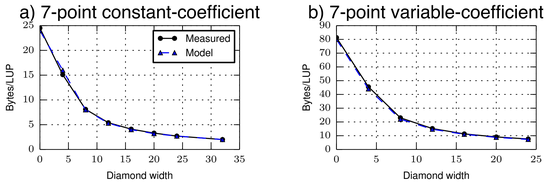


COMPUTATIONAL INTENSITY MODEL

D_W: Diamond width; N_b: Number of domain buffers; N_z: Size in z

$$LUPs = N_z \cdot D_w^2/2 \; [Diamond\ area]$$
$$Bytes = N_z \cdot ((2D_w - 2)[writes] + (N_b \cdot D_w)[reads]) \cdot 8$$
$$\frac{Bytes}{LUP} = \frac{16 \cdot ((2D_w - 2) + (N_b \cdot D_w))}{D_w^2}$$

Fig. 11: Computational intensity



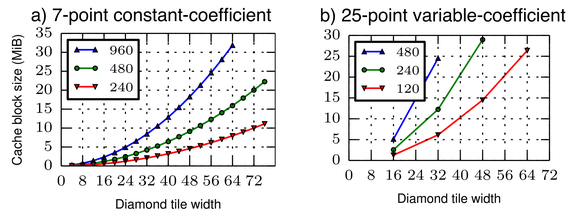
CACHE BLOCK SIZE REQUIREMENTS

N_{xb}: number of bytes in the x-axis; B_S: wavefront block size

W_W: Wavefront width; N_F: Number of frontlines;

$$W_w = D_w - 2 \cdot R + N_F + 1$$
$$B_s = N_{xb} \cdot \left[N_b \cdot D_w \cdot \left(\frac{D_w}{2} - R + N_F + 1\right) + 2 \cdot R \cdot (D_w + W_w)\right]$$

Fig. 12 Cache size requirements *per thread group*



SUMMARY OF SIGNIFICANCE

- **Large reduction in cache block size and memory bandwidth requirements**
- **Utilization of the shared cache between cores of modern processors**
- **Controllable tradeoff between memory per thread and frequency of synchronization**
- **Relaxed synchronization of MPI messages in distributed implementation**
- **Overlap of computations with communication in distributed implementations**

REFERENCES

[1] Strzodka, R., Shaheen, M., Pajak, D., & Seidel, H.-P. *Cache Accurate Time Skewing in Iterative Stencil Computations*. In ICPP '11 (pp. 571–581)

[2] Wellein, G., Hager, G., Zeiser T., Wittmann M., & Fehske H. *Efficient temporal blocking for stencil computations by multicore-aware wavefront parallelization.* In COMPSAC '09, vol. 1, (pp. 579-586).

Acknowledgements: Texas Advanced Computing Center (TACC) for hardware access
KAUST Extreme Computing Research Center for support of T. Malas

Paper available at arxiv.org/abs/1410.3060

Code available at github.com/tareqmalas/girih