Optimization of Geometric Multigrid for Emerging Multi- and Manycore Processors

Amik Singh
ParLab, UC Berkeley

I. INTRODUCTION

In the past decades, continued increases in clock frequencies have delivered exponential improvements in computer system performance. However, this trend came to an abrupt end a few years ago as power consumption became the principal rate limiting factor. As a result, power constraints are driving architectural designs towards ever-increasing numbers of cores, wide data parallelism, potential heterogenous acceleration, and a decreasing trend in per-core memory bandwidth. Understanding how to leverage these technologies in the context of demanding numerical algorithms is likely the most urgent challenge in high-end computing. In this work, we explore the optimization of geometric multigrid (MG) — one of the most important algorithms for computational scientists — on a variety of leading multi- and manycore architectural designs. Our primary contributions are:

We examine a broad variety of leading multicore platforms, including the Cray XE6, Intel Sandy Bridge and Nehalem-based Infiniband clusters, as well as NVIDIA's Fermi GPU.

We optimize and analyze all the required components within an entire multigrid V-cycle using a variable coefficient, Red-Black, Gauss-Seidel (GSRB) relaxation on these advanced platforms. This is a significantly more complex calculation than exploring just the stencil operation on a large grid.

We implement a number of effective optimizations geared towards bandwidth-constrained, wide-SIMD, manycore architectures including the application of wavefront to variable-coefficient, Gauss-Seidel, Red-Black (GSRB), SIMDization within the GSRB relaxation, and intelligent communication-avoiding techniques that reduce DRAM traffic.

We proxy the demanding characteristics of real simulations, where relatively small subdomains (32^3 or 64^3) are dictated by the larger application. This results in a broader set of performance challenges on manycore architectures compared to using larger subdomains.

Our performance analysis with respect to the underlying hardware features provides critical insight into the approaches and challenges of effective numerical code optimization for highly parallel, next-generation platforms.

II. MULTIGRID OVERVIEW

Multigrid (MG) methods provide a powerful technique to accelerate the convergence of iterative solvers for linear systems and are therefore used extensively in a variety of numerical simulations.

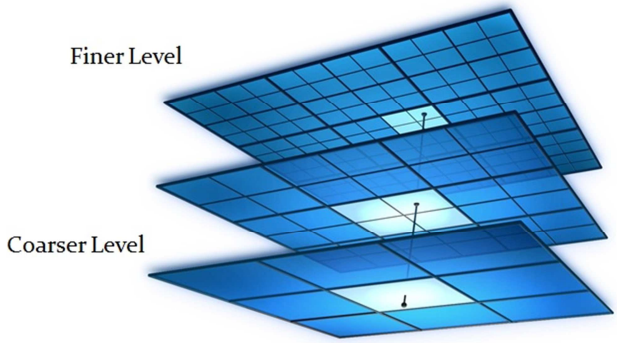

Fig 1 : Multigrid

Conventional iterative solvers operate on data at a single resolution and often require too many iterations to be viewed as computationally efficient. Multigrid simulations create a hierarchy of grid levels and use corrections of the solution from iterations on the coarser levels to improve the convergence rate of the solution at the finest level. Ideally, multigrid is

an O(N) algorithm; thus, performance optimization on our studied multigrid implementation can only yield constant speedups.

Figure 2 shows the three phases of the multigrid V-cycle for the solve of $Lu^h = f^h$. First, a series of smooths reduce the error while restrictions of the residual create progressively coarser grids. The

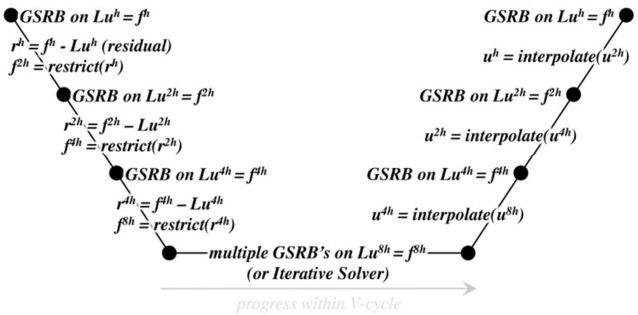

Fig 2 : V-cycle

smooth is a conventional relaxation such as Jacobi, successive over-relaxation (SOR), or Gauss-Seidel, Red-Black (GSRB) which we used in our study as it has superior convergence properties. The restriction of residual (f^h - Lu^h)is used to define the right-hand side at the next coarser grid. At each progressively coarser level, the correction (e.g. u^{2h}) is initialized to zero. Second, once coarsening stops (the grid size reaches one or terminated for performance), the algorithm switches to a bottom solver which can be as simple as applying multiple relaxes or as complicated as an algebraic multigrid or direct sparse linear solver. Finally, the coarsest correction is interpolated back up the V-cycle to progressively finer grids where it is smoothed.

Nominally, one expects an order of magnitude reduction in the residual per V-cycle. As each level performs O(1) operations per grid point and 1/8 the work of the finer grid, the overall computation is O(N) in the number of variables in u. The linear operator can be arbitrarily complex as dictated by the underlying physics, with a corresponding increase in runtime to perform the smooth computation.

III. RELATED WORK

Throughout this paper, we leverage the 3C's taxonomy when referring to cache misses [14]. In the past, operations on large structured grids could easily be bound by capacity misses, leading to a variety of studies on blocking and tiling optimizations [9], [10], [16], [21], [22], [27], [28]. However, a number of factors have made such approaches progressively obsolete on modern platforms. On-chip caches have grown by orders of magnitude and are increasingly able to capture sufficient locality, particularly for the fixed box sizes associated with memory-constrained, multi-variable or other complex MG methods. The rapid increase in on-chip parallelism has also quickly out-striped available DRAM bandwidth. Thus, in recent years, numerous efforts have focused on increasing temporal locality by fusing multiple stencil sweeps through techniques like cache oblivious, time skewing, or wavefront [8], [11], [12], [17], [19], [23], [26], [29]–[31]. Many of these efforts examined 2D or constant-coefficient problems — features rarely seen in real-world applications. Chan et al. explored how, using an auto-tuned approach, one could restructure the MG V-cycle to improve time-to-solution in the context of a 2D, constant-coefficient Laplacian [5]. This approach is orthogonal to our implemented optimizations and their technique could be incorporated in future work. Studies have explored the performance of algebraic multi-grid on GPUs [1], [2], while Sturmer et al. examined geometric multigrid [24]. Perhaps the most closely related work is that performed in Treibig's, which implements a 2D GSRB on SIMD architectures by separating and reordering the red and black elements [25], additionally a 3D multigrid on an IA-64 (Itanium) is implemented via temporal blocking. Our work expands on these efforts by providing a unique set of optimization strategies for multi- and manycore architectures.

IV. EXPERIMENTAL SETUP

A. Evaluated Platforms

We use the following systems in all our experiments. Their key characteristics are summarized in Table I.

Cray XE6 "Hopper":

Hopper is a Cray XE6 MPP at NERSC built from 6384 compute nodes each consisting of two 2.1 GHz 12-core Opteron (MagnyCours) processors [15]. In reality, each Opteron socket is comprised of two 6-core chips each with two DDR3-1333 memory controllers. Effectively, the compute nodes are comprised of four (non-uniform memory access) NUMA nodes, each providing about 12 GB/s of STREAM [18] bandwidth. Each core uses 2-way SSE3 SIMD and includes both a 64KB L1 and a 512KB L2 cache, while each socket includes a 6MB L3 cache with 1MB reserved for the probe filter. The compute nodes are connected through the Gemini network into a 3D torus.

Nehalem-Infiniband Cluster "Carver":

The Carver cluster at NERSC is built from 1202 compute nodes mostly consisting of two 2.66 GHz, quad-core Intel Xeon X5550 processors [4]. Thus, each compute node consists of two NUMA nodes. Each quad-core Nehalem (NHM) socket includes an 8 MB L3 cache and three DDR3 memory controllers providing about 18 GB/s of STREAM bandwidth. Each core implements the 2-way SSSE3 SIMD instruction set and includes both a 32KB L1 and a 256KB L2 cache. HyperThreading is disabled on Carver. The compute nodes are connected through the 4X QDR Infiniband network arranged into local fat trees and a global 2D mesh.

Sandy Bridge-Infiniband Cluster "Gordon":

The Gordon cluster at the San Diego Supercomputing Center is comprised of 1024 compute nodes each with two 2.6 GHz, 8-core Intel Xeon E5 2670 processors [13]. Each 8-core Sandy Bridge (SNBe) processor includes a 20 MB L3 cache and four DDR3-1333 memory controllers providing about 35 GB/s of STREAM bandwidth. Each core implements the 4-way AVX SIMD instruction set and includes both a 32KB L1 and a 256KB L2 cache. This provides Gordon with four times the peak performance and twice the sustained bandwidth as Carver. HyperThreading is disabled on Gordon. The compute nodes are connected through the 4X QDR Infiniband network with switches arranged into a 4x4x4 network.

Core Arch.	AMD Opteron	Intel NHM	Intel SNBe	NVIDIA Fermi
Clock (GHz)	2.10	2.66	2.60	1.30
DP GFlop/s	8.40	10.66	20.80	83.2
Memory-Parallelism	HW-prefetch	HW-prefetch	HW-prefetch	multi-threading
Node Arch.	**Cray XE6**	**Xeon X5550**	**Xeon E5 2670**	**Tesla M2090**
Cores/chip	6	4	8	16[1]
LL$/chip	5 MB	8 MB	20 MB	768 KB
Chips/node	4	2	2	1
DP GFlop/s	201.6	85.33	332.8	665.6
STREAM	49.4 GB/s	38 GB/s	70 GB/s	120 GB/s
Memory	32 GB	24 GB	64 GB	6 GB

Table 1 : Different machines used

NVIDIA M2090-accelerated Node:

The M2090 Fermi GPU includes 512 scalar "CUDA cores" running at 1.30 GHz and grouped into sixteen SIMT-based streaming multiprocessors (SM). This provides a peak double-precision floating-point capability of 665 GFlop/s. Each SM includes a 128 KB register file and a 64 KB SRAM that can be partitioned between cache and "shared" memory in a 3:1 ratio. Although the GPU has a raw pin bandwidth of 177 GB/s to its on-board 6 GB of GDDR5 DRAM, the measured bandwidth with ECC enabled is about 120 GB/s. ECC is enabled in all our experiments.

B. Problem Specification

A key goal of our work is to analyze the computational challenges of multigrid in the context of multi- and manycore, optimization, and programming model. We therefore construct a compact multigrid solver benchmark that creates a global 3D domain partitioned into subdomains sized to proxy those found in real MG applications. All subdomains (whether on the same node or not) must explicitly exchange ghost zones with their neighboring subdomains, ensuring an effective communication proxy of MG codes.

We use a double-precision, finite volume discretization of the variable-coefficient operator $L = a\vec{\alpha}I + b\nabla\vec{\beta}\nabla$ with periodic boundary

conditions as the linear operator within our test problem. Variable-coefficient is an essential (yet particularly challenging) facet as most real-world applications demand it. The right-hand side (f) for our benchmarking is sin(∏x)sin(∏y)sin(∏z) on the [0,1] cubical domain. The u, f, and α are cell-centered data, while the β's are face centered.

To enable direct time-to-solution comparisons of different node architectures, we fix the problem size to a 256^3 discretization on all platforms. This (relatively small) grid size in conjunction with partitioning into subdomains, the variable-coefficient nature of our computation, and buffers required for exchanging data, consumes more than 2.5GB. Although this is a small percentage of the XE6 node capacity, it is a large fraction of the memory available to most GPUs. Thus, our baseline for node comparison is the performance of the 4-chip XE6, the 2-chip Intel Xeons and the GPU accelerator solving one 256^3 problem.

To allow for uniform benchmarking across the platforms, we structure a truncated V-cycle where restriction stops at the coarsest level of 4^3. We fix the number of V-cycles at 10 and perform two relaxations at each level down the V-cycle, 24 relaxes at the bottom, and two relaxations at each level up the V-cycle.

Our relaxation scheme uses Gauss-Seidel Red-Black (GSRB) which offers superior convergence compared to other methods. It consists of two grid smooths per relax each updating one color at a time, for a total of eight smooths per subdomain per level per V-cycle. The pseudocode for the resultant inner operation is shown in Figure 3. Here, neither the

```
laplacian[i,j,k] = a*alpha[I,j,k]*phi[i,j,k] - b*h2inv*(
  beta_i[i+1,j,k] * ( phi[i+1,j,k] - phi[i,j,k]   ) -
  beta_i[i,j,k]   * ( phi[i,j,k]   - phi[i-1,j,k] ) +
  beta_j[i,j+1,k] * ( phi[i,j+1,k] - phi[i,j,k]   ) -
  beta_j[i,j,k]   * ( phi[i,j,k]   - phi[i,j-1,k] ) +
  beta_k[i,j,k+1] * ( phi[i,j,k+1] - phi[i,j,k]   ) -
  beta_k[i,j,k]   * ( phi[i,j,k]   - phi[i,j,k-1] )
)

phi[i,j,k] = phi[i,j,k] -
  lambda[i,j,k] * ( laplacian[i,j,k] - rhs[i,j,k] )
```

Fig 3 : Smooth Pseudo-code

Laplacian nor the Helmholtz of a grid is ever actually constructed, rather all these operations are fused together into one GSRB relax. A similar calculation is used for calculating the residual. Nominally, these operators require a one element deep ghost zone constructed from neighboring subdomain data. However, in order to leverage communication aggregation and communication avoiding techniques, we also explore a 4-deep ghost zone that enables optimization at the expense of redundant computation. Although the CPU code allows for residual correction form, it was not employed in our experiments; observations show its use incurs a negligible impact on performance.

The data structure within level for a subdomain is a list of equally-sized grids (arrays) representing the correction, right-hand side, residual, and coefficients each stored in a separate array. Our implementations ensure that the core data structures remain relatively unchanged with optimization. Although it has been shown that separation of red and black points into separate arrays can facilitate SIMDization [25], our benchmark forbids such optimization as they lack generality and challenge other phases. As data movement is the fundamental performance limiter, separating red and black will provide no benefit as the same volume of data transfer is still required.

C. Reference (Baseline) Implementation

This benchmark builds the MG solver by allocating and initializing the requisite data structures, forming and communicating restrictions of the coefficients, and performing multiple solves. Given that each NUMA node is assigned one MPI process, the single-node CPU experiments use 4 (XE6) or 2 (NHM, SNBe) MPI processes which collectively solve the 256^3 problem. OpenMP parallelization is applied to the list of subdomains owned by a given process. When using threads for concurrency, each thread operates independently on one subdomain at a time. For example, on Gordon with 64 subdomains per node, each of the 16 OpenMP threads will be responsible for 4 subdomains.

Conversely, the GPU required substantial modifications to the baseline CPU code to realize a reasonable CUDA implementation. First, the

baseline C code's grid creation routines were modified to allocate data directly in device memory on the GPU. This style of programming (as opposed to an off-load model) obviates PCIe transfers except to initiate kernels and communicate ghost zones to neighboring processes. Second, the core routines were modified from simple OpenMP into appropriate CUDA implementations. The CUDA version of Smooth uses three dimensions of nested parallelism (subdomains, Dim i and Dim j). Each 18*18 thread block within a subdomain streams an i,j patch through k using shared memory to capture temporal and inter-thread reuse of the correction, beta i, and beta j. Finally, subdomain ghost zone exchanges were maximally parallelized and performed explicitly in device memory. The GPU code was compiled with -dlcm=cg to compensate for the misaligned uncoalesced nature of our stencils.

V. PERFORMANCE CHALLENGES AND EXPECTATIONS

In our implementation, there are five principal functions at each level: smooth, residual, restriction, interpolation, and exchange. Smooth is called eight times, Exchange nine, and the others once. In the reference implementation, data dependencies mandate each function be called in sequence. Descending through the V-cycle, the amount of parallelism in smooth, residual, restriction, and interpolation decrease by a factor of eight, while working sets decrease by a factor of four. This creates an interesting and challenging interplay between intra- and inter-box parallelism.

Smooth: Nominally, smooth dominates the run time. This function performs the GSRB relax (stencil) on every other point (one color), and writes the resultant correction back to DRAM. For a 64^3 subdomain with a 1-deep ghost zone, this corresponds to a working set of about 340KB (10 planes from 7 different arrays), the movement of 2.3M doubles ($8*66^3$), and execution of 3.3M floating-point operations (25 flops on every other point). As a subsequent call to smooth is not possible without communication, its flop:byte ratio of less than 0.18 results in performance heavily bound by DRAM bandwidth. In fact, with 50 GB/s of STREAM bandwidth,Smooth will consume at least 1.88 seconds at the finest grid. Although GSRB is particularly challenging to SIMDize, its memory-bound nature avoids any performance loss without it.

Restriction: Restriction reads in the residual and averages eight neighboring cells. This produces a grid (f^{2h}) nominally8x smaller — a 64^3 grid is restricted down to 323 (plus a ghost zone). Such a code has a flop:byte ratio of just under 0.09 and is thus likely to be heavily bandwidth-bound.

Interpolation: Interpolation is the mirror image of restriction: each element of u^{2h} is used to increment eight elements of u^h. It typically moves almost twice as much data as restrict and has a flop:byte ratio less than 0.053.

Exchange Boundaries: This function contains three loop nests. First, the (non-ghost zone) surface of each 3D subdomain is extracted and backed into 26 surface buffers representing the 26 possible neighboring sub-domains. Second, buffers are exchanged between subdomains into a set of 26 ghost zone buffers.

VI. OPTIMIZATION

A. More Ghost Zones

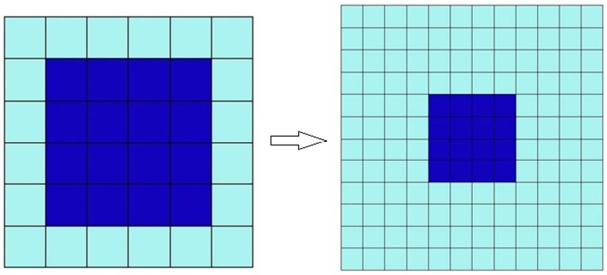

Fig 4 : Ghost Zones

To improve communication performance, it is well known that transferring a deeper ghost zone can be used to reduce the number of communication phases at the expense of redundant computation required to produce a bit-identical result. That is, rather than four-rounds of smoothing each 64^3 grid,

four smooths can be performed in one pass on a 72^3 (64+4*2) grid. One deep ghost zones and four deep ghost zones are shown in figure 4.

B. Wavefront Approach

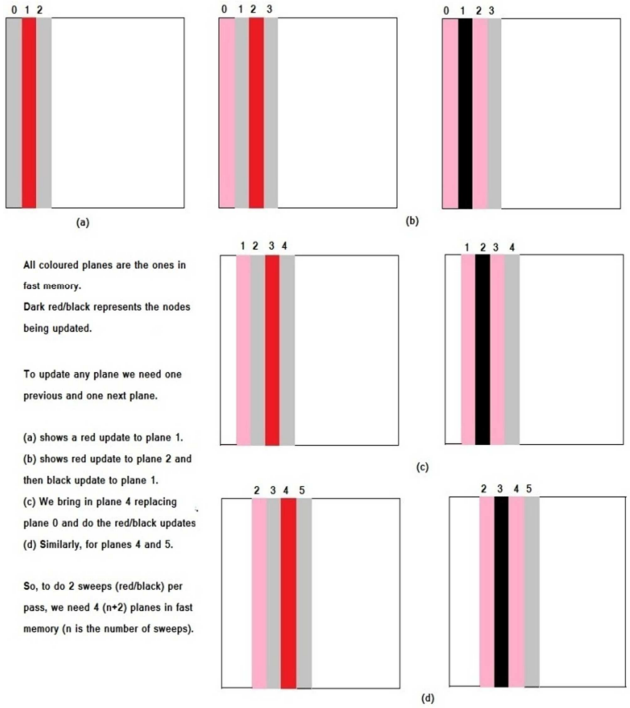

Fig 5 : Wavefront

To minimize the working set and enable a streaming model, we implement a wavefront approach [31] to the Gauss-Seidel, Red-Black relax operation. The working of wavefront approach is as shown in figure 5. The planes areloaded into fast memory and only the last plane (loaded first) is dropped to load a new plane in the wave.

C. Avoiding inter-thread block communication

The GPUs have a very small shared memory (48 KB in our case). If we try to apply the wavefront approach on a whole sub-domain we won't be able to use the shared memory. The size of one single plane at the finest level is 72*72*8(size of double), i.e. 40.5 KB, whereas we need many planes for the actual realization of wavefront approach. So, to tackle this problem we divided our subdomain further into smaller cuboids to make sure data resides in shared memory. But for the realization of this approach, we had to load too much redundant data to save on inter-thread block communication because CUDA does not provide any mechanism for inter thread block coherency of global writes. Figure 6 shows assignment of different small planes to the thread blocks (at the finest level). Thread block zero gets a sub-cuboid going from 0-16 in i and j directions and extends upto the full 72 elements in k direction.

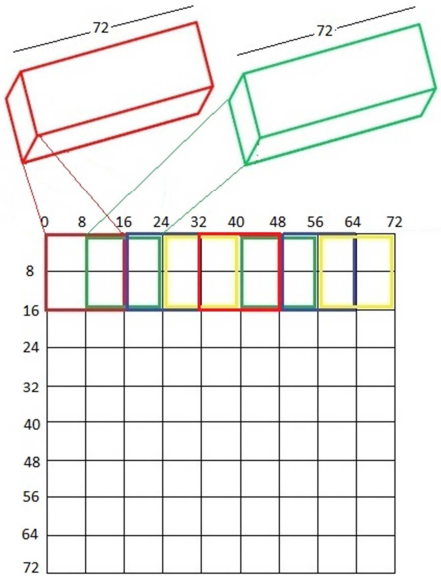

Fig 6 : Avoiding inter-thread block communication on GPUs

VII. RESULTS

A. Performance on the Finest Grids

Figure 7 shows a breakdown of the total time spent on the finest grids before and after optimization. Note that the multigrid time is expected to be dominated by the execution at the finest resolution. We got speedup of 1.6x on the Nehalem cluster however, GPU performance at the finest level actually decreases by 1.6x. Observe that the overhead of Smooth for DRAM communication-avoiding accelerated all CPUs, but actually impaired GPU performance. Conversely, we see that aggregating ghost zone exchanges showed practically no performance benefit on the CPUs,

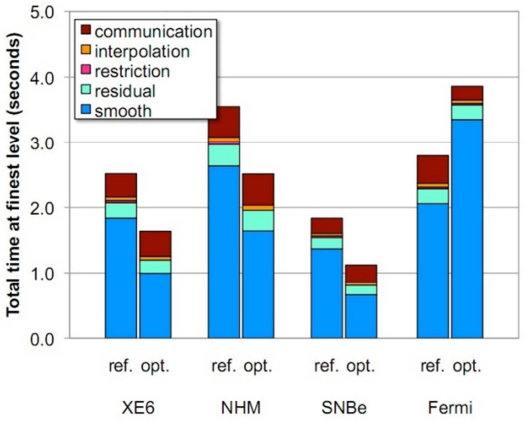

Fig 7 : Breakdown of time for different machines

while significantly reducing communication overheads on GPU by 55%. In order to understand these seemingly contradictory performance effects on Smooth for CPUs and GPUs, we construct a simple data movement and bandwidth model for each architecture. Table II shows that the reference

	System	XE6	NHM	SNBe	Fermi
	STREAM (GB/s)[3]	49.4	38	70	120
Reference	Time (seconds)	1.84	2.63	1.37	2.06
	Data Moved (10^9B)	94.2	94.2	94.2	136[1]
	Bandwidth (GB/s)	51.2	35.8	68.7	66.3
Comm. Avoiding	Time (seconds)	1.00	1.64	0.66	3.35
	Data Moved (10^9B)	30.6	30.6	30.6	192[2]
	Bandwidth (GB/s)	30.7	18.6	46.4	57.3
	Speedup	1.8×	1.6×	2.1×	0.6×

Table II : reference vs communication-avoiding

implementation of Smooth on all CPUs attain an extremely high percentage of STREAM bandwidth. Data movement estimates for the GPU based on 32 byte cache line show that the baseline GPU implementation moves 44% more data than the basic CPU versions. This is because on-chip memory heavily constrains thread block surface:volume ratios and adjacent thread blocks cannot exploit shared locality. The lack of extensive array padding, alignment, and incomplete memory coalescing on the GPU may lead to suboptimal bandwidth.

B. Overall Speedup for the MG Solver

Figure 8 presents the overall performance (higher is better) of the multigrid solver before and after optimization, normalized to reference XE6

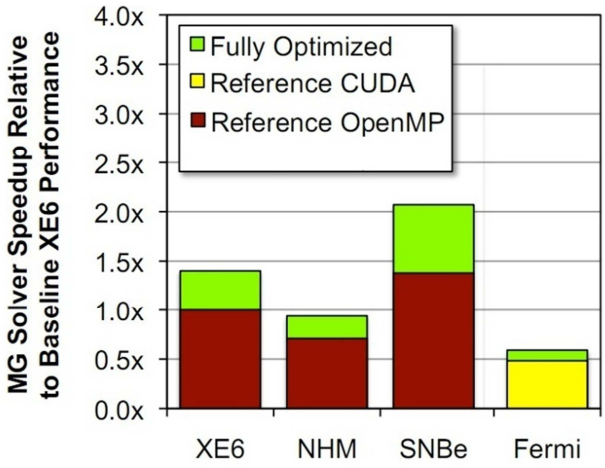

Fig 8 : Normalized solver times

performance. With full tuning (communication-avoiding, SIMD, prefetch, etc.), results show see an 1.5x increase in solver performance on SNBe. The high working sets associated with the smooth operator limited the GPUs ability to form sufficient locality on chip, thus wasting much of its bandwidth potential on redundant data movement. Moreover, without sufficient on-chip memory, the GPU was unable to realize the potential of communication-avoiding.

VIII. CONCLUSIONS

Data movement is the primary impediment to high performance on existing and emerging systems. To address this limitation there are fundamentally two paths forward: algorithmic and architectural. The former attempts to restructure computation to minimize the total data movements, while the latter leverages technological advances to maximize data transfer bandwidth. We explore both of these avenues in the context of 3D geometric multigrid with a non-trivial operator — a demanding algorithm widely-used by the computational community. To reduce data movement, we use communication-avoiding techniques to create larger (but less frequent) messages between subdomains, while

simultaneously increasing the potential temporal reuse within the GSRB relaxation. Additionally, we evaluate performance on NVIDIA's Tesla M2090. Result show that our wavefront approach can dramatically improve Smooth runtime on the finer grids despite the redundant work. Effectively implementing this approach poses two significant challenges: how to productively decouple DRAM loads from the in-cache computation on the wavefront, and how to efficiently express sufficient parallelism without sacrificing sequential locality. On CPUs the hardware prefetchers designed to decouple memory access through speculative loads are hindered by the lack of sequential locality — an artifact of extreme thread-level parallelization. On highly-multithreaded architectures like the GPU, this is not an issue and parallelization in the unit-stride is feasible. However, whereas the CPUs have sufficient on-chip memory and efficient intra-core coalescing of memory transactions, the GPUs limited on-chip memories hamper realization of communication-avoiding benefits.

IX. FUTURE WORK

Future work will examine techniques to improve productivity and portability in collaboratively threaded environments as well as high-order and multi-variable operators. We observe that on-node data marshaling for MPI is a substantial impediment to communication performance, particularly at the bottom of the V-cycle. Exploration of techniques to mitigate this are increasingly important for large-scale MPI concurrency. Finally, we will explore the use of communication-avoiding techniques in matrix-free Krylov Subspace methods like BiCGstab for fast bottom solves on supercomputers.

ACKNOWLEDGEMENT

I am really thankful to Sam Williams, Brian Van Straalen and Leonid Oliker for their support and guidance in the project. I did work on GPUs, all the rest of work on different architectures was done by Sam Williams.

REFERENCES

[1] N. Bell, S. Dalton, and L. Olson. Exposing fine-grained parallelism in algebraic multigrid methods. NVIDIA Technical Report NVR-2011-002, NVIDIA Corporation, June 2011.

[2] J. Bolz, I. Farmer, E. Grinspun, and P. Schr ¨ ooder. Sparse matrix solvers on the gpu: conjugate gradients and multigrid. *ACM Trans. Graph.,* 22(3):917–924, July 2003.

[3] BoxLib. https://ccse.lbl.gov/BoxLib.

[4] Carver website. http://www.nersc.gov/users/computational-systems/carver.

[5] C. Chan, J. Ansel, Y. L. Wong, S. Amarasinghe, and A. Edelman. Autotuning multigrid with petabricks. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis,* SC '09, pages 5:1–5:12, New York, NY, USA, 2009. ACM.

[6] CHOMBO. http://chombo.lbl.gov.

[7] A. Danalis, G. Marin, C. McCurdy, J. Meredith, P. Roth, K. Spafford, V. Tipparaju, and J. Vetter. The scalable heterogeneous computing (SHOC) benchmark suite. In *Proc. 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU '10),* pages 63–74. ACM, 2010.

[8] K. Datta, S. Kamil, S. Williams, L. Oliker, J. Shalf, and K. Yelick. Optimization and performance modeling of stencil computations on modern microprocessors. *SIAM Review*, 51(1):129–159, 2009.

[9] 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 *Proc. 2008 ACM/IEEE Conf. on Supercomputing (SC 2008),* pages 1–12, 2008.

[10] C. C. Douglas, J. Hu, M. Kowarschik, U. Rde, and C. Weiss. Cache optimization for structured and unstructured grid multigrid. *Elect. Trans. Numer. Anal,* 10:21–40, 2000.

[11] M. Frigo and V. Strumpen. Evaluation of cache-based superscalar and cacheless vector architectures for scientific computations. In *Proc. Of the 19th ACM International Conference on Supercomputing (ICS05),* Boston, MA, 2005.

[12] P. Ghysels, P. Kosiewicz, and W. Vanroose. Improving the arithmetic intensity of multigrid with the help of polynomial smoothers. *Numerical Linear Algebra with Applications,* 19(2):253–267, 2012.

[13] Gordon website. http://www.sdsc.edu/us/resources/gordon.

[14] M. D. Hill and A. J. Smith. Evaluating Associativity in CPU Caches. *IEEE Trans. Comput.,* 38(12):1612–1630, 1989.

[15] Hopper website. http://www.nersc.gov/users/computational-systems/hopper.

[16] M. Kowarschik and C. Wei. Dimepack - a cache-optimized multigrid library. In *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001),* volume I, pages 425–430. CSREA, CSREA Press, 2001.

[17] J. McCalpin and D. Wonnacott. Time skewing: A value-based approach to optimizing for memory locality. Technical Report DCS-TR-379, Department of Computer Science, Rugers University, 1999.

[18] J. D. McCalpin. STREAM: Sustainable Memory Bandwidth in High Performance Computers. http://www.cs.virginia.edu/stream/.

[19] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey. 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, SC '10,* pages 1–13, Washington, DC, USA, 2010. IEEE Computer Society.

[20] B. Palmer and J. Nieplocha. Efficient algorithms for ghost cell updates on two classes of MPP architectures. In *Proc. PDCS International Conference on Parallel and Distributed Computing Systems,* pages 192–197, 2002.

[21] G. Rivera and C. Tseng. Tiling optimizations for 3D scientific computations. In *Proceedings of SC'00,* Dallas, TX, November 2000. Supercomputing 2000.

[22] S. Sellappa and S. Chatterjee. Cache-efficient multigrid algorithms. *International Journal of High Performance Computing Applications,* 18(1):115–133, 2004.

[23] Y. Song and Z. Li. New tiling techniques to improve cache temporal locality. *In Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation,* Atlanta, GA, 1999.

[24] M. Sturmer, H. Kostler, and U. Rude. How to optimize geometric multigrid methods on GPUS. In *Proc. of the 15th Copper Mountain Conference on Multigrid Methods,* Copper Mountain, CO, March, 2011.

[25] J. Treibig. *Efficiency improvements of iterative numerical algorithms on modern architectures*. PhD thesis, 2008.

[26] G. Wellein, G. Hager, T. Zeiser, M. Wittmann, and H. Fehske. Efficient temporal blocking for stencil computations by multicore-aware wavefront parallelization. In *International Computer Software and Applications Conference,* pages 579–586, 2009.

[27] S. Williams, J. Carter, L. Oliker, J. Shalf, and K. Yelick. Lattice Boltzmann simulation optimization on leading multicore platforms. *In Interational Conference on Parallel and Distributed Computing Systems (IPDPS),* Miami, Florida, 2008.

[28] S. Williams, L. Oliker, J. Carter, and J. Shalf. Extracting ultra-scale lattice boltzmann performance via hierarchical and distributed autotuning. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11,* pages 55:1–55:12, New York, NY, USA, 2011. ACM.

[29] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick. The potential of the Cell processor for scientific computing. In *Proceedings of the 3rd Conference on Computing Frontiers,* New York, NY, USA, 2006.

[30] D. Wonnacott. Using time skewing to eliminate idle time due to memory bandwidth and network limitations. In *IPDPS:Interational Conference on Parallel and Distributed Computing Systems,* Cancun, Mexico, 2000.

[31] T. Zeiser, G. Wellein, A. Nitsure, K. Iglberger, U. Rude, and G. Hager. Introducing a parallel cache oblivious blocking approach for the lattice Boltzmann method. *Progress in Computational Fluid Dynamics,* 8, 2008.