HPC Performance Improvements Through Innovative Architecture

Project 38 Technical Report

October 2019

Contributors

David Mountain
John Shalf
Andrew Chien
Raymond Bair
Arun Rodrigues
Eric Cheng
Yuan Zeng
Xiaochen Guo
Dilip Vasudevan
Maya Gokhale
George Michelogiannakis
Anastasiia Butko
David Donofrio
Adolfy Hoisie
Kristopher Keipert
Hal Finkel
Antonino Tumeo
James Ang
Lingda Li
Noel Wheeler
Katherine Yelick
Marquita Ellis
George Fann
Scott Lloyd
Mark Raugas
Simon Hammond

Executive Summary

NSA, the DOE Office of Science, and the DOE National Nuclear Security Administration (NNSA) (in this document, these latter two organizations are referred to as "DOE") recognize the imperative to develop new mechanisms for engagement with the vendor community, particularly on architectural innovations with strategic value to USG HPC. Project 38 is an interagency collaboration between NSA and the DOE, focusing on a set of vendor-agnostic architectural explorations. These explorations are intended to accomplish the following:

Near-term goal: Quantify the performance value and identify the potential costs of specific architectural concepts against a limited, focused set of applications of interest to both the DOE and NSA.

Long-term goal: Develop an enduring capability for DOE and NSA to jointly explore architectural innovations and quantify their value.

Project 38 has made significant progress in meeting its near-term goal, and is beginning to share initial exploration results with the wider HPC community, both industry and academia.

Table of Contents

1 Motivation

The technology ecosystem for acquiring HPC systems of interest to DOE and NSA is becoming less favorable. The DOE and NSA have mission-critical application requirements that will not be satisfied by future HPC systems. The commercial HPC market is shrinking, focused on a relatively specific balance of computation and communication resources, and increasingly driven by a narrow class of deep-learning applications. The relatively small size of this market means the architectural diversity of commercial offerings are low, even though architectural specialization has been shown to provide performance benefits. In addition, the HPC ecosystem is under significant pressure from data center customers (Google, Facebook, etc.), which skews the computing market in ways that are not advantageous to NSA and DOE HPC needs.

USG HPC must make substantial investments to either acquire or build systems as they have mission drivers for architectural features that are not aligned with commercial drivers. These leading-edge HPC systems also tend to be more difficult for application developers and customers to port their codes and use productively.

The DOE has traditionally followed a "set requirements, solicit bids, deploy system" process, often with substantial Non-Recurring Engineering investments to influence architectural designs or add software features which otherwise would not be available in the vendors' regular product offerings. In addition, the prior DOE FastForward, DesignForward and PathForward efforts (starting in FY2012) modified this approach to influence vendors to pursue HPC-driven technology developments. In the current landscape where specialized hardware is increasingly important, USG HPC experts need to engage earlier and more directly in the design process to expand the scope of influence the DOE and NSA can have on the system design. If we do nothing, the narrow set of offerings is likely to continue - or even become worse due to the HPC ecosystem pressures noted earlier, and the looming end of Moore's Law lithography-based improvements.

Purpose-built architectures are a promising approach to improve this situation. While joint explorations/collaborations on purpose-built architectures can be effective, they require a new approach to exploring, designing and developing HPC systems. A successful DOE-NSA collaboration on architectural explorations for mission-critical applications would provide important technical information to increase the effectiveness of USG investments in HPC.

Project 38 will create a new way for USG HPC to engage with industry. If successful, it will broaden the industry players and develop more diversity in industry roadmaps. Purpose-built systems will become more common, widening the range of architectures and technologies available in general-purpose HPC systems, thereby reducing system development and acquisition costs.

This approach is a fundamentally new way of collaborating, which provides the USG with maximum ability to influence the future direction of HPC systems, components, and technologies. Both NSA and DOE recognize the potential value of these collaborations and are highly motivated to make them successful.

2 Business Plan (potential paths to system level impact)

The first implementation path is 'aggressive vendor partnerships.' In this approach, the USG collaborates with vendor research teams to explore architectural concepts that vendors might not otherwise consider due to market pressure. These concepts should be significant advances, and will presumably be more aggressive than existing industry efforts (either the innovations themselves or the timeline for introducing them). This approach builds on the architecture foundation of the vendors' respective technology roadmaps. As such, this approach focuses on closing gaps and accelerating technology roadmaps.

The second is 'innovative USG design.' In this approach, the USG becomes much more purposeful in the overall architecture and design, incorporating features and functions specifically targeted to support USG applications. This approach would enable the USG to develop purpose-built, advanced architectures that define new, perhaps disruptive, hardware designs that are not built on pre-existing product roadmaps.

Together, these paths enable the USG to evaluate two points on the risk-reward spectrum. The first implementation path has much more vendor collaboration, which presumably lowers the risk, and possibly the expense, of a successful HPC system delivery as an outcome. The improvement in the performance of the system on critical applications may be lower than a completely purpose-built architecture. The second path entails more risk (and possibly expense) for the USG; the performance improvement may be quite significant – purpose built architectures can show orders of magnitude improvement over more general-purpose designs.

Project 38 is currently focused on sharing its initial results with the wider HPC community.

3 Exemplar Applications

3.1 FFT

1D FFT based on the HPC Challenge benchmarks [HCC06]. The benchmark selected is a 32-bit floating point complex out-of-place transform with 1Gigabyte transform size (so as to be outside of the limits of on-chip memory). The transforms were computed using the FFTW library [FTW05] on CPU and Intel Xeon Phi devices, and using the cuFFT library [CUF19] on GPUs.

3.2 Kripke (radiation transport)

KRIPKE [KRIPKE15] was developed to study the performance characteristics of data layouts, programming models, and sweep algorithms. KRIPKE is designed to support different in-memory data layouts, and allows work to be grouped into sets in order to expose more on-node parallelism. Different data layouts change the way in which software is implemented, how that software is compiled for a given architecture, and how that generated code eventually performs on a given architecture. This benchmark emphasizes word-granularity gather/scatter requirements for the memory subsystem.

3.3 MTC (Quantum Simulations)

Mini-tensor contraction (MTC) is a microbenchmark kernel representing the contraction, scatter/gather and transpose operations for high dimensional tensor operations with irregular index patterns. These are some of the limiting factors in achieving high performance for current

architectures in quantum simulations in chemistry (NWChem, MADNESS), material science (LSMS, QMCPACK), and nuclear structures (NUCCOR). One example which chemists have developed over the years is the Tensor Contraction Engine (https://www.csc.lsu.edu/~gb/TCE/) [TCE05], a source to source translator, in an attempt to solve this problem. The MTC tensor kernel normally has high bandwidth waste on fetching non-contiguous data, contractions of 4D and 6D tensors, indices which are dynamically changing, typical of many current applications. It emphasizes word-granularity gather/scatter, irregular memory access, and data transpose requirements for the memory subsystems.

3.4 Stencil/PDE-solver

Stencil codes (e.g., [HPGMG14, HOMME07]) are a class of iterative kernels which update the array elements according to a fixed pattern. It is common in high-performance computer simulations that model the physical phenomena (such as fluid dynamics and heat diffusion) through finite difference or finite element methods. The stencil kernels normally have a high ratio of memory accesses to calculations and thus are memory-bound. When the stencil dataset is partitioned across different cores, exchange between the halo regions may incur significant communication overhead. This benchmark emphasizes the word-granularity scratchpad's benefit on reducing the overheads for halo-exchange and eliminating coherence overheads.

3.5 HipMER (bioinformatics)

HipMer [HIPMER17] is a parallel genome assembler pipeline that has been shown to scale to massive concurrencies. The benchmark version, called Meraculous, captures one stage of the HipMer computation and was developed as part of the NERSC-9 benchmark suite. Meraculous emphasizes system-wide random memory writes to construct a global hash-table of genomic fragments in order to find identical fragments which can be used to seed searches for longer (imperfect) matches. The application uses random access into the hash table memory, which differs from typical gather-scatter workloads in that the random access are individual rather than grouped. The hash table inserts and lookups are performed across nodes, so the benchmark performs small remote reads and writes (64bits or less) with nearly random communicating partners. Remote atomic operations like compare-and-swap are also key. One of the key local computations within HipMer's later stages is sequence alignment, identifying the match between two strings with the minimum number of insertions, deletions or replacements.

3.6 Sparse Matrix Trisolve (SpMTrsv) / SuperLU

Solving system of equations of the form Ax=b where A is a sparse triangular matrix, is required after the factorization phase in the direct methods of solving systems of linear equation (e.g., [SUPERLU05]). As a result, the triangular solver is an important computational kernel in many applications. On-node parallelism is increasingly important. Multiple approaches underlying multithreaded triangular solvers exist, but the overheads of direct execution of the dependency graph for the trisolve is a serious inhibitor to scalable performance.

3.7 HPGMG

HPGMG (High-performance Geometric Multigrid) [HPGMG14] contains both a finite element and a finite volume implementation of full multigrid algorithms. While HPGMG stresses floating point computation and the memory subsystem, it is primarily designed to stress system

interconnects. The inability to hide communication limits effective scaling in large-scale systems. This benchmark emphasizes the need to reduce communication traffic across the memory subsystem and across nodes

3.8 References

[CUF19] cuFFT v.10.1 official documentation. https://docs.nvidia.com/cuda/cufft/index.html. Accessed: 2019-07-24.

[FTW05] Matteo Frigo and Steven G. Johnson, The Design and Implementation of FFTW3, in Proceedings of IEEE, 2005

[HCC06] Piotr Luszczek, David Bailey, Jack Dongarra, Jeremy Kepner, Robert Lucas, Rolf Rabenseifner, and Daisuke Takahashi, The HPC Challenge (HPCC) Benchmark Suite, in Proceedings of the 2006 ACM/IEEE conference on Supercomputing, November 2006, Tampa, Florida.

[HOMME07] Taylor, M. A., J. Edwards, S. Thomas, and R. D. Nair, "A mass and energy conserving spectral element atmospheric dynamical core on the cubed-sphere grid," Journal of Physics Conference Series, 2007.

[HIPMER17] Georganas, et al., "MerBench: PGAS benchmarks for high performance genome assembly," PAW, 2017.

[HPGMG14] Adams, et al., "HPGMG 1.0: A benchmark for ranking high performance computing systems," Tech report, hpgmg.org, 2014.

[KRIPKE15] Kunen, et al., "Kripke - A massively parallel transport mini-app," American Nuclear Society M&C, April 2015.

[SUPERLU05] Li, X., "An overview of SuperLU: algorithms, implementation, and user interface," TOMS, 2005.

[TCE05] Baumgartner, et al., "Synthesis of high-performance parallel programs for a class of Ab initio quantum chemistry models," Proceedings of the IEEE, Feb. 2005.

4 Word Granularity Scratchpads for Gather/Scatter

4.1 Motivation

Numerous algorithms, including sparse matrices, tensor contractions, and many PDEs require good gather-scatter performance feeding wide vector units. However, common cache-line-oriented memory structures greatly reduce the efficiency of gather scatter operations. A word-granularity scratchpad can enable wide-vector units to perform at near maximum throughput, even for strided or highly irregular memory access patterns. We also note that it would be challenging to implement a scalable cache coherence policy that operates at a word granularity, so moving to word-granularity forces an alternate approach.

4.2 Architecture/Implementation

The scratchpad must be organized as at *least* 16 independent 64-bit (word-granularity) "lanes" to match the performance of a 128-bit cache line. We propose to expand to 64 lanes to reduce bank conflicts for an 8-slot AVX-512-like vector unit by a factor of 4x over the worst-case baseline. These 64-lanes are also the natural interface-point to the Recoding Engine described in the prior section, which also requires a 64-lane scratchpad memory. The scratchpad can therefore be shared by the AVX and the Recoding Engine.

4.3 Evaluation Conditions

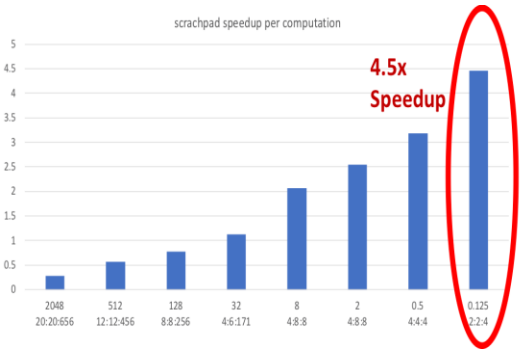
The hardware architecture parameters (power, area, performance) for the scratchpad and cache memory SRAM mats were computed using Cacti [CactiDocs]. Simulation of data access patterns using word-granularity vs. cache-line granularity was performed using a PINtool [PINdocsl] to intercept the load and store instructions and run them through a memory hierarchy model.

4.4 Results

Experiment1: This experiment with MTC demonstrates substantially reduced wasted data movement for word granularity vs. cache-line granularity for Tensor Contractions. The application used in this example is MTC, which is a microbenchmark meant to mimic high dimensional tensor calculations and is integral to quantum simulation software. We define "wasted data movement" as the ratio of useful vs. unnecessary data moved during load/store operations - ((fetched words-useful words)/useful words).

The figure to the left shows AVX performance measured by Intel's vTune as a function of memory stride. Performance drops off by a factor of 8x or more with even small strides in the gather pattern. The word granularity scratchpad is intended to eliminate this massive compute performance degradation while simultaneously reducing the amount of unnecessary data movement for such gather operations.

Experiment2: We demonstrate that the word granularity scratchpad enables a 4.5x increased ability to strong-scale PDEs on block-structured grids. We use the simple PDE solver that evolves the hyperbolic wave equation in 3D as our benchmark kernel for this experiment. We built an analytic model of the cache+prefetch design with a small cache-block size that modeled both the prefetch warm-up time, and the latency incurred by reading the ghost-zone data from other cores (the coherence latency) along with the grind-time for the compute part of the kernel. The analytic model was calibrated against a full cycle accurate architecture model, and the analytic model was used for the majority of the design space exploration because the cycle-accurate simulation would have been too slow to cover the full design space. We also used a script to automatically search through all possible loop blocking configurations to find the optimal implementation for each problem size.



Impact: As shown in figure [PDEsolverEval] on the left, the word granularity scratchpad greatly improved strong-scaling of processing elements because of the substantially reduced overheads for halo-exchange and by eliminating coherence overheads. There are numerous PDE problems that have already become limited by the inability to strong-scale. Weak scaling involves growing the size of the problem and proportional to the amount of parallelism to hide overheads (which express themselves as Amdahl's law). However, when you increase the problem resolution by a factor of N, the time-step must be decreased by a factor of N^2 to N^3 (depending on problem dimensionality) to maintain the Courant stability condition. The result is that the time it requires to solve such problems as resolution increases is growing exponentially. The dramatically reduced overhead enables more processing elements to be applied to a fixed problem size so that more speedup is possible with a larger number of cores (e.g. 4.5x increased strong-scaling over conventional cache).

Experiment3: The area and latency benefit of the scratchpad compared to cache is studied with the Cacti model [CactiDocs]. As shown in the figure, the scratchpad capacity is 72% larger than the cache when meeting the 1ns access latency (L1 speed). This substantially increases the amount of fast processor-local memory available to the accelerator cores. Additionally, word granularity scratchpads can be seen as "packed" caches which make better use of space.

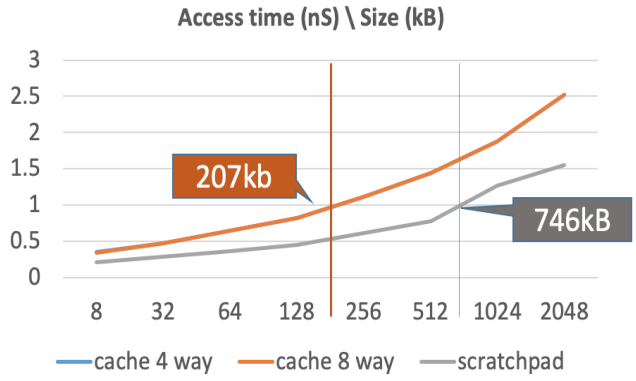


Figure [SRAM-area-timing] on the left shows the area and latency comparison between scratchpad and cache under 32nm technology.

4.5 Software

Traditionally scratchpad memories have been considered much more difficult to program than a conventional automatically managed cache. However, with the broad adoption of GPUs, which require a similar explicit copy-data-in + copy-data-out semantics to move data from the host to accelerator memory, the data movement operations can largely be automated using OpenMP and OpenACC style constructs. So, increased user familiarity with software managed memories (through the GPU experience) and today's software tools make coding manageable.

Besides, the scratchpad memory can be operated side-by-side with a conventional cache, provided an appropriate memory consistency model is used to avoid conflicts (such as adopting the GPU *release consistency model*). These concepts have been demonstrated to be partitionable to support both capabilities concurrently (and are supported by modern embedded core and DSP design libraries) [VirtualLocalStore]. Additionally, the software managed memories can be mapped into a global address space with little added complexity, to enable a very lightweight PGAS communication model with zero software overhead for inter-processor data exchange.

No additional code generation for vector instructions (such as SVE or AVX) would be required to access data that is placed in the local store – strided accesses would see no degradation except in the case of cache bank conflict.

4.6 References

[GreenFlash] John Shalf, David Donofrio, Chris Rowen, Leonid Oliker, Michael F. Wehner: **Green Flash: Climate Machine** Encyclopedia of Parallel Computing 2011: 809-819

[GreenFlash2]
https://cloudfront.escholarship.org/dist/prd/content/qt5958c7vj/qt5958c7vj.pdf

[VirtualLocalStore] Henry Cook, Miquel Moretó, Sarah Bird, Khanh Dao, David A. Patterson, Krste Asanovic: **A hardware evaluation of cache partitioning to improve utilization and energy-efficiency while preserving responsiveness.** ISCA2013: 308-319

[CactiDocs] Muralimanohar, Naveen, Rajeev Balasubramonian, and Norman P. Jouppi. **"CACTI 6.0: A tool to model large caches."** *HP laboratories* 27 (2009): 28.

[PINdocs] Levi, Osnat. **"Pin-a dynamic binary instrumentation tool."** (2018).

5 Recode Engine

5.1 Motivation

Data movement cost is a critical performance concern. Recode/UDP (Unstructured Data Processor) is an efficient, software programmable, data recoding accelerator that is orders of magnitude more energy-efficient (1,900x) and area-efficient than conventional CPU cores on recoding tasks [Acc19, UDP17, UAP15] as illustrated in Figure UDP-PERF. These capabilities enable systems/applications to select the right encoding of data for each stage of the computation, and in each part of the system (core, LLC, main memory, storage), and can enable significant performance increases as well as power and bandwidth reductions at fixed performance.

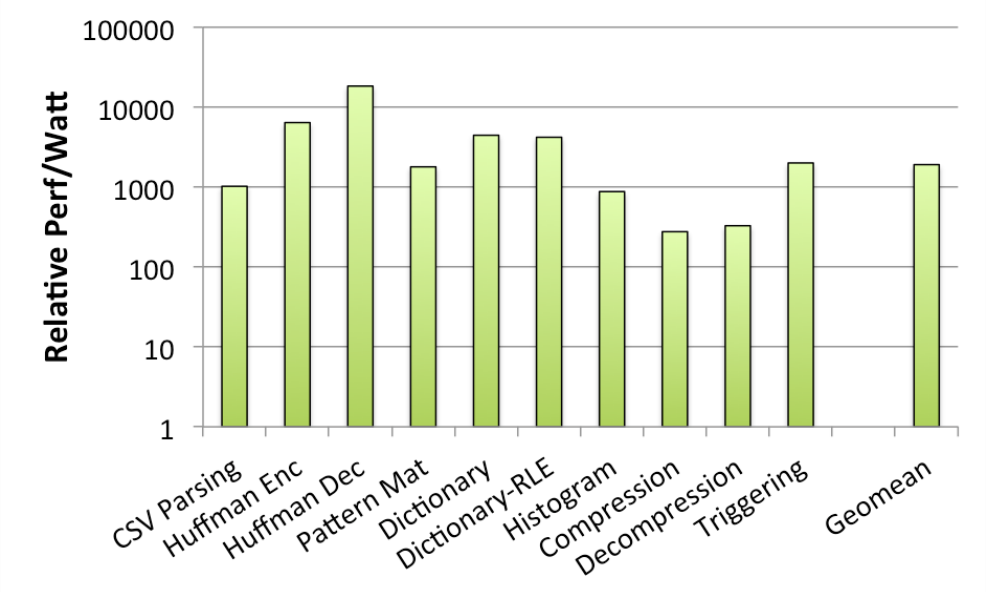


Figure UDP-PERF: Relative Performance of the Recode/UDP to an x86 core.

5.2 Evaluation/Results

We evaluated the Recode Engine on streaming SpMV computations for a wide range of the TaMU Sparse Matrix collection [HCW19], and report on selected matrices from Project 38, in Figure RECODE. Programmability enables the matrix encoding to be selected for effectiveness - here we use a delta, LZ, Huffman combination. Other applications well studied with even greater performance benefits include regex matching, network flow monitoring, export-transform-load (ETL) and data analytics [UAP15, UDP17, Acc19] as in Figure Acc.

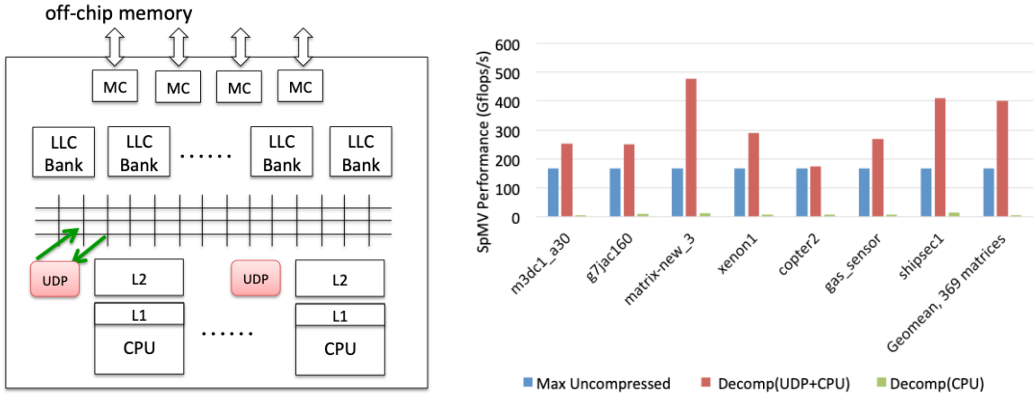


Figure RECODE: Recode Memory Hierarchy Integration and SpMV Performance benefits

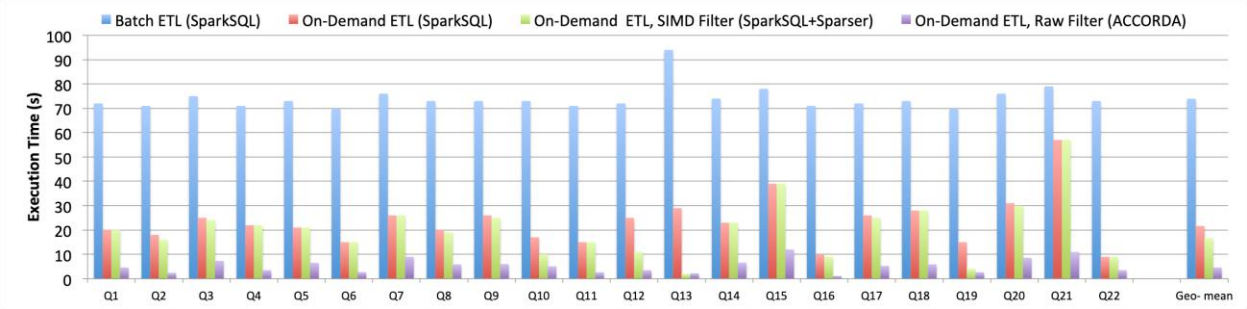


Figure Acc: TPC-H Raw Data Processing: Recode can effectively eliminate the cost (16x) of Raw Data processing

5.3 Implementation/Architecture

Placement of the Recode engine in the system architecture has several options: Scratchpad/L1 attach and LLC/MC attached on the CPU chip. In other research, storage attach is being explored as is cooperation between a Recode engine and flexible scatter-gather units (See Section 6).

5.4 Software

The software programming model for Recoding transformations will be familiar to developers who have used various forms of accelerators. Note that applications, or sections of code, that do not use or need the Recoding Engine can simply ignore it. For common transformations, one simply calls library functions that have been built for the Recoding Engine. For example, a set of assembly-coded libraries are available for Regex and for operations such as LZ compression, delta encoding, run-length, Huffman and so on.

In cases where an existing transformation is not available, we are developing high level tools to support code development with a familiar C-level source interface. The programming environment includes LLVM compiler code generation and a library composition framework to ease use by applications and amplify performance benefits. The compiler will recognize patterns, e.g., sparse matrix operations, and automatically target the Recode Engine. System software and runtime additions will complete the Recoding support.

5.5 References

[Acc19] Yuanwei Fang, Chen Zou, and Andrew A. Chien, Accelerating Raw Data Analysis with the ACCORDA Software and Hardware Architecture", in Proceedings of the 45th International Conference on Very-large Databases (VLDB), Los Angeles, August 2019.

[UDP17] Yuanwei Fang, Chen Zou, Aaron Elmore, and Andrew A. Chien. UDP: A Programmable Accelerator for Extract-Transform-Load Workloads and More, in Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50), October 2017, Boston, Massachusetts.

[UAP15] Yuanwei Fang, Tung Hoang, Michela Becchi, and Andrew A. Chien. Fast Support for Unstructured Data Processing: The Unified Automata Processor, Proceedings of IEEE Conference on Micro-architecture (MICRO-48), December 2015, Honolulu, Hawaii.

[HCW19] Arjun Rawal, Yuanwei Fang, and Andrew A. Chien. Programmable Acceleration for Sparse Matrices in a Data-movement Limited World , in Heterogeneous Computing Workshop 2019, Rio de Janeiro, Brazil, May 2019. Affiliated with the International Parallel and Distributed Processing Symposium (IPDPS).

6 Flexible Scatter-Gather Memory Controller

6.1 Motivation

HPC applications are often thought of as "math-heavy." However, analysis of these application's actual dynamic instruction mix usually reveals that floating-point instructions make up only a small portion of total instructions (see Figure [MemUse]). Inspection of the code indicates that the majority of integer instructions are used to calculate memory addresses.

It is reasonable to say that what **HPC programs really "do" is not floating-point, but memory.** Unfortunately, memory performance is still the limiting factor for on-node performance of many HPC applications. In three of the four applications studied, for every cacheline of eight words that is brought to the L1, less than 5.6 words (mean) were used before the line was evicted. These miniapps appear to waste 30-35% of their cacheline data and the bandwidth which feeds it. Just as it is reasonable to say that HPC programs really "do" memory, **it is also reasonable to say that we do it somewhat poorly.**

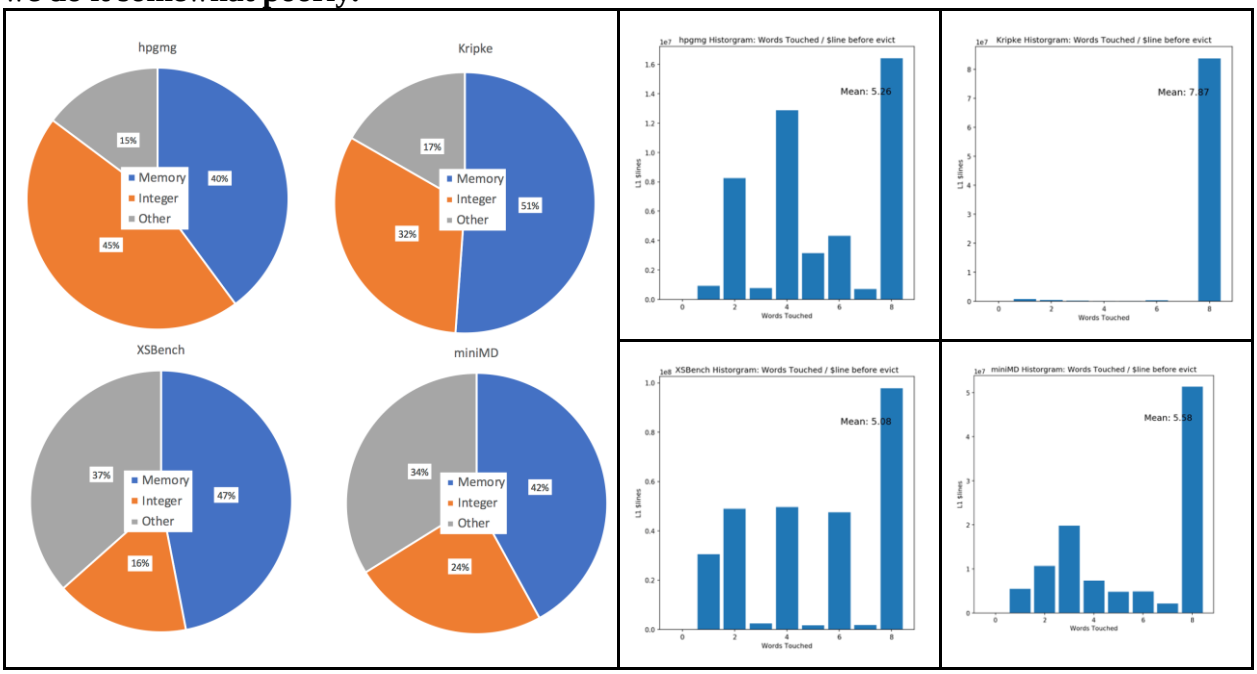


Figure [MemUse]: Memory use and misuse in HPC applications

6.2 Architecture

Our proposed architecture is a near-memory scatter-gather unit (SGU), similar to the Data Rearrangement Engine (DRE) [DRE], which gathers (scatters) data in to (from) a scratchpad. In

this model, the memory transactions are issued by the DRE to a near-memory controller and only packed cache lines are forwarded to (received from) the CPU. Near-memory scatter/gather conserves memory bandwidth and reduces cache pollution. Calculations with the packed data can be vectorized or fed to a Recode Engine, yielding even greater performance benefits. While this work focuses on applications performing memory moves, it could be part of a longer function pipeline such as a key/value store lookup accelerator [KVStore].

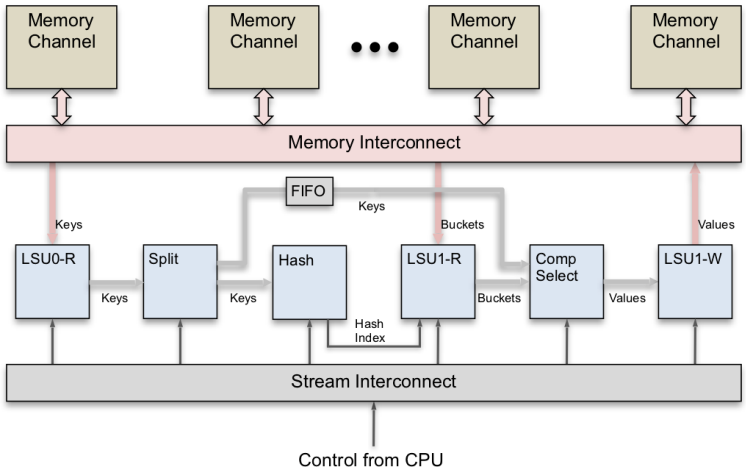


Figure [KVStore] Open address hash table lookup pipeline uses DRE components to access hash table slots

6.3 Software Issues

The most direct programming model for scatter-gather units is the setup/fill/drain model first developed for the DRE architecture. In this model the SGU is first configured (setup) with base addresses and sizes of data structures, and then fill and drain commands are used to fill a scratchpad with gathered values or to scatter values from the scratchpad out to memory.

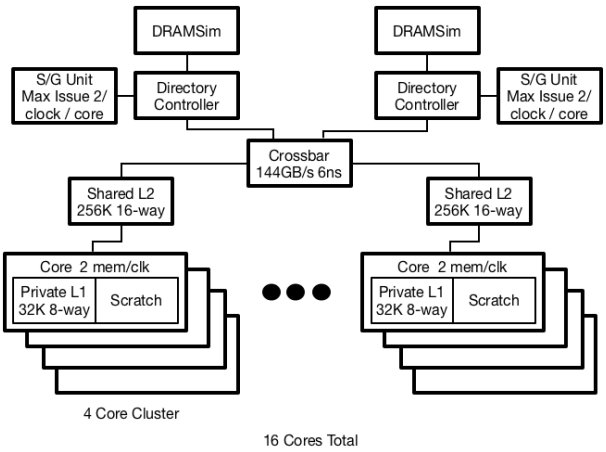
This pattern can be used by higher level abstractions, such as those found in emerging HPC performance portability frameworks like Kokkos [Kokkos] and Raja [Raja]. Setup/fill/drain commands, could map into framework abstractions such as Kokkos' Kokkos::View or the proposed C++2x mdarrary which support arbitrary load mechanisms on access.

Sharing of memory regions between CPU and near memory scatter-gather units requires careful coordination. In the absence of a near-memory MMU, the hardware architecture of the near-memory SGU is most efficient when applied to large contiguous physical memory regions. To support contiguous physical memory allocation through the OS, a Linux Contiguous Memory Allocator (CMA) region can be reserved at boot time. Other approaches are investigated by the Spidre architecture.

6.4 Evaluation

Preliminary analysis was performed with the Structural Simulation Toolkit (SST) [SST] and with gem5 [GEM5]. The SST was modified to include a simple SGU attached to the memory system

and gem5 with a similar module [PIMSIM]. Simulations were performed in two ``modes'': Prefetch and Scratchpad.



In the Prefetch mode, the SGU acted only as a prefetcher targeting the cache. A command is issued from the processor to the SGU which begins loading from memory into the processor's cache. This has the benefit of easier implementation - no new memory has to be added to the processor chip, and no synchronization is required between the processor core and the SGU. The downside is the possibility that prefetched lines will be evicted before they can be used, reducing efficiency.

In Scratchpad mode, the SGU transfers data to/from a user-visible scratchpad. This scratchpad is physically located next to the L1 and virtually located in a reserved portion of the address space. We assume a hardware synchronization mechanism between the core and scratchpad which delays any load to scratchpad data that has not yet arrived.

Sections of each target application were manually modified to add control information for the SGU. For the hpgmg application we examined two different scatter-gather strategies, gathering across different loop nestings.

6.5 Results

The most important goal of the SGU study is to evaluate how much the SGU can improve whole application performance. Simulations for the SGU in Prefetch mode (Figure [SGPerf]) indicate that for the chosen application set execution time improves by 15-28%. Though these numbers are substantial, it should be noted that this is total application performance. Since only a portion of the memory accesses (30-50%) are amenable to SGU acceleration, the speedup of these portions is even higher - 30-270%. Also, these results show that a poor scatter-gather operation (the alternate ``hpgmg-ijk'' strategy tried with hpgmg) can lead to poor performance.

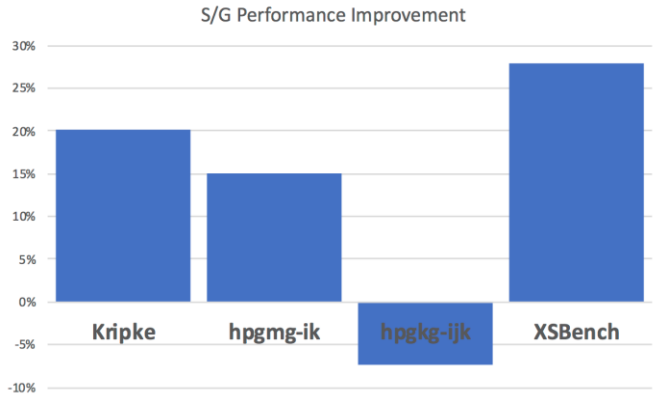


Figure [SGPerf]: Performance improvement for different applications using the SGU as a prefetcher

The SGU improves the L1 miss rate. The impact on miss rate is modest for Kripke and hpgmg (<18%), but substantial for XSBench, reducing L1 misses by over 50%. Another benefit of the SGU is that it offloads many address calculations from the CPU cores into the memory system. Though it accounts for only a small minority of instructions on most codes (4-9%), for XSBench it is close to 15%.

Performance results for the SGU to Scratchpad case are not yet available. However, early simulation does provide some positive findings. Most data which is moved to the scratchpad by the SGU is reused several times before the scratchpad is cleared. For example, on average each hpgmg data word is accessed 25.8 times before it is drained. It is possible that with a more adept SGU strategy and better application knowledge the reuse rate could be even higher. Another useful result is the SGU requires relatively modest scratchpad sizes of 6768 to 41080 bytes – similar to or smaller than an L1 cache.

Simulations with gem5+PIMSIM show the limits of in-memory scatter/gather. For applications sizes which already achieve a high cache-hit rate and are amenable to existing vector extensions (e.g. ARM SVE), an SGU may not be effective as the latency to memory is higher than to cache. Additionally, care must be taken in cache management (e.g. invalidation/flushing) to ensure 'good' data is not lost.

6.6 SGU + Recode Engine

Scatter/gather hardware can be used in conjunction with other modules. For example, a combined architecture in which the DRE gathers sparse data for processing by the Recode Engine, as illustrated in Figure [SGArch]. In this model, the SGU gathers sparse accesses to the dense vector while the Recode Engine performs the dot products. Early results (Figure [RCDREperf]) indicate that combining the SGU and Recode engine can lead to a 40% improvement on sparse matrix-vector multiple compared to the SGU alone.

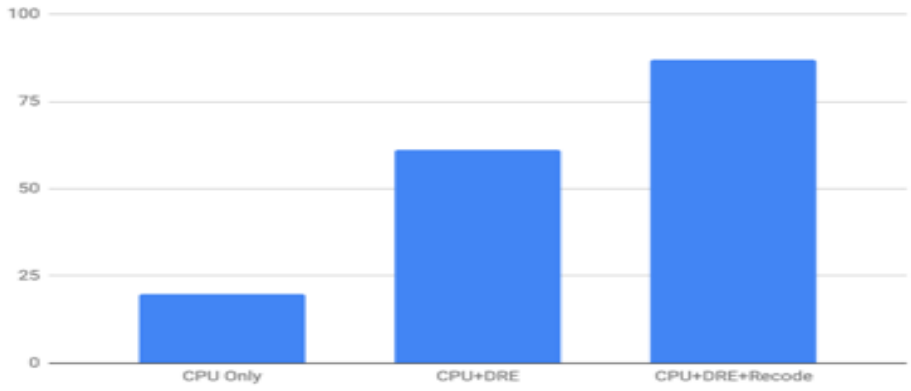


Figure [RCDREperf] MFlop/s for SpMV with Recode+SGU.

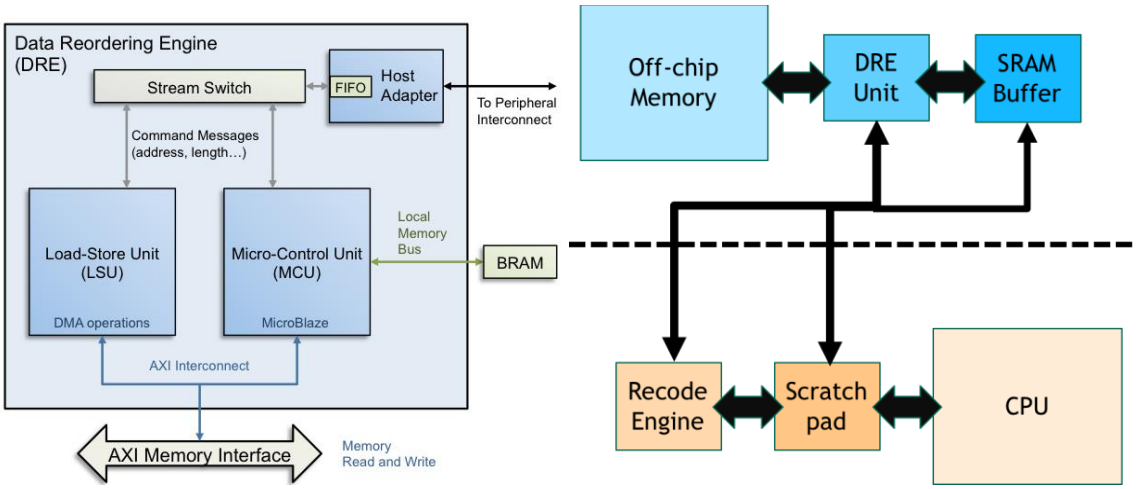


Figure [SGArch]: DRE and Recode engine

6.7 References

[DRE] Gokhale, M., Lloyd, S., and Hajas, C. Near memory data structure rearrangement. In Proceedings of the 2015 International Symposium on Memory Systems (2015), ACM, pp. 283–290.

[KVStore] Lloyd, S., and Gokhale, M. Near memory key/value lookup acceleration. In Proceedings of the 2017 International Symposium on Memory Systems (2017), ACM.

[SST] Rodrigues, A. F., Hemmert, K. S., Barrett, B. W., Kersey, C., Old eld, R., Weston, M., Risen, R., Cook, J., Rosenfeld, P., Cooper-Balis, E., and Jacob, B. The structural simulation toolkit. SIGMETRICS Perform. Eval. Rev. 38 (March 2011), 37–42.

[Kokkos] Edwards, H. C., Trott, C. R., and Sunderland, D. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. Journal of Parallel and Distributed Computing 74, 12 (2014), 3202 – 3216. Domain-Specific Languages and High- Level Frameworks for High-Performance Computing.

[Raja] Tramm, J. R., Siegel, A. R., Islam, T., and Schulz, M. Bench - the development and verification of a performance abstraction for monte carlo reactor analysis. In PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future (Kyoto, 2014).

[GEM5] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. 2011. The gem5 simulator. SIGARCH Comput. Archit. News 39(2):1-7.

[PIMSIM] S. Xu, X. Chen, Y. Wang, Y. Han, X. Qian and X. Li, "PIMSim: A Flexible and Detailed Processing-in-Memory Simulator," in IEEE Computer Architecture Letters, vol. 18, no. 1, pp. 6-9, 1 Jan.-June 2019.

7 Low-Overhead Interprocessor Messaging (MessageQueues)

7.1 Motivation

DAG-type communication patterns are widely-used to express complex dependent operations in sparse matrix computations. Libraries such as PLASMA and MAGMA [Dongarra] and SuperLU [SherryLi] demonstrated opportunities for increased parallel efficiency of such algorithms, but cache-coherence mechanisms alone are inefficient in implementing such communication patterns. Traversing each dataflow edge (activate the next step DAG node) is costly, reducing efficiency. Direct message queues could improve efficiency for these algorithmic patterns.

7.2 Architecture/Implementation

Message queues that connect processing elements with hardware-managed atomic insertion can eliminate many mutex locks (necessary in cache coherent systems). Such queues can be integrated with word-granularity local store, setting aside some store space for implementation. Interthread latencies can be reduced as shown in figure [MessageQueues] (performance from cycle-accurate RTL emulation). Additional documents describe the complete architecture for the NoC and hardware generator [OpenSoC] and evaluation of the message queues using FPGA emulation [OpenSoC2]. Message queues implementations are available in embedded design libraries such as Tensilica's TIEqueues and in the ARM design library.

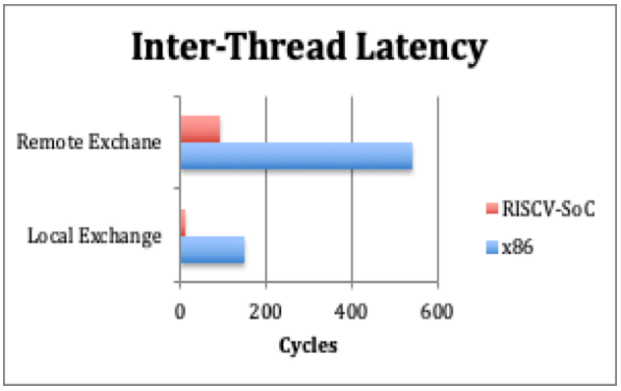
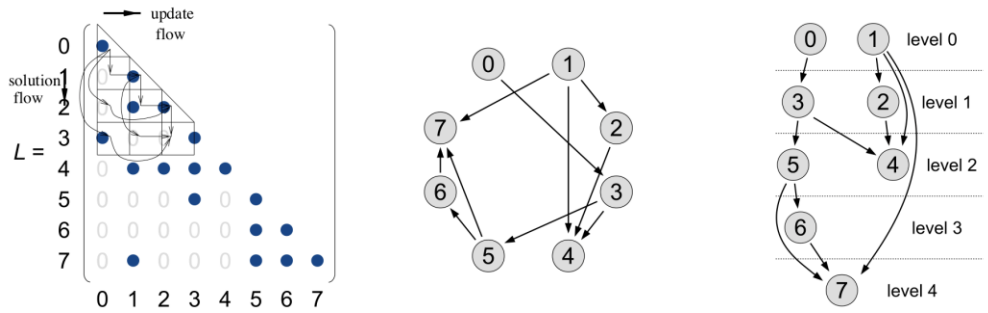


Figure [MessageQueues]: Interprocessor latency: Cache coherence vs. message queues.

7.3 Experiment

We used the Sparse Matrix Tri-Solve to assess the impact of message queues on DAG communication patterns used to track tri-solve row dependencies. We instrumented SuperLU to collect the dependency graph (DAG) and fed it into an analytic model. The performance model for the cache system was calibrated with vTune on the Xeon Phi platform (NERSC Cori), whereas the message queue (MsgQ) model come from a full RTL design [OpenSoC]. In all cases, the Intel Xeon Phi (KNL) cores are used for processor core performance. The DAG for the sparse matrix trisolve Figure [TrisolveDAG] tracks the dependencies between rows given the sparsity pattern (a). This can be represented, as level-sets (c), requiring barriers between levels (poor scalability). We study directly following the DAG edges as in (b).



(a) L's matrix form. (b) L's graph form. (c) Level-sets generated.

Figure [TrisolveDAG]: This shows 3 representations of the trisolve dependency graph (DAG).

7.4 Results

Baseline SuperLU uses OpenMP atomics to track the DAG edges; performance is limited by the underlying hardware - not the OpenMP implementation. vTune measurements show the impact of the overhead on trisolve's overall scalability. Parallel efficiency suffers as the overheads for dependency tracking increasingly dominate runtime (see figure [TrisolveOMP]).

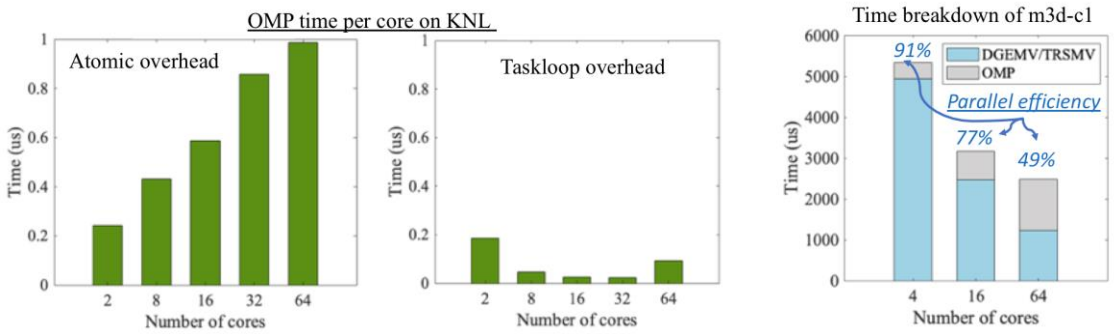


Figure [TrisolveOMP]: Breakdown of timing and overheads for sparse matrix trisolve (SuperLU) on 64-core KNL.

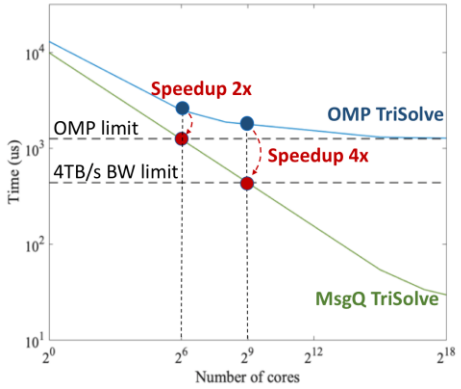


Figure [TrisolveMsgQ]: Trisolve performance model is used to project scalability of MsgQ vs. OMP implementations out to 2^18 compute elements.

With direct hardware message queues, performance improves as in Figure [TriSolveMsgQ]. Beyond 32 cores, the OpenMP implementation's scaling degrades, and cannot even saturate the memory interface of KNL, whereas the MsgQ scales to many more cores and thereby can saturate memory bandwidths of 4TB/s or more. In summary, the hardware message queues for fast interprocessor communication offer a 4x-8x improvement of performance over the baseline OpenMP/atomics, and offer the ability to scale performance to higher parallelism. Beyond TriSolve, DAG tracking is used in many sophisticated Sparse Matrix algorithms.

7.5 Software

The message queues interface is typically exposed as intrinsics that look to the programmer to be subroutine calls for all practical purposes. The intrinsics are implemented using macro expansion of embedded assembly snippets, or by compiler modification. In terms of programmer experience, many pseudocode implementations of sparse matrix algorithms are expressed using a message queue semantics, so we believe programming is natural. Figure [SuperLUcode] shows a side-by-side comparison of the OpenMP and MsgQ implementations of the key DAG-tracking kernel for SuperLU. Minimal changes were required, and readability improved.

OMP Original Code

```
1.  Initialization: x := b;  lsum := 0
2.  while (I have work to do) do
3.      find fmod(K) ≤ 0 for some K
4.      if (fmod(K) = −1 and I own this column)
5.          for (each of my L(I,K) ≠ 0, I > K)
6.              lsum(I,myid) = lsum(I,myid) + L(I,K)x(K)
5.      #pragma omp atomic
7.          fmod(I) = fmod(I) − 1
18.     end for
19.     else if (fmod(K) = 0 and I own the diagonal block)
20.         x(K) = x(K) − lsum(K,id) for each id in row K;
23.         x(K) = L(K,K)^−1 x(K)
5.      #pragma omp atomic
14.         fmod(K) = fmod(K) − 1
26.     endif
27. end while
```

Algorithm 1: Lower triangular solve $Lx = b$ with OpenMP.

Converted to MsgQ

```
1.  Initialization: x := b;  lsum := 0
2.  while (I have work to do) do
4.      if (pollQ() > 0)
5.          getQ(&vfunc, &K)
3.          fmod(K) = fmod(K) − 1 or fmod(K) = −1 by &vfunc
26.     endif
4.      if (fmod(K) = −1 and I own this column)
5.          for (each of my L(I,K) ≠ 0, I > K)
6.              lsum(I,myid) = lsum(I,myid) + L(I,K)x(K)
7.          putQ(&id, &vfunc, &I) for id on diagonal block
18.     end for
19.     else if (fmod(K) = 0 and I own diagonal block)
20.         x(K) = x(K) − lsum(K,id) for each id in row K
23.         x(K) = L(K,K)^−1 x(K)
7.          putQ(&id, &vfunc, &K) for each id in column K
26.     endif
27. end while
```

Algorithm 2: Lower triangular solve $Lx = b$ with TaskQ.

Figure [SuperLUcode]: Comparison of the OMP and MsgQ implementations' code.

7.6 References

[OpenSoC] Farzad Fatollahi-Fard, David Donofrio, George Michelogiannakis, John Shalf:

OpenSoC Fabric: On-Chip Network Generator: Using Chisel to Generate a Parameterizable On-Chip Interconnect Fabric. NoCArc@MICRO 2014: 45-50.

[OpenSOC2] Farzad Fatollahi-Fard, David Donofrio, George Michelogiannakis, John Shalf: **OpenSoC Fabric: On-chip network generator.** ISPASS 2016: 194-203.

8 Fixed Function Hardware for FFT and Bioinformatics Acceleration

8.1 Motivation

One of the techniques used by the smartphone industry to squeeze extra performance out of an SoC is to have dozens of discrete fixed-function accelerators co-integrated onto the chip. However, the functionality of those accelerators is tailored to cellphone workloads.

We wondered whether this strategy could be utilized for accelerating HPC-relevant kernels, and what kernels would offer the most benefit. We started with the FFT challenge problem to better understand how fixed-function accelerators can be used productively for scientifically relevant kernels using the SPIRAL FFT hardware generator. This approach does not out-perform tuned GPU and CPU software implementations because performance ultimately gets bounded by memory bandwidth, but it does achieve a ***1-2 orders of magnitude improvement in performance/chip-area and performance/watt*** compared to the CPU and GPU implementations.

We extended this fixed-function accelerator study to investigate a bioinformatics accelerator for the HipMER de-novo genome assembly application. One of the most resource intensive steps of the HipMER application is the banded Smith-Waterman (SW) for sequence alignment [MerAlign2015]. A conventional implementation of accelerated Smith Waterman is fundamentally unscalable because complexity grows with sequence length. For this reason, we adopted a String Independent Localized Levenstein Automata (SILLA) approach [GenAx2018], which implements a non-deterministic Finite State Automata (FSM) for sequence matching where the complexity is independent of sequence size. Whereas Smith-Waterman FSM is O(N) complexity in area and $O(N^2)$ compute (where N=sequence length) for the general case and O(kN) complexity for banded SW (where k is the match distance), the SILLA FSM is $O(k^2)$ in area and O(N) in compute complexity. SILLA FSMs are very simple with 13 logic gates per FSM state and multiple implementations available (FPGA, ASIC, etc.). The challenge is that an FPGA implementation is too slow and an ASIC would restrict it to a fixed function. How can we get speed and also reusability/programmability from our accelerator solution? To get both flexibility and performance, we implemented SILLA using ***Recoding Engine*** as a reprogrammable FSM accelerator. In doing so, we achieve a ***53x speedup*** over GPU implementations of banded SW.

8.2 Implementation and Experimental Conditions for FFT Acceleration

We used the SPIRAL hardware generator (https://www.spiral.net/hardware.html) to create a custom FFT hardware accelerator circuit block. To compute power and area, we ran the FFT circuit through the Mentor Graphics Design Synthesis to route the circuit for a 14nm ASIC target as shown in Figure [FFThw]. This provides us with accurate area, power consumption, and timing estimates for the circuit. The throughput of the circuit can be computed using an analytic model, but we also routed the full circuit design combined with a RISC-V core onto an FPGA emulation platform to serve as a full cycle accurate model of the chip design. For a baseline comparison, we measured the performance of the optimized FFTW library on Intel Xeon Phi (KNL) and an NVIDIA V100 GPU. The benchmark used was the stock HPCC FFT benchmark, which is a very large 1D FFT using 32-bit floating point complex operands.

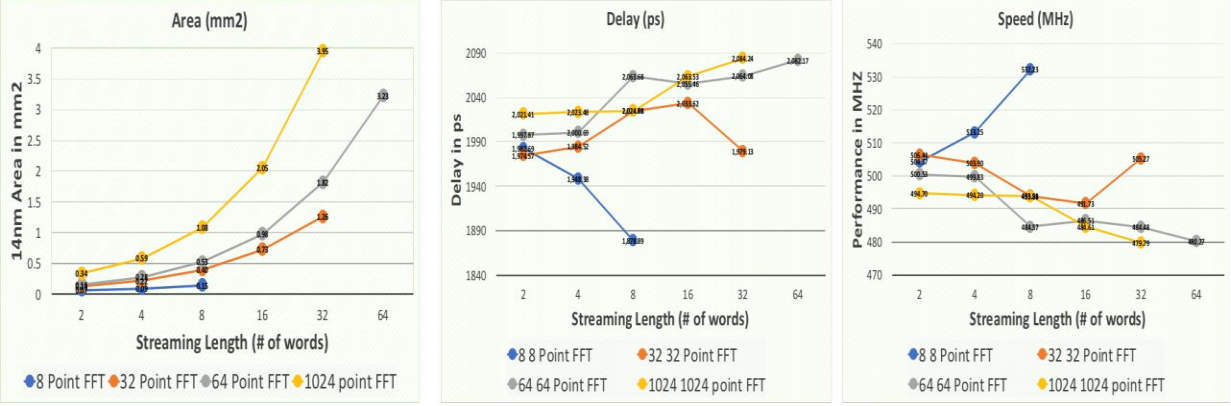


Figure [FFThw]: Chip-layout at 14nm using Mentor Design Synthesis Flow. Shows 2x improved density improvement over an analytic model, but 2x slower clock. Floating point multiplier is the critical path around 1900 ps leading to 500 MHz design for standard cell based synthesis. An improved StdCell library (better than OpenSDK) could result in further improvement.

8.3 Results for FFT accelerator

The benchmark used was HPC-Challenge 32bit complex out-of-place transform with CUFFT for the V100 and FFTW for Intel. The FFT accelerator hardware was generated by SPIRAL [SPIRAL] with one targeting just enough hardware to saturate a 100GB/s memory interface (much like the Intel server chips) and 1TB/s memory interface bandwidth limit, which can keep more accelerator hardware utilized. The performance results shown in Figure [FFTperf] show that the performance delivered is not substantially higher than that of the NVIDIA GPU as the performance is memory subsystem limited, but the performance per area (area efficiency) and performance per watt (energy efficiency) is a full one to two orders of magnitude better than for the GPU and three orders of magnitude better than the equivalent of the x86 with AVX512 (note the logarithmic scale for the horizontal axis).

We also investigated packaging the accelerator as an ISA extension which enables tighter coupling of the programmable nature of the processor core with the fixed-function acceleration of FFTs, and the emulated RTL on our FPGA platform performed nearly 10x faster than the native x86 host processor for FFT-heavy image processing tasks despite running at a 10x slower clock rate. Next steps will be to study other common numerical kernels such as BLAS for fixed function accelerators.

	Peak FP32 TFLOP/s	Measured FFT Performance	Area mm^2 @14nm	Power (Watts TDP)
NVIDIA V100	14	2	610	250
Intel Xeon Phi (KNL)	3	0.095	450	250
Intel Haswell (E2699v3) (note: 22nm)	1.29	0.041	660	145
SPIRAL FFT (100GB/s Dram)	-	0.192	3	1
SPIRAL FFT (1TB/s Dram)	-	3	64	22

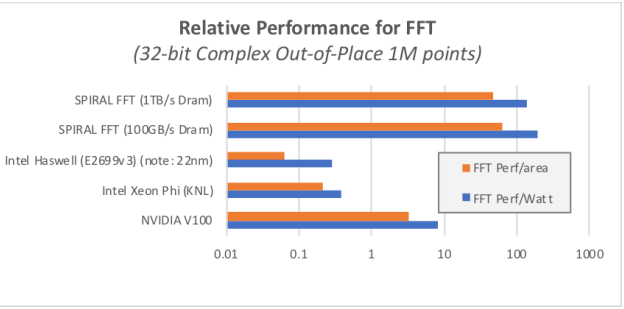


Figure [FFTperf] Performance of the FFT benchmarks (on the left). Performance/area and performance/watt for the fixed function hardware (SPIRAL generated) FFT is shown on the right.

8.4 Implementation and Experimental Conditions for Bioinformatics Accelerator

Acceleration of the banded Smith-Waterman sequence matching involves efficient representation of the Levenshtein automata and hardware architecture supporting this implementation. To enable high throughput acceleration of computation involved in the Smith-Waterman algorithm, we introduce the first method of using the SILLA automaton-based acceleration using a Recoding Engine as shown in Figure [P38-SW]. This SILLA (String Independent Local Levenshtein Automata) automaton was introduced in [GenAx] and was used as one of the components of the GenAx architecture. We extend this implementation to be realized using the instructions and intrinsics available in our Recoding Engine architecture framework. To do this we take two-prong approach of acceleration: first, by using the already available instructions from the Recoding engine and next by implementing a custom hardware accelerator. The performance improvement achieved by the custom accelerator called GenAx reported in the literature [GenAx] is 64x compared to the sequential cpu baseline.

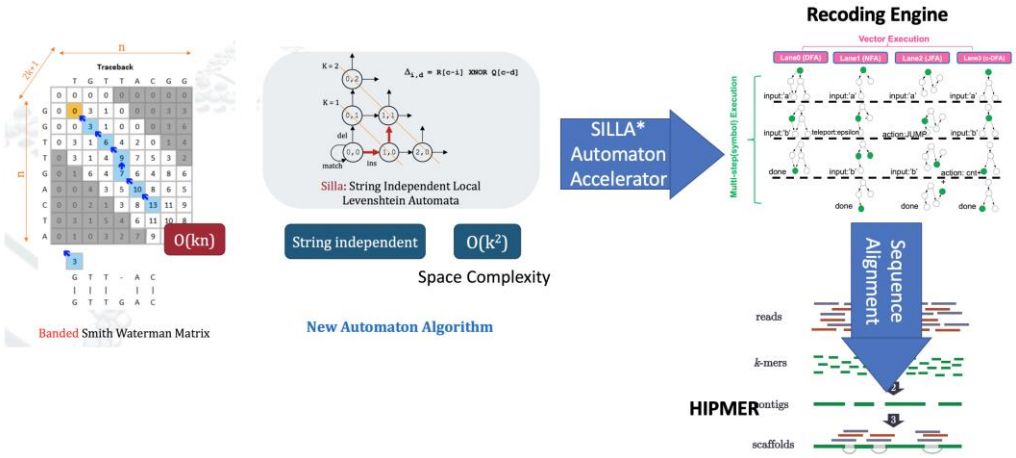


Figure [P38-SW]: Acceleration of the banded Smith-Waterman (SW) Algorithm with Recoding Engine and Custom Implementation.

It enables 4058 Kreads/second sequence alignment throughput which is due to the efficient implementation of the automaton data-structure and the custom accelerator design using this data-structure. But, this style of acceleration lacks programmability and is expected to run as a standalone co-processing element. Integrating them into a programmable heterogeneous architecture will enable broader use of these accelerators to speed up genomics applications.

8.5 Preliminary Results and Analysis for SILLA Bioinformatics Accelerator

Before implementing the algorithm using the SILLA automaton, we did a scalability experiment to discover the achievable bandwidth limit for this type of accelerator. To do so, one of the authors of the LBNL Meraligner software, Marquita Ellis, evaluated the Smith-Waterman section of the software on Cori. Results are shown in Figure [Cori-Eval] which compares the results from Cori HPC system in LNBL, sequential cpu results of running SeqAn on Xeon, and scaled results from the SILLAx, the SILLA accelerator. As shown in the Figure [Cori-Eval], Meraligner on Cori empirically scales from 2.5 up to 12.6 million alignments/Second using 1 to 8 nodes. The SeqAn scaling trend is lower than Meraligner due to its sequential implementation. For Sillax, we scaled

the single node performance using the trend from Cori, and for 1 to 8 nodes, alignment from 56.6 Million alignments/sec to 288.8 Million alignments/sec can be achieved.

Finally in Figure [Perf-Pow] the expected performance and power results of our standalone implementation of the SILLA accelerator are shown. The comparison with the CPU, GPU, and GenAx accelerator is shown. We expect the performance and power of our implementation to be between the standalone GenAx accelerator and the GPU implementation. This is due to the overhead of using Recoding Engine's intrinsics to implement the SILLA accelerator based Smith-Waterman algorithm and custom accelerator implementation without using all the acceleration in GenAx which will impact the performance and power. Nevertheless, better power and performance results are predicted compared to the reported CPU and GPU implementations and close to the full custom GenAx accelerator implementation.

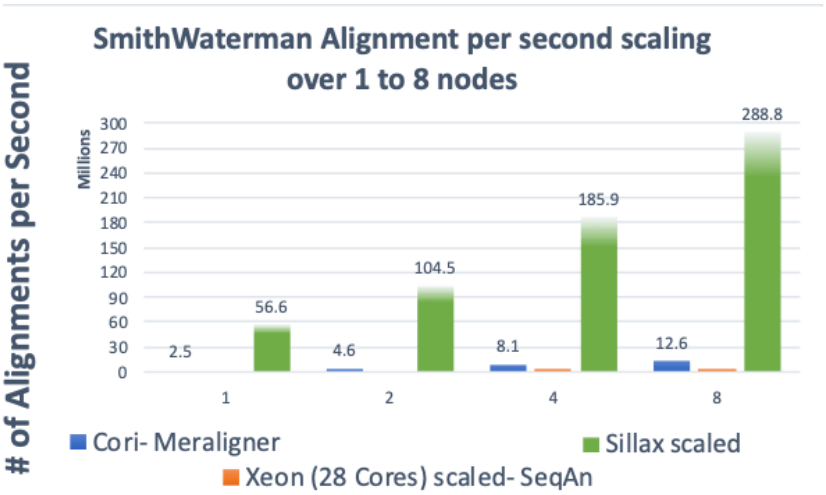


Figure [CoriRes]: Initial assessment results of Smith-Waterman alignment algorithm. Comparison is shown for the baseline (sequential implementation in Xeon (28 core)), the Smith-Waterman part of the Meraligner run on Cori, and accelerated alignment using the Sillax accelerator. Experiments on Cori with Meraligner are run by Marquita from LBNL.

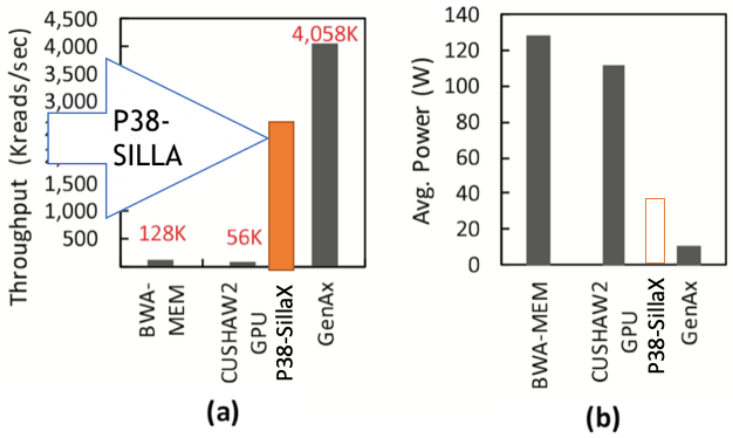


Figure [PerfPow]: Expected results from P38-SILLA. (a) throughput expectation shown with arrow, (b) power results expected to be between custom accelerator like GenAx and GPU based design

8.6 Software for FFT and SILLA

In both the case of the SILLA accelerator and for the FFT accelerator, the interfaces make use of existing API interfaces to existing software implementations of these respective functions. For example, the interface to the discrete accelerator can be hidden behind an FFTW API. In the case of the SILLA accelerator personality for the sequence alignment matching, we had to create our own API interface to invoke the sequence matcher. However, the interface invokes a software implementation of the SILLA FSM by default, and will invoke the Recoding Engine accelerated FSM if the personality is loaded using the Recoding Engine's APIs as described in section 5.4 (Recoding engine software).

We also experimented with exposing the FFT interface as an ISA extension to the RISC-V instruction set architecture for much tighter integration. The ISA extension, albeit more flexible, requires more extensive modifications to the compiler to add the FFT construct to the intermediate representation and to add hooks to emit the FFT assembly instructions.

8.7 References

[SPIRAL] Thom Popovici **"An Approach to Specifying and Automatically Optimizing Fourier Transform Based Operations,"** PhD. thesis, Electrical and Computer Engineering, Carnegie Mellon University, 2018. (http://spiral.net/hardware.html)

[GenAx2018] D. Fujiki *et al.*, **"GenAx: A Genome Sequencing Accelerator,"** *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, Los Angeles, CA, 2018, pp. 69-82.

[SeqAn2017] K. Reinert, T. H. Dadi, M. Ehrhardt, H. Hauswedell, S. Mehringer, R. Rahn, J. Kim, C. Pockrandt, J. Winkler, E. Siragusa, G. Urgese, and D. Weese, **"The seqan c++ template library for efficient sequence analysis: a resource for programmers,"** *Journal of biotechnology*, vol. 261, p. 157–168, 2017.

[MerAlign2015] E. Georganas, A. Buluç, J. Chapman, L. Oliker, D. Rokhsar and K. Yelick, **"merAligner: A Fully Parallel Sequence Aligner,"** *2015 IEEE International Parallel and Distributed Processing Symposium*, Hyderabad, 2015, pp. 561-570.

[Darwin2018] Yatish Turakhia, Gill Bejerano, and William J. Dally. 2018. **Darwin: A Genomics Co-processor Provides up to 15,000X Acceleration on Long Read Assembly.** SIGPLAN Not. 53, 2 (March 2018), 199-213.

9 Curation and Knowledge Transfer of Experiments, Results and Research Material

9.1 Motivation

The curation and dissemination of the experiments, results, and supporting research material is critical to ensuring that the knowledge developed in this project is readily transferable to enable follow-on projects. To support this, the Open Curation for Computer Architecture Models

[OCCAM] tool suite has been selected to serve as the confluence for the research derived from each of the respective P38 teams.

9.2 OCCAM Overview

OCCAM is a community-driven active curation platform that allows its users to contribute and share their artifacts and experiments [OCCAM]. Developed by the University of Pittsburgh (Pitt), under the leadership of Professor Bruce Childers, OCCAM satisfies the research community's need to preserve experiments, experimental data and results at every stage of the research process. OCCAM satisfies these requirements by offering long term preservation and reproducibility of software experiments, to include the model source code, build and run instructions, configuration parameters, and critical dependencies. This data is preserved recursively and accessible with the same fidelity at all levels. The OCCAM tool suite offers a user-friendly web-based interface, allowing researchers to access and conduct experiments, view resulting data, and review available supporting documentation. Additionally, when using Sandia National Laboratories' Structural Simulation Toolkit [SST] OCCAM offers a graphical interface to archived models, enabling researcher's ability to connect (wire) models, configure and execute experiments and examine the results.

9.3 OCCAM's Application to P38

ACS has initiated a preliminary discussion with Pitt to arrange for the activation of an OCCAM instance dedicated to P38 research. Pitt has recommended the use of the Pittsburgh Supercomputing Center [PSC] as the site to host OCCAM and archive the P38 research materials. Based on discussions with Pitt, the PSC has the experience and resources needed to readily support the needs of P38 research teams.

9.4 OCCAM Availability

Based on conversations with Pitt and PSC it is anticipated that all of the necessary computer and software resources will be available for P38 use by December of 2019. To ensure the availability of P38 materials, the hosting site will need to employ highly reliable computer systems and provide timely administration.

9.5 OCCAM Training

To support the use of the OCCAM platform, a hands-on training session for all of the P38 researchers will be available. It is anticipated that the training will be offered around the same time that the PSC activates the P38 OCCAM website. Additionally, the possibility of on-line training, presented by Pitt professors, is under consideration. Forums that have been suggested include a live-VTC and a recorded video option with availability in Dec 2019 / Jan 2020.

9.6 References

[OCCAM] L. Oliveira, D. Wilkinson, D. Mosse, B. Childers, Supporting Through Artifact Evaluation with OCCAM - http://rescue-hpc.org/_resources/20181111-occam_rescue-hpc-sc18-workshop-paper.pdf

[PSC] Pittsburgh Supercomputing Center https://www.psc.edu/

[DEMO] OCCAM Demonstrations: https://occam.cs.pitt.edu

[SST] Sandia National Laboratories, Structural Simulation Toolkit: http://sst-simulator.org/

10 Conclusions

The first phase of Project 38 has demonstrated significant potential benefits to USG-relevant applications. Six key proxy applications of interest and five key architectural features were analyzed and determined to provide improvements in areas ranging from performance, energy-efficiency, throughput, or capacity from 40% to over 10x. Further integration efforts and studies are required to fully characterize the benefits realizable at a full-system scale.

Significant progress has been made in identifying the design decisions, integration challenges, and software ecosystem support required to allow such architectural features to be successfully incorporated into future HPC designs. The lessons learned from this first phase of the project serve as starting points to help communicate USG needs, requirements, and desires to the larger community of vendors, system integrators, and academic researchers.

Documentation of the work, to enable knowledge transfer and to support vendor engagements, is underway. Initial vendor engagements have been identified; these will become ongoing in FY20.

Project 38 has served as a catalyst for developing a deeper understanding of USG HPC interests across the NSA and DOE. It has increased interactions between researchers across the USG, helped reveal the unique requirements and challenges faced by each organization, and established a foundation upon which future engagements can build.

11 Appendix A: Background Information

In September 2016, ~60 HPC experts met to discuss/respond to the June 2016 announcement of China's TaihuLight supercomputer. The attendees reached the following consensus (a full report is available [DJM1]).

- The HPC technology ecosystem is changing in ways that are less favorable to HPC
- There are national security implications to this change
- Leadership in innovative architectures is critical
- Joint architectural explorations between NSA and DOE might be useful and interesting

Approximately 40 DOE and NSA HPC experts met again at IDA/CCS-Bowie for a weeklong technical deep dive from September 25-29, 2017. DOE and NSA held follow on meetings, telecons, and VTCs. Key accomplishments from these activities are:

- An improved understanding of key applications
- Description of specialized architecture development process and specific examples
- Discussion of possible architectures and applications of interest
- Defining a value proposition for joint explorations

- Refinement of the exploration space and exploration process

On April 12, 2018, ~25 DOE and NSA experts held an all-day meeting that identified a small, specific set of architectural ideas worth exploring, and developed initial work plans for exploring the ideas. Improvements in data access are the initial focus, and the explorations have been generally limited to node level results. This meeting represents the formal launch of Project 38. The near-term milestones for Project 38 are:

2018
Quantify the benefits of the explorations, primarily through modeling and simulation. The benefit is determined by quantification of the performance improvement against a baseline of vendors' technology roadmaps (business as usual).

2019
Complete the existing explorations by increasing understanding of the costs - adverse changes to programming models, application development, system SW stacks, etc. Document the results to enable knowledge transfer and extend the initial explorations.

2020
Improve the fidelity of the cost-benefit analyses.
Push the best ideas towards implementation, primarily through aggressive vendor engagements.
Develop one system reference design of mutual interest.

Project 38 explorations should accomplish the following:

- Establish general boundaries of the risk/reward for purpose-built architectures of shared interest;
- Quantify the improvement of purpose-built architectures for applications of interest;
- Greatly increase the understanding by the DOE and NSA of architectural features that have value to both organizations;
- Greatly increase the understanding by the DOE and NSA of the possibility of developing a shared architecture;
- Increase the depth and the quality of the DOE and NSA technical working relationship;
- Enable the DOE and NSA to communicate both shared and individual priorities in architectures and innovations to the vendor community and senior USG stakeholders.
- Develop a capability for the DOE and NSA to individually or jointly explore innovations with the ability to quantify their value (for certain classes of applications) to their partner.

11.1 References

[DJM1]
https://www.nitrd.gov/nitrdgroups/images/b/b4/NSA_DOE_HPC_TechMeetingReport.pdf.

"Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Networking and Information Technology Research and Development Program."

The Networking and Information Technology Research and Development (NITRD) Program

Mailing Address: NCO/NITRD, 2415 Eisenhower Avenue, Alexandria, VA 22314

Physical Address: 490 L'Enfant Plaza SW, Suite 8001, Washington, DC 20024, USA Tel: 202-459-9674, Fax: 202-459-9673, Email: nco@nitrd.gov, Website: https://www.nitrd.gov