Scaling the Power Wall: A Path to Exascale

Oreste Villa, Daniel R. Johnson, Mike O'Connor, Evgeny Bolotin, David Nellans,
Justin Luitjens, Nikolai Sakharnykh, Peng Wang, Paulius Micikevicius, Anthony Scudiero,
Stephen W. Keckler and William J. Dally
Email: {ovilla, djohnson, moconnor, ebolotin, dnellans, jluitjens, nsakharnykh,
penwang, pauliusm, ascudiero, skeckler, bdally}@nvidia.com

***Abstract*—Modern scientific discovery is driven by an insatiable demand for computing performance. The HPC community is targeting development of supercomputers able to sustain 1 ExaFlops by the year 2020 and power consumption is the primary obstacle to achieving this goal. A combination of architectural improvements, circuit design, and manufacturing technologies must provide over a 20× improvement in energy efficiency. In this paper, we present some of the progress NVIDIA Research is making toward the design of Exascale systems by tailoring features to address the scaling challenges of performance and energy efficiency. We evaluate several architectural concepts for a set of HPC applications demonstrating expected energy efficiency improvements resulting from circuit and packaging innovations such as low-voltage SRAM, low-energy signaling, and on-package memory. Finally, we discuss the scaling of these features with respect to future process technologies and provide power and performance projections for our Exascale research architecture.**

I. Introduction

Modern scientific discoveries in areas such as genetics, drug discovery, and particle physics rely heavily on large-scale simulation and modeling. The demand for expanding computational capability drives the creation of increasingly higher-performance supercomputers. A continuation of historical scaling would predict that exascale supercomputers (i.e. able to sustain 1 ExaFlops = 10^{18} Flops) will start to appear by the year 2020 [1]. One of the main challenges in achieving this goal is power consumption, which a range of HPC system operators have suggested be limited to 20 MW for a full exascale-capable system to mitigate cost of ownership and new power delivery infrastructure costs [2]. The current generation of energy-efficient supercomputers rely heavily on general purpose Graphic Processing Units (GPUs) to scale up their floating point throughput [3]. GPUs have a large energy efficiency advantage with respect to conventional CPUs because they are designed to exploit high levels of data and thread level parallelism for performance rather than extracting instruction-level parallelism from a small number of threads [4]. For highly parallel workloads, GPUs can be up to $10\times$ more energy efficient than CPUs within the same area and power budget [5].

Exascale designers are facing a "power wall" when trying to design future systems capable of sustaining 50 GFlops per Watt (GF/W) on dense codes such as Linpack [6]. The current Titan supercomputer at Oak Ridge National Laboratory consists of 18,688 nodes, each containing one NVIDIA Tesla K20X GPU. On the Linpack benchmark, this system achieves 17.59 PetaFlops while requiring 8.2 MW of power – delivering 2.14 GF/W [7]. To enable exascale systems in the future, energy efficiency must improve by a factor of over $20\times$ to reach the 50 GF/W target. To achieve such a large improvement in energy efficiency will require many contributions beyond process technology scaling [8]. Process technology improvements alone will only provide approximately $4.3\times$ of the required improvement in energy efficiency. An additional $1.9\times$ can be extracted from circuit improvements, enabling operation at lower VDD. A further $2.5\times$ contribution to energy efficiency must come from additional architectural and circuit techniques, as well as a system level design and interconnect that allow efficient scale-up of node-level performance.

At NVIDIA Research, we are developing an integrated set of technologies spanning architecture, circuits, and co-designed HPC applications targeting these performance and energy goals. While we recognize the resilience and programming systems are also important to exascale, this paper focuses solely on the power challenges. Because no single silver bullet will enable exascale level performance and energy efficiency, it is critical for designers to understand the opportunities for improvement provided by each area alongside the predicted process technology scaling. This paper aims to quantify the contributions of different technologies on the road to exascale and project performance and energy consumption of a 2020 exascale system on a suite of DoE HPC proxy applications [9], [10]. We expect that exascale applications, like the proxy-apps that represent them, will exhibit highly varied execution profiles when compared to Linpack. Consequently, exascale systems must be able to efficiently handle these variations and not be over-optimized for dense matrix multiplication. This paper builds upon the earlier Echelon project by expanding upon and extending these research concepts with quantitative analysis of application-level power and performance through a variety of future technology nodes [11]. This paper makes the following contributions:

- We summarize and explain the behavior of a suite of DoE Proxy applications on a modern GPU-based supercomputer.
- We propose an energy-efficient node and system architecture, combining new architectural and circuit innovations.
- We compare the power and performance of our proposed design to currently available GPUs in a 28nm process technology and quantify the contributions of the architecture and circuit-level features as technology scales down to 7nm.
- We estimate application-level power/performance for exascale systems by combining node level simulation with a scaling model derived from scalability measurements taken on modern supercomputers.

SC14, November 16-21, 2014, New Orleans
978-1-4799-5500-8/14/$31.00

To our knowledge, no prior work has delivered a single comprehensive overview of the future applications, challenges, and features to provide exascale-level performance and power estimates for future HPC systems.

This paper is organized as follows. Section II describes the application used in this study. Section III presents the overall system architecture. Section IV presents the methodology used in our studies. Section V presents the power and performance results at node, comparison with a modern GPU, discusses scaling at full system level. Section VI presents related work focusing on architecture, circuit features, and technology scaling studies. Finally Section VII concludes the paper.

II. Scientific Applications

While ultra-dense workloads, such as Linpack, are often the benchmark by which HPC systems are measured, real supercomputing applications are rarely as regular. As a result, machines that are over-optimized for these dense workloads tend to have poor real-world utilization when other applications are ported to them. To encourage the design and development of balanced high performance supercomputers, the DoE has funded the development of proxy applications (proxy-apps) for workloads they view as important to scientific computing in the next 10 years [9], [10]. These proxy-apps are designed to be simplified, but representative, versions of these workloads suitable for public sector research and development. To ensure maximum portability, the proxy-apps consist of Fortran/C/C++ code using MPI for intra-node communication. Parallelism within the node is typically exposed with threading libraries such as POSIX threads or OpenMP pragma directives. For this work, NVIDIA helped develop GPU implementations of the proxy-apps using numerically equivalent CUDA kernels or OpenACC compiler directives injected into the original code.

In this section we describe the proxy-apps used in this study, focusing on the particular characteristics each application presents and the challenges it introduces for building a well-balanced exascale architecture. At the compute-node level, we focus on aspects such as the desired bytes to Flops ratio, memory intensity, control flow divergence, and CPU/GPU work partitioning. At the system level, we discuss scale-out properties for the application such as the communication to computation ratio and communication overlap opportunities to hide latency overheads. All of these application properties are integrated into our performance and power model for exascale computing which we discuss in Section IV-B.

A. CoMD

CoMD is a proxy-app for molecular dynamics simulations which represent a significant fraction of the DoE workloads [12]. CoMD solves Newton's laws of motion for individual atoms for short-range inter-atomic potentials. CoMD uses the Embedded Atom Model (EAM) potential as an approximation for describing the energy between two atoms. EAM is a widely-used model of atomic interactions in metallic systems. At each discrete time-step, this model requires evaluation of the two-body force and embedded energy contribution. The core algorithm behind the computations is the neighboring atoms traversal within a cutoff radius to accumulate forces or energy for each atom.

The parallelization scheme divides the 3D domain into sub-boxes which are assigned to MPI ranks. After receiving the force and position information of the neighboring sub-boxes from the previous time-step, each MPI rank can compute forces and positions at the current time-step for the atoms assigned. Because of the short-range potential calculation and the relatively short cutoff radius, larger sub-boxes result in more computation and less communication. The computation of forces, which can take up of 95% of the total time, can be separated among internal-to-internal atoms and external-to-internal atoms interactions. This approach enables overlap of intra-node communication with internal-to-internal force calculation if the problem size is sufficiently large.

B. LULESH

LULESH represents a typical hydrodynamics modeling application which describes the motion of materials relative to each other when subject to forces [13]. LULESH is a simplified application implemented to only solve a simple Sedov blast problem with analytic answers but represents the numerical algorithms, data motion, and programming style typical in production codes. LULESH approximates the hydrodynamics equations discretely by partitioning the spatial problem domain into a collection of volumetric elements defined by a mesh. A node on the mesh is a point where mesh lines intersect. LULESH uses a regular mesh, but to retain unstructured data proprieties, the data structures make use of indirection arrays to index into the mesh.

At scale, the parallelization consists of assigning a subset of elements to each MPI rank. Because the amount of data communicated among MPI ranks is proportional to the number of surface elements while computation is proportional to the volume of assigned elements, communication becomes relatively inexpensive with respect to computation for sufficiently large problem sizes. Because communication overhead has minimal performance impact, it is not anticipated that LULESH-like workloads will need to implement communication and computation overlapping to enable good weak-scaling on exascale systems. Within the node, the GPU executes a total of 9 kernels of which kinematic and volume force calculations are the most time consuming (about 50% of the total application time). These two functions are latency limited on current GPU hardware and do not sustain high utilization of compute or memory systems. The remaining 7 kernels are fully memory bandwidth bound.

C. MiniFE

MiniFE is a proxy-app that mimics the finite element generation, assembly, and solution of an unstructured grid problem [14]. The physical domain is a 3D box with configurable dimensions. The domain is decomposed using a recursive coordinate bisection (RCB) approach and the elements are simple hexahedra. The problem is linear and the resulting matrix is symmetric, so a standard conjugate gradient algorithm is used with a general sparse matrix data format and without preconditioning. At scale, hexahedral elements are partitioned across MPI ranks, with communication occurring on the contacting faces, edges, or corners, causing each MPI rank to communicate with at most with 26 neighbors (6 faces, 12 edges and 8 corners). With a sufficient number of MPI

ranks (128 in our experiments), communication bandwidth between neighbors becomes nearly uniform, resulting in near ideal load balancing across MPI ranks. The communication can be easily overlapped with the sparse-matrix vector product (SpMV) computation, which takes more than 80% of the total execution time. SpMV is fully memory bound, requiring up to 28 bytes of streaming loads for each double-precision operation. To represent the sparse matrices, different data structures can be used. Typically, Compressed Sparse Row (CSR) representation results in simpler code while ELLPACK matrix format (ELL) results in higher performance [15]. We find that on our research architecture, the presence of large caches makes this performance difference negligible; we use the CSR format to ease programmer burden.

D. SNAP

SNAP is a proxy-app that models a modern discrete ordinates neutral particle transport application, used to solve the linear Boltzmann transport equation (TE) [16]. The Boltzmann TE is a governing equation for determining the number of neutral particles (e.g., neutrons and gamma rays) in a multi-dimensional space. The solution to the time-dependent TE is a "flux" function of seven independent variables: three spatial (3D spatial mesh), two angular (set of discrete ordinates, directions in which particles travel), one energy (particle speeds binned into groups), and one temporal. The iterative strategy consists of two nested loops which are performed at each sequential time-step of the simulation. The outer iterative loop involves solving for the flux over the energy domains (typically tens to hundreds of domains) while the inner loop involves sweeping across the entire 3D mesh along each discrete direction of the angular domain.

Within the node, a single kernel consumes 95% of intra-node execution time, calculating the energy for the mesh's cells. At scale, the 3D mesh is spatially decomposed and mapped on a 2D domain of MPI ranks using the KBA method [17]. Due to the wavefront propagation method used in the algorithm, each MPI rank receives data from at most 3 upstream MPI ranks and sends data to 3 downstream MPI ranks following the wave propagation. For this reason, an MPI rank can start computation only after its inputs arrive in the propagating wavefront. This in turn limits weak scaling efficiency, although absolute performance still increases with system size. Given tens or hundreds of energy domains, the compute kernel can be executed in two ways: (1) starting as many energy groups as possible in parallel or (2) working on a few energy groups at a time as fast as possible. Depending on which strategy is used, application scaling behavior changes dramatically. At scale, we have observed that working on a few energy groups as fast as possible results in better scalability and overall shorter execution time even if single node performance is not maximized.

E. CNS

CNS is an application developed by the DOE ExaCT co-design center to serve as a proxy for the dynamical core computation of a combustion code [18]. CNS is a stencil-based computation used to integrate the Compressible Navier Stokes (CNS) equations with constant viscosity and thermal conductivity. Because each updated element in a stencil requires

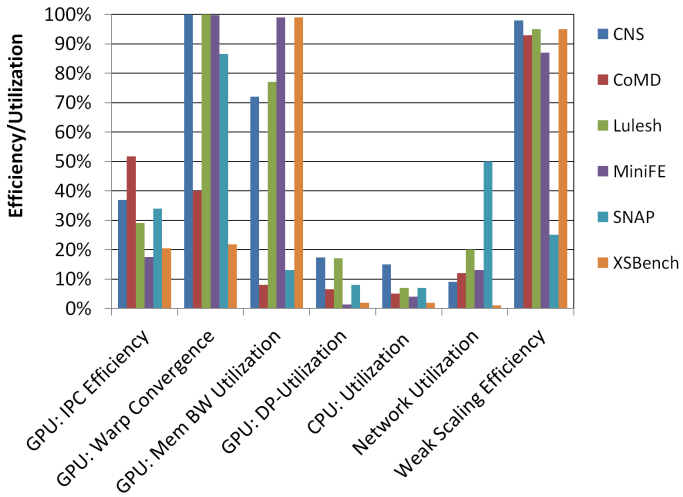

Fig. 1. Key characteristics of DoE proxy-apps on current HPC systems.

a large amount of data from neighboring points (dependent on the radius of the stencil), stencil computation is typically memory bandwidth bound. A third order TVD Runge-Kutta scheme is used to integrate a set of PDEs with 5 unknowns: density, three momenta, and energy. Eighth order stencils are used to compute discrete spatial derivatives. The process can be viewed as computing various derivatives and combining them to produce outputs. Both first and second derivatives need to be computed. Furthermore, second derivatives include mixed derivatives in addition to axis-aligned derivatives. Mixed derivatives are computed in two passes in both the CPU and GPU codes. This application has reduced arithmetic intensity due to the majority of time being spent in stencil computation for derivatives. For example, a first derivative computation requires 64 bytes of data for 11 arithmetic operations. Thus, it is imperative to exploit spatial data locality to reduce pressure on memory bandwidth.

Within the node, the computation is performed almost entirely by the GPU in nine different kernels, while the CPU coordinates execution and handles communication. The full domain is a large grid regularly decomposed in sub-domains among MPI ranks, each of which is a regular 3D grid. For proper time-stepping, these sub-domains must exchange the halos with variables that are being integrated. A given sub-domain can have up to 26 neighbors for halo exchange. However, halo data exchange becomes negligible or completely overlapped by the computation if the problem size is large enough.

F. XSBench

The XSBench proxy-app models the most time-consuming part of the OpenMC Monte Carlo neutronics application, the calculation of macroscopic material cross sections [19]. This kernel accounts for up to 85% of the total runtime of OpenMC when running the Hoogenboom and Martin reactor benchmark. Typical scaling of Monte Carlo simulations consists of executing many separate instances of the same code (with different initial conditions) followed by a reduction to obtain the final results. XSBench is primarily focused on intra-node performance characteristics. The cross section calculation in

OpenMC is a memory intensive operation because neutron energy is handled continuously, i.e. it is not quantized. XSBench models the two main Hoogenboom and Martin problem size variants: small and large. The only difference between these variants is the number of nuclides represented in the fuel. In both benchmarks, the Flop to byte ratio is approximately 0.2, with the large case using much more data in the fuel. Since the additional data is only in the fuel (responsible for approximately 14% of lookups), large amounts of execution divergence is experienced. However, the proxy-app performance will be dominated by memory bandwidth on future exascale machines, not floating-point throughput.

As XSBench models node-level behavior for the larger OpenMC application, it is not scaled directly. However, the behavior of an individual node on XSBench models the expected behavior of each node within a system at scale, each operating on independent data and ultimately bound by memory bandwidth.

G. Applications Summary

Figure 1 summarizes the application characteristics relevant to the performance and power studies presented in the remainder of this work. These metrics represent average efficiency and utilization for some of the resources for a system like the Titan supercomputer at Oak Ridge National Labs (ORNL Titan). Figure 1 shows the following metrics:

- *GPU: IPC Efficiency*. The average percentage of IPC achieved versus the maximum theoretical capability of the GPU. IPC is limited by stalls due to memory access latency, resource conflicts, and control divergence.
- *GPU: Warp Convergence*. The average execution path convergence of threads within a warp. A value of 100% indicates that all threads within a warp execute the same instruction every cycle (full convergence). A value of 50% indicates that, on average, only half of the threads within a warp are executing on a given instruction.
- *GPU: Mem BW Utilization*. The average percentage of peak GPU memory bandwidth used.
- *GPU: DP-Utilization*. The average utilization of the double-precision floating-point units in the GPU with respect to maximum theoretical utilization.
- *CPU: Utilization*. The average utilization of the CPU (across all cores).
- *Network: Utilization*. The utilization of the network calculated as a percentage of the achieved bandwidth with respect to maximum bandwidth.
- *Weak Scaling Efficiency*. The ability of the application to scale across nodes as problem size increases. A value of 100% indicates that the time to solve a problem of size $S \times N$ on N nodes is identical to the time to solve a problem of size S on 1 node. A value of 10% indicates that execution takes $10\times$ longer than perfect scaling efficiency (100%).

The GPU node-level data are measured on real hardware, with results weighted by the relative execution times of the kernels in each application. When possible, network and CPU utilization have been measured on the largest number of nodes feasible and scaled with analytical models to a full system of 18,688 nodes to match the maximum size of ORNL Titan. A more detailed description of the scaling methodology is presented in Section IV-C. While Figure 1 does not provide a comprehensive evaluation of the behavior of the proxy-app suite for every dataset, configuration, and system, it provides high level information about the typical behavior of these applications on a modern GPU-based supercomputer.

Figure 1 shows that the behavior of the studied applications vary widely. For example, XSBench and MiniFE are largely bottlenecked on main memory bandwidth, while SNAP and CoMD use very little. SNAP is much more network-limited than the other applications due to its wave-propagation implementation at the system level. CoMD and XSBench both exhibit significant control divergence, making ultra-wide SIMD designs inefficient, while the other applications benefit from very wide designs. CNS and LULESH are somewhat similar in many of these metrics but CNS uses more CPU cycles while LULESH requires more network bandwidth. While many applications exhibit favorable weak scaling efficiency, SNAP scaling is limited. In contrast to the ubiquitous Linpack, these proxy-apps represent a diverse suite of applications to drive the design of a well-balanced exascale-class system architecture. Such a system should be balanced to perform well on a wide range of supercomputing applications; over-optimization on any individual axis could result in poor performance for many real-world applications.

III. Exascale System Architecture

To achieve the goal of more than $20\times$ improvement in energy efficiency to reach exascale will require contributions from technology scaling, low-voltage operation, and architecture and circuit innovations. We estimate that process scaling will provide only $4.3\times$ from 28nm to 7nm and low-voltage operation will provide only about $1.9\times$. The remaining $4.3\times$ must come from energy efficient architecture and circuits. Section II identified application characteristics which are important to address when designing a future exascale machine. This section describes technologies and architecture capabilities that aim to deliver that efficiency for the representative exascale applications. We first describe the components of an exascale system, including node architecture, network, system architecture, and programming model. We also describe circuit technologies necessary to achieve the energy efficiency. This research system targets a peak performance of 1.3 double-precision ExaFlops (EF) at a peak of 23 MW and sustained performance over 1 ExaFlops on Linpack with a target of 80% efficiency. As no application could simultaneously use all of the resources of the machine, application-level power will be less than 20 MW.

A. Node and System Architecture

Figure 2 shows the high-level system architecture, highlighting the logical design of a single node and the hierarchical system-level organization. A single node consists of a processor chip integrating many energy-efficient throughput

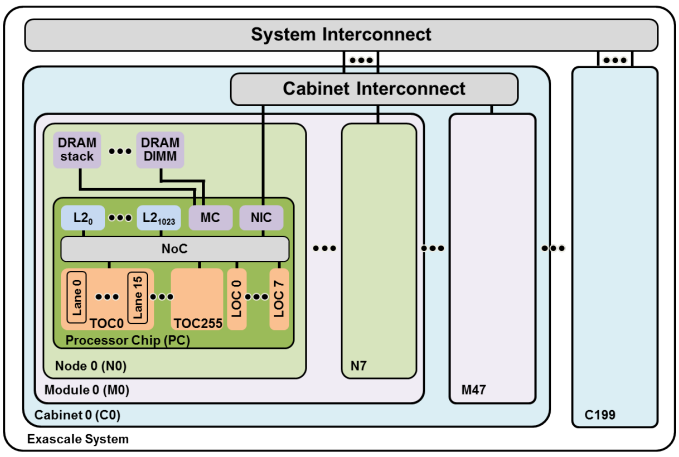

Fig. 2. Proposed exascale system architecture.

TABLE I. THROUGHPUT OPTIMIZED CORE AND EXASCALE MODEL PARAMETERS.

Node Configuration	
Technology	7nm
Number of TOCs	512
Number of LOCs	8
LOC Maximum IPC	3.0
DP ALU per TOC	16
DP ALU Total	8,192
L2 per TOC	256 KB
Total L2	128 MB
Memory Controllers	64
Total Memory BW	4 TB/s
NIC Bandwidth	100 GB/s
LOC Frequency	2 GHz
TOC Frequency	1 GHz
TOCs Total DP Perf.	16 TFlops
Processor Area	650 mm^2
Processor peak power	230 W

Exascale System Configuration	
Number of Cabinets	200
Nodes per Cabinet	384
Number of Nodes	76,800
Number of Network Slices	4
Total Router Count	19,200
Peak DP PetaFlops	1,258
Max Node Power	300 W
Max System Power	23 MW

optimized cores (TOCs), a small number of latency optimized cores similar to traditional CPUs (LOCs), memory controllers (MCs), a network interface (NIC), and on-chip memory (L2) all connected via a network-on-chip (NoC) [20]. The processor chip communicates directly with on-package DRAM stacks, off-package DRAM DIMMs, and the cabinet-level interconnect. At a TOC target frequency of 1GHz, the 8,192 double-precision multiply-add units (DFMA) produce a peak node-level performance of 16 TeraFlops (TF) with a thermal design point (TDP) power envelope of 230 Watts per socket including processor, memory, and voltage regulation. The per node power budget increases to 300 Watts when including system infrastructure such as network, cooling, and power distribution. Scaling to a large cabinet containing 384 of these nodes has more than 6 PetaFlops (PF) and 200 cabinets of those are able to to achieve a peak performance of 1.3 ExaFlops (EF). Table I summarizes the proposed configuration of a single node within broader system architecture. As the LOC is intended to provide support for the computations of the TOCs, it can be a fairly generic CPU; we do not describe the microarchitectural details of the LOCs in this paper.

B. Throughput Optimized Core Architecture

Figure 3 shows a detailed diagram of a TOC. Similar to contemporary GPUs, threads are organized into groups called *warps* that can be executed in a synchronized or lock-step fashion by hardware when the threads all take the same control flow path. Current hardware implements a warp size of 32, while our TOC is based on narrower 4-wide warps which reduce the impact of control flow divergence and helps improve power efficiency for control-flow divergent applications. Our architecture exploits converged warps, when all threads are simultaneously executing the same instruction (the same program counter), to build more efficient hardware with the SIMT (single instruction multiple thread) execution model [21]. When warps diverge, wide spatial SIMT hardware in current GPUs suffers reduced hardware utilization due to disabled lanes of execution. Our proposed architecture applies the SIMT model *temporally* such that threads in a warp execute sequentially within a single lane in the TOC, rather than spatially across many lanes as in existing GPUs [11]. With temporal SIMT (TSIMT), the hardware can amortize instruction fetch and decode overheads across all of the threads in a warp and can reduce divergence penalties by eliminating lane idle cycles. TSIMT simply collapses the bubbles that exist within the diverged warp and immediately begin executing the next scheduled warp with no need to clock or power gate individual lanes.

To optimize energy efficiency, our TOC applies *scalarization* within the temporal SIMT model [22]. While TSIMT execution amortizes the cost of the instruction fetch and decode across multiple threads, scalarization takes this energy optimization one step further by factoring out identical work performed across threads. With TSIMT, the first thread in the warp can execute the instruction once on behalf of all of the threads in the warp. Subsequent threads reuse the scalarized instruction result rather than re-executing the instruction. For example, loop counters or base address pointers may be common across many threads within a warp. Rather than storing and updating a copy per-thread, a copy can be maintained per-warp and stored in a shared register, saving storage space and energy by removing redundant computation and copies of identical data.

In addition to instruction pipeline optimizations such as temporal SIMT and scalarization, a large fraction of GPU chip area and power is spent maintaining the large register file required to feed all of the execution pipelines. To improve on the register energy efficiency, we adopt a compiler controlled *hierarchical register file* implementation [23], [24]. In our design, the operand register file (ORF) contains just 8 entries per thread, while each thread may have 32-256 registers in the main register file (MRF). Movement of data between the ORF and MRF is coupled with computation instructions to eliminate excess instruction fetch.

To further improve upon the energy efficiency of our register file and caching sub-system, we recognize that future HPC applications are likely to have varying needs for register file versus cache capacity to achieve optimal efficiency. We implement a *malleable memory* system proposed by Gebhart et al. that allows flexible use of on-chip SRAM to optimize energy efficiency [25]. Rather than having a fixed pool or registers per thread and cache-capacity per thread or compute cluster, malleable memory allows the compiler to identify and expose the number of registers that will be needed for any given kernel execution. If the number of registers is small, the remaining SRAM capacity can be used to expand the

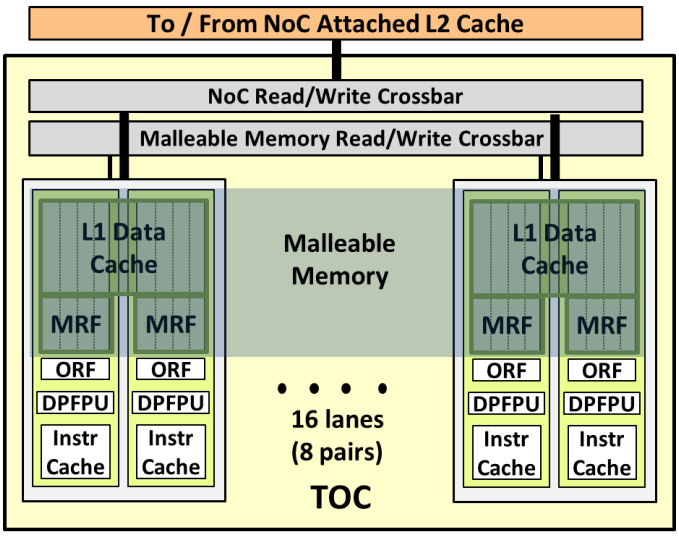

Fig. 3. Architecture of throughput optimized core (TOC).

reach of the data caches. If the compiler identifies that a large number of registers are needed for a thread's working set, the processor can decrease the capacity of the caches accordingly. By flexibly moving on-chip SRAM resources between data caches and register file usage, malleable memory helps ensure that resources are not being wasted due to a fixed provisioning of capacity.

C. Cabinet and System Interconnect

Our system supports multiple network topologies, including Dragonflies [26] which are used in this work, Flattened butterflies [27], and folded Clos (a.k.a. fat-trees) directly out of the on-die NIC within each node. Of these, the Dragonfly topology offers the lowest network diameter. The network is designed specifically to minimize the number of long network cable traversals that a packet must take and thus the total network cost to sustain a given amount of global network bandwidth. A packet can route between any nodes in the system with just three router-to-router hops: one within the local group/slice, one global hop to the target group, and one within the target group to the recipient node. This network configuration results in low cost, regular wiring within groups, high bandwidth configurability, and extreme scalability. Total network bandwidth can be configured by varying the number of nodes within a network slice/group. For this work we assume a single slice is able to provide 25 GB/s of injection bandwidth. With 64 port routers and a 3-hop diameter, the network can be scaled up to 262,144 nodes, well above our anticipated node count to reach exascale.

D. Programming Model

For the applications in this work, the overall system is programmed with a combination of MPI and CUDA. Node-level code is implemented in CUDA, and MPI orchestrates the communication across the system. This model matches the way that existing GPU-accelerated supercomputers are programmed. While we recognize and appreciate that innovation in programming models may be required for exascale systems, such an investigation is beyond the scope of this paper.

E. Circuit and Packaging Technologies

In addition to microarchitectural innovations, several circuit and packaging innovations will be crucial to meet exascale energy efficiency requirements.

Reduced-voltage operation: While Dennard voltage scaling has largely come to an end, voltage reduction remains one of the most effective approaches power reduction. Circuit techniques to reduce the effects of voltage noise, such as on-die voltage regulators [28] and voltage-adaptive clocking [29], can reduce voltage and timing margins. We expect some natural reduction in voltage with technology scaling from 0.9V in 28nm down to 0.7V in 7nm, as shown in Table II. Further voltage scaling will require circuit innovations; however because of the severe increase in gate delay, we do not expect that near-threshold voltage approaches will be viable for HPC systems.

While reduced voltage does not present challenges to the logic gates, SRAM circuits begin to fail as voltage drops [30]. We see two promising approaches to enable voltage scaling of on-chip RAM arrays. First are latch-based RAM designs which use area-optimized standard cell latches in place of SRAM. In certain capacities, these latch arrays can be area-competitive with SRAMs as SRAMs typically require additional support circuitry [31]. Second, we are investigating read-assist and write-assist circuits that enable SRAMs to operate at voltages below their natural stability point [32], [33].

Energy-efficient signaling: We have developed new signaling circuits targeting both on-package and off-package communication, called *ground-referenced signaling* (GRS) [34]. Initial silicon measurements from a 28nm test chip demonstrates 20Gb/s operation at 0.54pJ/bit over 4.5mm channels at a nominal 0.9V power supply voltage. By comparison, competing signaling techniques operate at around 5pJ/bit. We include these circuits in our analysis to reduce on-package DRAM signaling energy. While most on-chip wiring for processor chips is performed using full-swing CMOS circuits, moving to low-swing circuits provides the opportunity for substantial energy reduction. Ho et al. demonstrated a 10x energy/bit reduction in 180nm by using low-swing differential signaling, ultimately reaching 100fJ/bit-mm at a 650mV transmitter voltage [35]. In contemporary technology, we expect that the voltage levels could be reduced further an obtain perhaps 30fJ/bit-mm. Such wires are ideal for processor level network-on-chip or long wires in DRAM chips.

Stacked memory technology: We expect that the biggest improvement in memory energy will come from integrating DRAM onto the same package as the processor chip. This approach not only improves the available bandwidth with much denser and wider interfaces to memory, but it also reduces the length and capacitance of the memory IOs resulting in improved power efficiency. This technology enables memory interfaces with thousands of bits and memory bandwidths above 1 TB/s depending on the configuration. Current off-chip DDR memory systems require in excess of 15pJ/bit to obtain a per-pin bandwidth of 3.2Gbps. On-package memory, such as High-Bandwidth Memory [36], eliminates the expensive I/O drivers, termination resistors and DLLs found in DDR DRAMs, thus reducing the energy of access to 9pJ/bit at 3.2Gbps.

IV. Methodology

We rely on three major infrastructure components for simulating and projecting the power and performance of our exascale research architecture: (1) an execution-driven architectural simulator and associated compilation toolchain, capable of running CUDA-based applications; (2) a detailed event-driven energy model integrated with the architectural level simulator; and (3) a large scale system scaling model for performance and power.

A. Simulation and Toolchain

To evaluate our proposed node-level architecture, we have developed an execution-driven simulator to model our throughput oriented cores (TOCs) and their memory system. The simulator executes real CUDA applications using the NVIDIA production compiler with customized code generation for our research ISA, allowing us to compare CUDA applications that run on our research architecture against current hardware. Simulated architectural pieces are implemented as a collection of interconnected component models. A machine definition file allows dynamic configuration of models with varying fidelity and organizations, enabling rapid design space exploration. The simulator provides a detailed, cycle-accurate microarchitectural model of the proposed throughput processor, including key architectural features such as temporal SIMT pipelines with narrow warps, scalarization, a hierarchical register file, and a flexible primary memory hierarchy as described in Section III. The memory system consists of cache L2 tiles, NoC interconnect, memory controllers (MCs), and DRAM. The cache hierarchy supports hardware coherence between TOCs as well as programmer controlled cache bypassing. Off-chip DRAM models are parameterized for bandwidth and latency to match the characteristics of targeted memory technologies.

The simulator model has been verified functionally via both directed tests and comparison of program output and memory state against functional emulation and real hardware. In addition, we have correlated expected performance for key benchmarks and microbenchmarks against real hardware. Default latencies and timings are configured for components such as datapath elements and caches based on VLSI analysis and experimental RTL implementations of components, scaling or correcting for expected process technology improvements as necessary. We have configured the simulator to approximate the feeds, speeds, and capacities of a contemporary commercial GPU.

B. Energy and Area Modeling

To evaluate application-level power consumption for the proposed architecture, we developed an area model and an event-based energy model which accounts for a wide range of architectural and technological features and variations. Our model builds up a representation of the underlying architecture from individual components (such as ALUs, register files, and wires) to an entire chip and its off-chip memory system. For example, the model for a register file includes its area, the energy to access it (read and write), and the energy to send a bit across several *mm* of on-chip wires using a particular signaling technology. The model parameters are based on microarchitectural components implemented in 28nm which can be validated against current generation GPUs.

TABLE II. Technology Scaling Factors From 28nm to 7nm.

	28nm	20nm	14nm	10nm	7nm
C_g scaling factor	1.0	0.75	0.56	0.42	0.31
E_{wire} scaling factor	1.0	0.89	0.75	0.62	0.46
Area scaling factor	1.0	0.63	0.39	0.24	0.15
Leakage/mm^2 scaling factor	1.0	0.85	0.80	0.75	0.70
Nominal voltage	0.90V	0.85V	0.75V	0.725V	0.70V
Low voltage	0.85V	0.80V	0.70V	0.675V	0.65V

The modeling environment provides a menu of circuit implementation options, including low-voltage SRAM with write assists, high-density latch arrays, and optimized on-chip and off-chip signaling, as well as various memory technologies, including conventional DDR and on-package stackable memories such as the emerging JEDEC high-bandwidth memory (HBM) standard. Using performance counters and energy costs, the model calculates the total energy required, including both static leakage and dynamic energy based on the simulated machine configuration at a given frequency, voltage, and operating temperature. Our energy and area model also provides estimates for future manufacturing processes beyond 28nm.

Our strategy for estimating chip capabilities in future process generations uses the expected scaling properties predicted by the International Technology Roadmap for Semiconductors (ITRS) and other proprietary sources. We assume gate capacitance (C_g) scaling of 0.75 times per generation and fixed-length wire energy (E_{wire}) scaling at between 0.75× and 0.9× per generation. Overall, we anticipate technology scaling from 28nm to 7nm (without any circuit or architecture modification) to provide about 4.3× improvement in energy efficiency. We assume area scaling of 1.8× across technology nodes instead of the optimistic 2.0× typically assumed by squaring the ratio of gate lengths (e.g. $(20nm/14nm)^2$). We expect circuit design enhancements to enable a further scaling factor of 1.9× via enabling lower VDD. This effects result in a combined overall energy efficiency scaling factor of about 8× from 28nm to 7nm. Table II summarizes the key technology scaling factors and parameters we use in our model.

C. Large Scale System Modeling

To estimate the performance and power at scale of a contemporary system, we built analytical weak scaling performance models for each application studied and correlated them with actual experimental measurements taken on the largest systems available to us, including ORNL Titan.

For each proxy-app, our models incorporate various parameters such as problem size, memory footprint, number of nodes, network bandwidth, and single-node execution time. For power modeling, we have collected data on ORNL Titan for statistics such as CPU, memory, and network utilization, instruction mix, and GPU power consumption. CPU and memory dynamic power is computed using maximum TDP and utilization factors. Network power consumption instead is largely workload insensitive and assumed static regardless of the application. Similarly, fans are typically run near peak RPM and consume roughly the same power regardless of the workload. Finally, we combine all power contributions with the weak scaling performance model to estimate the behavior of a particular system size. The model has been validated against

numbers publicly available on the Top500/Green500 lists and other smaller machine data points collected on real hardware. This model has been used to produce the results in Section V-C for a system similar to ORNL Titan.

Our exascale system-level performance estimation is performed using weak scaling with much larger problem sizes than what could run on ORNL Titan. As per-node Flops capacity and node count for our exascale system increase $\sim$10$\times$ and $\sim$5$\times$ respectively, the average problem size is increased by a factor of $\sim$50$\times$. To estimate the performance of our proposed exascale design, we expect performance scaling for each application to be directly proportional to today's scaling factors adjusted by different possible ratios of node peak performance, obtained via our node level simulation model, to network injection bandwidth. Under this model, an exascale system with the same node level $PeakGFlops/NetworkBandwidth$ ratio of today's system (provided enough memory is available to work on a proportionally larger problem) will have similar weak scaling performance. This model neglects differences in system-level network latencies, expected to be significantly smaller in our proposed design due to the use of high-radix routers.

For estimating power at exascale, we use the model described previously in Section IV-B to estimate power for the TOCs, MCs and NoC. We augment this model with power estimates for the LOCs based on projected utilization factor and maximum TDP. For the system network components, including interconnect cabling, we model expected costs in the exascale time frame. In our proposed exascale network design, in contrast to today's networks, the power consumption of the electrical portion of the network (with the exception of the SerDes circuits) consumes energy proportionally to utilization. The overall network power at full bandwidth is 56W per node. For the integrated NIC, we estimate full utilization at 10W and idle power of 7W. Our router chip is estimated to consume around 20W at full utilization and 10W at idle. Finally, we assume 20W of active cooling (fans) per node and voltage regulator efficiency of 95%.

V. Results

This section presents the performance and power results on the proxy-apps described in Section II. First, we evaluate the proxy-apps in the context of several key architectural features. Next, we examine the independent effects of architecture innovations, circuits innovations, and technology scaling on node level power consumption. Finally, we provide projections for application scaling, performance, and power on our proposed exascale system.

A. Architectural Feature Evaluation

As described in Section II, our HPC applications have a variety of differing execution patterns as compared to a dense linear algebra code such as Linpack. This section examine two properties, control divergence and scalarization, to demonstrate the effectiveness of two key microarchitectural innovations that we expect will be necessary for exascale-level power efficiency on HPC workloads.

Warp size and divergence: Existing SIMD style GPU-based systems excel at dense, regular parallel problems common in HPC. However, applications with irregular control flow

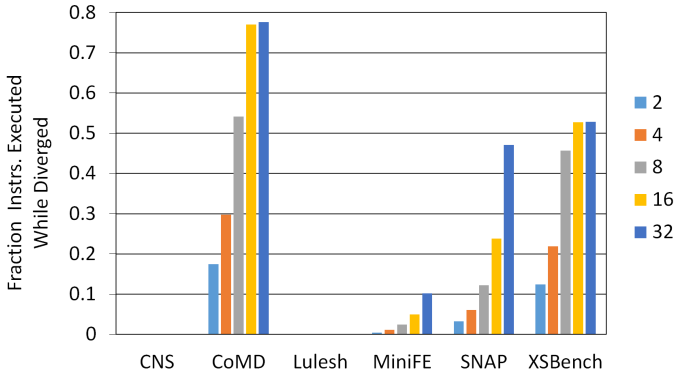

Fig. 4. Impact of warp width on control divergence in our TOC architecture.

can suffer from control-flow divergence, reducing hardware utilization. For these irregular applications, using a smaller warp size will reduce control divergence by decreasing the probability that any given warp will diverge based on control flow. At the limit, a warp width of one will never diverge, but sacrifices the opportunity for energy efficiency due to re-use of instruction fetch and decode.

Figure 4 shows the fraction of instructions each application executes while diverged, as a function of warp width (2 to 32). Contemporary GPUs typically employ a warp width of 32 or 64 threads, which is ideal for dense regular application such as CNS and LULESH that exhibit no control divergence. However, the divergence in applications such as CoMD, SNAP, and XSBench can result in 50% or more of instructions to be issued from diverged warps. For these applications, not only do wider warps not improve performance on current GPU's but they actually reduce throughput due to lanes sitting idle. In the march to exascale, we expect that warp width choices will reverse direction and trend back down to just 4 or 8-wide to provide a balance of performance and energy efficiency on both dense and irregular HPC workloads.

Scalarization: As described in Section III, scalarazation can improve energy consumption by eliminating redundant instruction execution, unnecessary register reads and writes, and duplicate memory accesses. Identifying shared scalar values also reduces register file pressure, increasing the effective available register pool per thread. Static scalarization relies upon the compiler to detect instruction redundancy and produce functionally correct code that has a reduced operand and instruction footprint. Unfortunately, the compiler only has limited knowledge of program execution, which makes identifying all scalarization opportunities difficult. Alternatively, dynamic hardware scalarization can detect duplicate operations with perfect runtime information, but has a high implementation overhead and is not as efficient as static scalarization at saving register file capacity.

Figure 5 shows the effect of scalarization on the proxy-apps. The first bar shows the fraction of instructions executed while converged for our 4-wide warp architecture; warps must be converged for instructions to be statically identified as scalarizable. The second bar shows the dynamically identified scalar opportunity present for each application. This runtime identification of scalar opportunities sets the upper bound on what our static compiler-based implementation could possibly

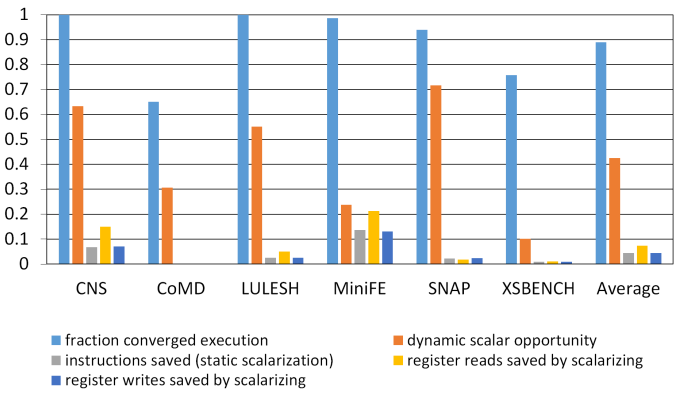

Fig. 5. Opportunity and effectiveness of compiler-based scalarization.

achieve. The third, fourth, and fifth bars show the fraction of instructions, register file reads, and register file writes that can be eliminated via static scalarization. Across the range of applications, compiler-directed scalarization is able to save 5-15% of instructions, 7-21% of register reads, and 4-13% of register writes. While our static compiler based implementation does not currently come close to the dynamic scalarization oracle, there is almost no downside to implementing static scalarization as it simply recaptures energy efficiency that has been left on the floor by current SIMT style architectures.

B. Node-Level Energy Efficiency

To compare the power efficiency of our proposed architecture, we modeled a chip-level 28nm configuration (XGPU28) of our design with characteristics similar to a contemporary GPU (CGPU28) as shown in Table III. These two architectures have similar features such as peak DP floating-point capability, memory bandwidth, and chip size but have large differences in the memory hierarchy organization. CGPU28 has a large register file (almost 4MB for the entire chip) that is used to hold the state for all the threads. XGPU28 uses malleable memory and configures the same SRAM structure for use both as L1 cache and storage of the main register file.

Figure 6 compares the energy efficiency the XGPU28 design normalized to the contemporary CGPU28 design across the suite of proxy-apps. The XGPU28 bars represents our baseline architecture defined in Table III while the remaining columns progressively add the architecture and circuit features described in Section III. Comparing the baseline CGPU28 and XGPU28 architectures, energy efficiency is similar when the workloads have little or no divergence, a result that is not surprising given that the machines have a similar level of peak floating-point performance. For those applications that have significant divergence, XGPU28 substantially outpaces CGPU28 due in large part to increased machine utilization enabled by the narrow warp width described in Figure 4.

As we add architectural and circuit level improvements on top of the baseline XGPU28 architecture, individual improvement per application may have a small or large affect on energy efficiency. The hierarchical register file design is most effective for computationally intense applications that have significant short-term value reuse that is exploitable (CoMD), but less effective for applications that uniformly

TABLE III. PROPOSED 28NM RESEARCH XGPU28 COMPARED TO A CONTEMPORARY 28NM GPU CGPU28.

GPU Resources	CGPU28	XGPU28
Process Technology	28nm	28nm
Voltage	0.9V	0.9V
Clock frequency	744 MHz	1000 MHz
Area	551 mm^2	542 mm^2
Units	15 SMs	48 TOCs
SP ALUs per SM (or TOC)	128	32
DP ALUs per SM (or TOC)	64	16
SP ALUs Total	1920	1536
DP ALUs Total	960	768
Peak SP performance	4.29 TF	4.60 TF
Peak DP performance	1.43 TF	1.53 TF
ORF capacity	-	192 KB
MRF capacity	3840 KB	-
Malleable Memory (MRF + L1)	-	9216 KB
L1 capacity	1680 KB	-
L2 capacity	1536 KB	6144 KB
Warp size	32	4
Off-chip Memory BW	288 GB/sec	250 GB/sec

touch large numbers of registers. Scalarization provides small but measurable benefit to all applications by reducing the cost of redundant calculations and data. Scalarization's overall improvement is hampered by our design choice of a very narrow warp width. If we chose 8-wide warps instead of 4-wide warps improvement from scalarization would nearly double across all applications. The circuit level optimizations that enable lower power supply voltage reduce chip-level power quadratically and have a significant effect on all applications. The increased bandwidth and more efficient IO drivers used when accessing stacked DRAM provides a large improvement to memory intensive applications. Finally, using more efficient wire signaling techniques like GRS [34] reduces the cost of off-chip signaling for all applications.

Most importantly, Figure 6 shows that across this range of diverse applications there is no single magic bullet that can dramatically improve the energy efficiency of all workloads. Reducing warp width is the single largest improvement to help the energy efficiency of irregular applications and is evidence that optimizing HPC architectures for dense codes such as Linpack can sacrifice performance on real-world scientific applications. While none of the features in isolation provides the energy efficiency improvements required to reach exascale, in aggregate all of the applications see improvements ranging from 1.5–4.2x without yet applying the gains from technology scaling or voltage improvements due to aggressive circuit optimizations.

C. System Level Scaling: Performance and Power

This section projects the power and performance for the proxy-apps presented in Section II using the methodology described in Section IV-C. Table IV shows the performance, power, and energy efficiency of the proxy-apps on a baseline system sized similar to the ORNL Titan system using the CGPU28 node configuration. Table V presents the anticipated performance of these same proxy-apps on our projected exascale-class system, which consists of 76,800 7nm nodes and includes all of the architecture and circuit improvements discussed in Section III. For both tables, "Ops" is defined as any math (including floating-point), load/store, or control-flow operation. The tables show that Linpack PetaFlops throughput,

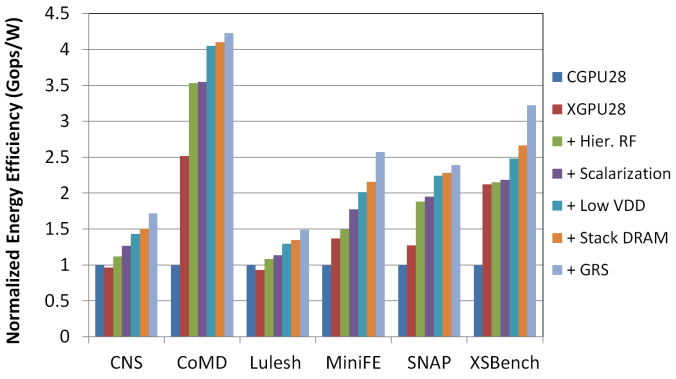

Fig. 6. Relative energy efficiency of single XGPU28 compared to a contemporary design CGPU28.

typically used as the measure of supercomputer sizing, is an incomplete metric for projecting the performance of future HPC applications. Actual applications use the machine in very different ways from Linpack and in general require less power. Much of the difference in power consumption is that the proxy-apps have far lower processor and network utilization than Linpack (Figure 1).

Table V shows that utilization of machine resources varies dramatically across the applications. On Linpack, our projections show that an exascale machine is possible within a power budget of 20MW. While Linpack pushes the total power consumption of the exascale system, the power consumption of the remaining applications ranges from 14-16MW. If provisioned for expected application power, rather than Linpack power, the machine could include 15% more nodes and deliver 15% more performance on real applications. As a result, we believe future systems would be better served by using more conservative real-world workloads when making provisioning decisions or by using application-specific performance metrics such as Figure of Merit (FOM) when making sizing decisions for performance targets [37].

Figure 7 shows the improvement in energy efficiency (GF/W, GOps/W) of our 7nm exascale machine normalized to the Titan-like machine configuration. The two horizontal lines show (1) the energy efficiency goal required to meet exascale level performance (estimated at 25×, rounded up from the 23.4× improvement required over Titan's 2.14 GF/W to reach 50 GF/W) and (2) the portion of the improvement that we can expect to receive if we simply allowed technology scaling to take its course through 2020 (4.3×). Linpack does in fact achieve the energy efficiency improvement goals, while also meeting the performance target set for an exascale system within a 20MW power budget. Of the remaining applications, the computationally dense but control divergent applications show the most improvement, even exceeding the relative improvement of Linpack, though their total Flop throughput is typically one third or less than that of Linpack.

Both SNAP and XSBench fail to reach our energy efficiency targets. For SNAP, this is due to low scaling efficiency. Achieving the energy efficiency goal for SNAP will likely require a more efficient wave-front propagation algorithm used at the MPI level to improve overall node-level utilization. For XSBench, there simply is not enough memory bandwidth compared with the compute resources available, limiting improvement. As XSBench is severely memory bandwidth bound and generates random lookups, it consumes all available bandwidth. While bandwidth utilization is high, bandwidth efficiency can potentially be improved in software or even in hardware with smaller cache lines. Simply providing additional memory bandwidth in the overall system design would help XSBench but hurt the energy efficiency of other applications, reinforcing our assertion that thoughtful trade-offs are required to realize a well-balanced HPC machine.

TABLE IV. PERFORMANCE, POWER, AND ENERGY EFFICIENCY FOR A SUPERCOMPUTER SIMILAR TO ORNL TITAN.

APPS	PetaFlops	PetaOps	MWatts	GFlops/W	GOps/W
CNS	4.61	9.95	5.25	0.88	1.89
CoMD	1.65	13.93	5.32	0.31	2.62
LULESH	4.41	7.82	5.16	0.85	1.51
MiniFE	0.33	4.71	5.08	0.06	0.93
SNAP	0.54	9.15	5.20	0.10	1.76
XSBench	0.52	5.52	5.02	0.10	1.10
LINPACK	17.56	21.95	8.20	2.14	2.68

TABLE V. PERFORMANCE, POWER, AND ENERGY EFFICIENCY OF A TARGET 7NM EXASCALE MACHINE.

APPS	PetaFlops	PetaOps	MWatts	GFlops/W	GOps/W
CNS	369.94	1065.98	16.71	22.13	63.78
CoMD	140.43	1158.51	15.12	9.29	76.61
LULESH	370.57	394.47	16.09	23.04	24.52
MiniFE	21.89	470.73	15.57	1.41	30.23
SNAP	22.02	251.66	16.58	1.33	15.18
XSBench	23.91	179.31	14.81	1.61	12.11
LINPACK	1019.22	1223.06	18.43	55.30	66.36

VI. RELATED WORK

Contemporary high-performance computing systems are based on parallel accelerators to meet performance and power efficiency targets. Among the most popular high performance compute accelerators today are GPU-based architectures, such as NVIDIA Kepler GPGPUs [4], CPU-based architectures such as Intel Xeon Phi coprocessors [38], or other custom architectures such as IBM Blue Gene [39].

Multiple research architectures have been proposed targeting the energy efficiency and throughput objectives for future exascale systems. The X-Caliber project developed a heterogeneous architecture that combines latency optimized cores and compute-near-memory units using 3D die-stacking technology [40]. The Ansgtrom project explored universal technologies for exascale computing, such as a self-aware computational model and distributed factored architecture [41]. Runnemede is designed to achieve maximal energy efficiency unconstrained by the need to support existing programming models or backward compatibility [42]. Its energy-centric approach combines hierarchical simple cores, dedicated hardware for runtime and application code, and near-threshold circuit techniques. NVIDIA Echelon is a GPU-inspired heterogeneous computing architecture that addresses energy-efficiency and memory bandwidth challenges, along with features to facilitate programming of scalable and parallel systems [11].

Substantial work has also been done on workload characterization using HPC proxy applications on existing architectures [9], [10], [12], [13], [14], [15], [16], [18]. Additional research efforts have targeted specific software and architectural co-design for future exascale systems. Balaprakash et al. analyzed

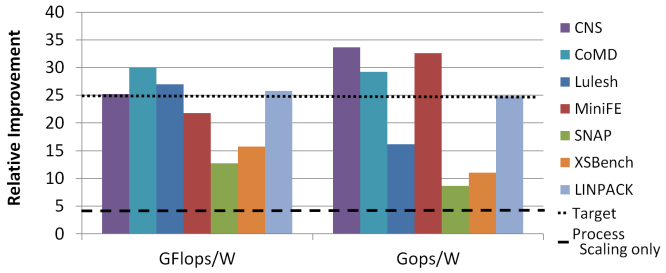

Fig. 7. Expected relative energy efficiency improvements for the proxy-apps.

a broad set of exascale proxy applications and created an application property database [43]. Based on this database the authors develop a statistical exascale computing workload model which is as a function of problem size. This exascale workload was used to to evaluate the proposed processor under memory architecture (PUM). Chan et al. explored hardware software co-design and performance optimization for improved performance of combustion simulation codes by introducing ExaSAT, which is a compiler-driven static analysis and performance modeling framework [18]. The authors showed how such framework can provide an essential speed advantage over simulators while helping with decisions regarding the most suitable cache and memory architectures.

VII. Conclusions

The future of scientific discovery depends on the continued expansion of computing capability. From a hardware perspective, the most substantial barrier to achieving cost-effective exascale is energy efficiency. In this paper we have examined how architecture, circuits, packaging, and process technology scaling can all contribute to improving HPC system-level energy efficiency by the necessary factor of 20 between now and 2020. Technology scaling can still provide a substantial fraction of the target (4.3×), but other sources must account for at least an additional 5×. Our results show that depending on the application, the node level architecture can contribute as much as 3.55× (avg. 1.96×) in energy efficiency, circuits (including aggressive voltage scaling) can contribute as much as 2.5× (avg. 2.2×), and memory packaging technologies can contribute as much as 1.30× (avg. 1.13×).

Our simulation and modeling-based projections show that Linpack can reach exascale at about 20MW, resulting in energy efficiency of 50GFlops/W. The other applications in our suite of HPC proxy-apps show wide variation in their energy-efficiency improvements; SNAP is on the low end with 10-15x improvement and MiniFE is on the high end with 20-25x improvement. In general, real applications will only achieve a fraction of performance of Linpack due to factors including a high fraction of integer operations (CNS and CoMD), memory bandwidth limits (MiniFE and XSBench), or scaling efficiencies (SNAP). Pushing the performance of these applications will require further innovations in algorithms and architecture to improve memory locality, better scaling, and integer execution efficiency.

References

[1] P. Kogge *et al.*, "ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems," University of Notre Dame, Tech. Rep. TR-2008-13, 2008.

[2] "The Opportunities and Challenges of Exascale Computing," http://science.energy.gov/~/media/ascr/ascac/pdf/reports/exascale_subcommittee_report.pdf, 2010.

[3] "ORNL Titan Supercomputer," https://www.olcf.ornl.gov/titan/.

[4] "NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110," http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf, 2012.

[5] S. Huang, S. Xiao, and W. Feng, "On the Energy Efficiency of Graphics Processing Units for Scientific Computing," in *Proceedings of the International Parallel & Distributed Processing Symposium (IPDPS)*, May 2009, pp. 1–8.

[6] A. Petitet, R. Whaley, J. Dongarra, and A. Cleary, "HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers," http://www.netlib.org/benchmark/hpl/, September 2010.

[7] "Green 500," http://www.green500.org/lists/green201311.

[8] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark Silicon and the End of Multicore Scaling," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2011, pp. 356–376.

[9] "Architects of the Exascale Enlist Proxy Apps to Mimic Big Codes," http://ascr-discovery.science.doe.gov/kernels/proxy_apps1.shtml.

[10] R. F. Barrett, M. A. Heroux, P. T. Lin, C. T. Vaughan, and A. B. Williams, "Poster: Mini-applications: Vehicles for Co-design," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[11] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, September/October 2011.

[12] J. Mohd-Yusof and N. Sakharnykh, "Optimizing CoMD: A Molecular Dynamics Proxy Application Study," in *GPU Technology Conference 2014*, March 2014.

[13] I. Karlin, A. Bhatele, J. Keasler, B. L. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang, D. Richards, M. Schulz, and C. Still, "Exploring Traditional and Emerging Parallel Programming Models using a Proxy Application," in *Proceedings of the International Parallel & Distributed Processing Symposium (IPDPS)*, May 2013.

[14] M. A. Heroux, D. W. Doerfler, J. M. W. P. S. Crozier, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving Performance via Mini-applications," Sandia National Laboratories, Tech. Rep. SAND2009-5574, September 2009.

[15] E. N. Houstis, J. R. Rice, N. P. Chrisochoides, H. C. Karathanasis, P. N. Papachiou, M. K. Samartzis, E. A. Vavalis, K. Y. Wang, and S. Weerawarana, "ELLPACK: A Numerical Simulation Programming Environment for Parallel MIMD Machines," in *Proceedings of the International Conference on Supercomputing (ICS)*, September 1990, pp. 96–107.

[16] R. J. Zerr and R. S. Baker, "SNAP: SN (Discrete Ordinates) Application Proxy: Description," Los Alamos National Laboratories, Tech. Rep. LA-UR-13-21070, March 2013.

[17] R. S. Baker and K. R. Koch, "An Sn algorithm for the massively parallel CM200 computer," *Nucl. Sci. and Eng., 128, 312*, 1998.

[18] C. Chan, D. Unat, M. Lijewski, W. Zhang, J. Bell, and J. Shalf, "Software Design Space Exploration for Exascale Combustion Co-design," in *Supercomputing*, ser. Lecture Notes in Computer Science, J. M. Kunkel, T. Ludwig, and H. W. Meuer, Eds. Springer Berlin Heidelberg, 2013, vol. 7905, pp. 196–212.

[19] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz, "XSBench - 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*, September 2014.

[20] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-chip Interconnection Networks," in *Proceedings of the Annual Design Automation Conference*, 2001, pp. 684–689.

[21] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A Unified Graphics and Computing Architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[22] Y. Lee, R. Krashinsky, V. Grover, S. W. Keckler, and K. Asanovic, "Convergence and Scalarization for Data-parallel Architectures," in *International Symposium on Code Generation and Optimization (CGO)*, February 2013, pp. 1–11.

[23] M. Gebhart, S. W. Keckler, and W. J. Dally, "A Compile-time Managed Multi-level Register File Hierarchy," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2011, pp. 465–476.

[24] J. Zalamea, J. Llosa, E. Ayguadé, and M. Valero, "Two-level Hierarchical Register File Organization for VLIW Processors," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2000, pp. 137–146.

[25] M. Gebhart, S. W. Keckler, B. Khailany, R. Krashinsky, and W. J. Dally, "Unifying Primary Cache, Scratch, and Register File Memories in a Throughput Processor," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, December 2012, pp. 96–106.

[26] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, Highly-scalable Dragonfly Topology," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2008, pp. 77–88.

[27] J. Kim, W. J. Dally, and D. Abts, "Flattened Butterfly: A Cost-efficient Topology for High-radix Networks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, May 2007, pp. 126–137.

[28] Z. Zeng, X. Ye, Z. Feng, and P. Li, "Tradeoff Analysis and Optimization of Power Delivery Networks with On-Chip Voltage Regulation," in *Design Automation Conference (DAC)*, June 2010, pp. 831–836.

[29] A. Uht, "Uniprocessor Performance Enhancement Through Adaptive Clock Frequency Control," *IEEE Transactions on Computers*, vol. 54, no. 2, pp. 132–140, Feb 2005.

[30] B. Zimmer, S. O. Toh, H. Vo, Y. Lee, O. Thomas, K. Asanovic, and B. Nikolic, "SRAM Assist Techniques for Operation in a Wide Voltage Range in 28-nm CMOS," *Express Briefs, IEEE Transactions on Circuits and Systems II*, vol. 59, no. 12, pp. 853–857, December 2012.

[31] P. Meinerzhagen, C. Roth, and A. Burg, "Towards Generic Low-Power Area-Efficient Standard Cell-based Memory Architectures," in *International Midwest Symposium on Circuits and Systems (MWSCAS)*, August 2010, pp. 129–132.

[32] M. Sinangil and A. Chandrakasan, "Application-Specific SRAM Design Using Output Prediction to Reduce Bit-Line Switching Activity and Statistically Gated Sense Amplifiers for up to 1.9 Times Lower Energy/Access," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 1, pp. 107–117, January 2014.

[33] A. Banerjee, M. Sinangil, J. Poulton, C. Gray, and B. Calhoun, "A Reverse Write Assist Circuit for SRAM Dynamic Write VMIN Tracking using Canary SRAMs," in *International Symposium on Quality Electronic Design (ISQED)*, March 2014.

[34] J. Poulton, W. Dally, X. Chen, J. Eyles, T. Greer, S. Tell, J. Wilson, and C. Gray, "A 0.54 pJ/b 20 Gb/s Ground-Referenced Single-Ended Short-Reach Serial Link in 28 nm CMOS for Advanced Packaging Applications," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 12, pp. 3206–3218, December 2013.

[35] R. Ho, K. Mai, and M. Horowitz, "Efficient On-chip Global Interconnects," in *Proceedings of the Symposium on VLSI Circuits*, June 2003, pp. 271–274.

[36] JEDEC, "High Bandwidth Memory (HBM) DRAM Specification," *JESD235, Oct*, 2013.

[37] "CORAL collaboration and Benchmark Codes," https://asc.llnl.gov/CORAL-benchmarks.

[38] "Intel Xeon Phi Product Family," http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html.

[39] IBM Blue Gene Team, "Overview of the IBM Blue Gene/P Project," *IBM Journal of Research and Development*, vol. 52, no. 1/2, pp. 199–220, January-March 2008.

[40] "DARPA selects Sandia National Laboratories to design new supercomputer prototype," https://share.sandia.gov/news/resources/news_releases/supercomputer-prototype/#.U03ST_ldV8E, August 2010.

[41] "The MIT Angstrom project: Universal Technologies for Exascale Computing," http://projects.csail.mit.edu/angstrom/node/3, August 2010.

[42] N. P. Carter, A. Agrawal, S. Borkar, R. Cledat, H. David, D. Dunning, J. Fryman, I. Ganev, R. A. Golliver, R. Knauerhase, R. Lethin, B. Meister, A. K. Mishra, W. R. Pinfold, J. Teller, J. Torrellas, N. Vasilache, G. Venkatesh, and J. Xu, "Runnemede: An Architecture for Ubiquitous High-Performance Computing," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, February 2013, pp. 198–209.

[43] P. Balaprakash, D. Buntinas, A. Chan, A. Guha, R. Gupta, S. H. K. Narayanan, A. A. Chien, P. Hovland, and B. Norris, "Exascale Workload Characterization and Architecture Implications," in *Proceedings of the High Performance Computing Symposium*, April 2013, pp. 5:1–5:8.