



FFT, FMM, and multigrid on the road to exascale: Performance challenges and opportunities

Huda Ibeid*, Luke Olson, William Gropp

University of Illinois at Urbana-Champaign, Urbana, IL 61801, United States of America

ARTICLE INFO

Article history:

Received 31 July 2018

Received in revised form 28 May 2019

Accepted 23 September 2019

Available online 9 October 2019

Keywords:

Fast Fourier transform

Fast multipole method

Multigrid

Exascale

Performance modeling

ABSTRACT

FFT, FMM, and multigrid methods are widely used fast and highly scalable solvers for elliptic PDEs. However, emerging large-scale computing systems are introducing challenges in comparison to current petascale computers. Recent efforts (Dongarra et al. 2011) have identified several constraints in the design of exascale software that include massive concurrency, resilience management, exploiting the high performance of heterogeneous systems, energy efficiency, and utilizing the deeper and more complex memory hierarchy expected at exascale. In this paper, we perform a model-based comparison of the FFT, FMM, and multigrid methods in the context of these projected constraints. In addition we use performance models to offer predictions about the expected performance on upcoming exascale system configurations based on current technology trends.

© 2019 Elsevier Inc. All rights reserved.

1. Introduction

Elliptic PDEs arise in many applications in computational science and engineering. Classic examples are found in computational astrophysics, fluid dynamics, molecular dynamics, plasma physics, and many other areas. The rapid solution of elliptic PDEs remains of wide interest and often represents a significant portion of simulation time.

The fast Fourier transform (FFT), the fast multipole method (FMM), and multigrid methods (MG) are widely used fast and highly scalable solvers for elliptic PDEs. The FFT, FMM, and MG methods have been used in a wide variety of scientific computing applications such as particle-in-cell methods, the calculation of long-range (electrostatic) interactions in many-particle systems, such as molecular dynamics and Monte Carlo sampling [3], and in signal analysis. The performance expectations of these methods helps guide algorithmic changes and optimizations to enable migration to exascale systems, as well as to help identify potential bottlenecks in exascale architectures. In addition, modeling helps assess the trade-offs at extreme scales, which can assist in choosing optimal methods and parameters for a given application and specific machine architecture.

Each method has advantages and disadvantages, and all have their place as PDE solvers. Generally, the FFT is used for uniform discretizations, FMM and geometric MG are efficient solvers on irregular grids with local features or discontinuities, and algebraic

MG can handle arbitrary geometries, variable coefficients, and general boundary conditions. The focus of this study is on FFT, FMM, and *geometric* MG, although several observations extend to an algebraic setting as well [7].

One aim of the International Exascale Software Project (IESP) is to enable the development of applications that exploit the full performance of exascale computing platforms [14]. Although these exascale platforms are not yet fully specified, it is widely believed that they will require significant changes in computing hardware architecture relative to the current petascale systems. The IESP roadmap reports that technology trends impose severe constraints on the design of an exascale software. Issues that are expected to affect system software and applications at exascale are summarized as

Concurrency: Future supercomputing performance will depend mainly on increases in system scale. Processor counts of one million or more for current systems [26] whereas exascale systems are likely to incorporate one billion processing cores, assuming GHz technology. As a result, this 1000× increase in concurrency necessitates new paradigms for computing for large-scale scientific applications to ensure extrapolated scalability.

Resiliency: The exponential increase in core counts expected at exascale will lead to increases in the number of routers, switches, interconnects, and memory systems. Consequently, resilience will be a challenge for HPC applications on future exascale systems.

Heterogeneity: As accelerators advance in both performance and energy efficiency, heterogeneity has become a critical

* Corresponding author.

E-mail addresses: hibeid@illinois.edu (H. Ibeid), lukeo@illinois.edu (L. Olson), wgropp@illinois.edu (W. Gropp).

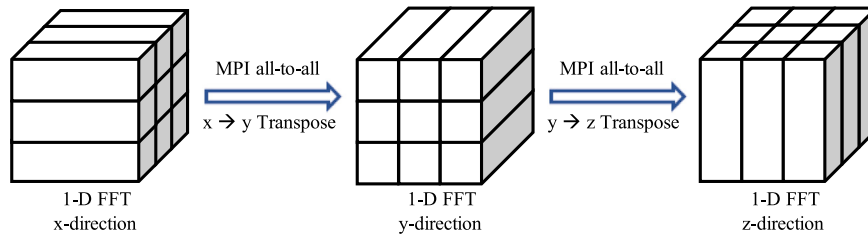


Fig. 1. Illustration of the 3-D FFT calculation flow using pencil decomposition.

ingredient in the pursuit of exascale computing. Exploiting the performance of these heterogeneous systems is a challenge for many methods.

Energy: Power is a major challenge. Current petascale systems would reach the level of 100 MW if extended to exascale.¹ This imposes design constraints on both the hardware and software to improve the overall efficiency. Likewise, exascale algorithms need to focus on maximizing the achieved ratio of performance to power/energy consumption (power/energy efficiency), rather than focusing on raw performance alone.

Memory: The memory hierarchy is expected to change at exascale based on both new packaging capabilities and new technologies to provide the memory bandwidth and capacity required at exascale. Changes in the memory hierarchy will affect programming models and optimizations, and ultimately performance.

In this manuscript, we perform model-based comparison of the FFT, FMM, and MG methods vis-à-vis these challenges. We also use performance models to estimate the performance on hypothetical future systems based on current technology trends. The rest of the manuscript is organized as follows. A short description of the FFT, FMM, and MG methods is provided in Section 2. In the following sections, we present, compare, and discuss the performance of these methods relative to the exascale constraints imposed by technology trends. These constraints are: concurrency (Section 4), resiliency (Section 5), heterogeneity (Section 6), energy (Section 7), and memory (Section 8). Observations and conclusions are drawn in Sections 9 and 10, respectively.

2. Methods

In this section we provide a brief description of FFT, FMM, and MG, in order to establish notation and as preamble to the performance analysis.

2.1. Fast Fourier transform

The FFT is an algorithm for computing the N -point Discrete Fourier Transform (DFT) with $\mathcal{O}(N \log N)$ computational complexity. Let $\mathbf{x} = (x_1, x_2, \dots, x_N)$ be a vector of N complex numbers, the 1-D DFT of \mathbf{x} is defined as

$$\hat{x}_k = \sum_{j=1}^N x_j e^{-i \frac{2\pi k j}{N}}. \quad (1)$$

The 3-D FFT is performed as three successive sets of independent 1-D FFTs.

¹ For example, the Piz Daint supercomputer, which is ranked third and tenth on the TOP500 and Green500 lists, respectively, has power efficiency of 10.398 GFLOPs/W. An exascale machine with the same power efficiency will require 96 MW per exaFLOP.

2.1.1. Parallel domain decomposition

To compute the parallel 3-D FFT, the computational domain is decomposed across processors. There are two popular decomposition strategies for parallel computation: the slab decomposition (1-D decomposition) and the pencil decomposition (2-D decomposition).

In the case of a slab decomposition a 3-D array is partitioned into slabs along one axis so that each processor consists of $\frac{3\sqrt{N}}{P} \times \sqrt{3\sqrt{N}} \times \sqrt{3\sqrt{N}}$ points. This decomposition scheme is unsuitable for massively parallel supercomputer as the number of processors that can be used is limited by the number of slabs. In contrast, in pencil decomposition (a 2-D decomposition) a 3-D array is partitioned in two dimensions, which allows the number of processors to increase. Two of the three dimensions of the cube are divided by \sqrt{P} . Hence, each processor has $\frac{3\sqrt{N}}{\sqrt{P}} \times \frac{3\sqrt{N}}{\sqrt{P}} \times \sqrt{3\sqrt{N}}$ points. A pencil decomposition is used in the current analysis.

2.1.2. FFT calculation flow

The pencil decomposition of a 3-D FFT consists of three computation phases separated by two all-to-all communication phases. Each computation phase computes $\sqrt{3\sqrt{N}} \times \sqrt{3\sqrt{N}}$ 1-D FFTs of size $\sqrt{3\sqrt{N}}$ in parallel. Each all-to-all communication requires $\mathcal{O}(\sqrt{P})$ exchanges for the transpose between pencil-shaped subdomains on P processes. This calculation flow is illustrated in Fig. 1.

The solution of the Poisson equation $-\Delta u = f$ based on FFT is

$$x = \hat{x}^{-1}(\hat{f}/|k|^2), \quad (2)$$

where \hat{f} is the Fourier transform of f and \hat{x}^{-1} is the inverse Fourier transform of x . Thus, solving the Poisson equation using Fourier transform can be broken down into three steps: (1) compute the FFT of f ; (2) scale \hat{f} by $|k|^2$ in Fourier space; and (3) compute the inverse Fourier transform of the result.

2.2. Fast multipole method

N -body problems are used to simulate physical systems of particle interactions under a physical or electromagnetic field [20]. The N -body problem can be represented by the sum

$$f(y_j) = \sum_{i=1}^N w_i K(y_j, x_i), \quad (3)$$

where $f(y_j)$ represents a field value evaluated at a point y_j that is generated by the influence of sources located at the set of centers $\{x_i\}$. $\{x_i\}$ is the set of source points with weights given by w_i , $\{y_j\}$ is the set of evaluation points, and $K(y, x)$ is the kernel that governs the interactions between evaluation and source points.

The direct approach to simulate the N -body problem evaluates all pair-wise interactions among the points which results in a computational complexity of $\mathcal{O}(N^2)$. This complexity is prohibitively expensive even for modestly large data sets. For simulations with large data sets, many faster algorithms have been

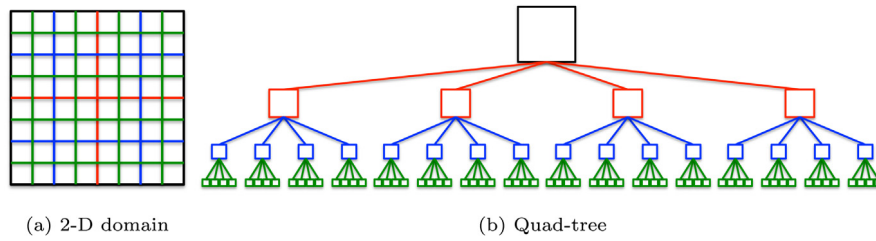


Fig. 2. Decomposition of a 2-D computational domain into a quad-tree.

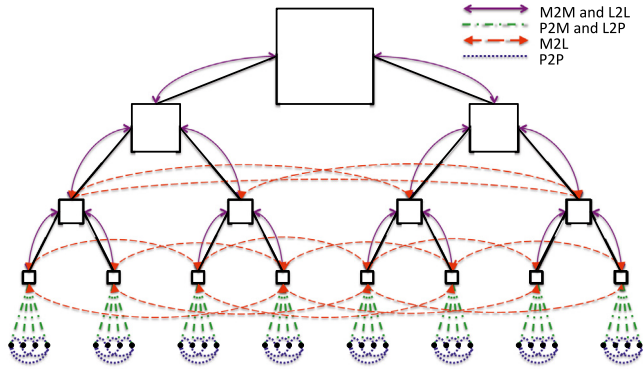


Fig. 3. Illustration of the FMM kernels: P2M (Point-to-Multipole), M2M (Multipole-to-Multipole), M2L (Multipole-to-Local), L2L (Local-to-Local), L2P (Local-to-Point), and P2P (Point-to-Point).

invented, e.g., tree code [5] and fast multipole methods [20]. The fast algorithms cluster points at successive levels of spatial refinement. The tree code clusters the far points and achieves $\mathcal{O}(N \log N)$ complexity. The further apart the points, the larger the interaction groups into which they are clustered. On the other hand, FMM divides the computational domain into near-domain and far-domain and computes interactions between clusters by means of local and multipole expansions, providing $\mathcal{O}(N)$ complexity. Other N -body approaches follow a similar strategy [6,9]. FMM is more than an N -body solver, however. Recent efforts to view the FMM as an elliptic PDE solver have opened the possibility to use it as a preconditioner for even a broader range of applications [22].

2.2.1. Hierarchical domain decomposition

The first step of the FMM algorithm is the decomposition of the computational domain. This spatial decomposition is accomplished by a hierarchical subdivision of the space associated with a tree structure. The 3-D spatial domain of FMM is represented by oct-trees, where the space is recursively subdivided into eight boxes until the finest level of refinement or “leaf level”. Fig. 2 illustrates an example of a hierarchical space decomposition for a 2-D domain that is associated with a quad-tree structure.

2.2.2. The FMM calculation flow

The FMM calculation begins by transforming the mass/charge of the source points into multipole expansions by means of a Point-to-Multipole kernel (P2M). Then, the multipole expansions are translated to the center of larger boxes using a Multipole-to-Multipole kernel (M2M). FMM calculates the influence of the multipoles on the target points in three steps: (1) translation of the multipole expansions to local expansions between well-separated boxes using a Multipole-to-Local kernel (M2L); (2) translation of local expansions to smaller boxes using a Local-to-Local kernel (L2L); and (3) translation of the effect of local

Table 1 Amount of communication in FMM.

	Boxes to send/level
Global M2L	26×8
Local M2L	$(2^i + 4)^3 - 8^i$
Local P2P	$(2^i + 2)^3 - 8^i$

expansions in the far field onto target points using a Local-to-Point kernel (L2P). All-pairs interaction is used to calculate the near field influence on the target points by means of a Point-to-Point kernel (P2P). Fig. 3 illustrates the FMM main kernels: Point-to-Multipole (P2M), Multipole-to-Multipole (M2M), Multipole-to-Local (M2L), Local-to-Local (L2L), Local-to-Point (L2P), and Point-to-Point (P2P). The dominant kernels of the FMM calculation are P2P and M2L.

2.2.3. FMM communication scheme

In this study, we adopt a tree structure that is similar to the one described in [1,21] where FMM uses a separate tree structure for the local and global trees. Each leaf of the global tree is a root of a local tree for a particular MPI process. Therefore, the depth of the global tree depends only on the number of processes P and grows with $\log_8(P)$ in 3-D. Each MPI process stores only the local tree, which depth grows with $\log_8(N/P)$, and communicates the halo region at each level of the local and global tree. Table 1 shows the number of boxes that are sent at the “Global M2L”, “Local M2L”, and “Local P2P” phases where i refers to the level in the local tree and 26 is the number of nearest neighbors.

2.3. Multigrid

Multigrid methods are among the most effective solvers for a wide range of problems. They target the solution of a sparse linear system $Ax = b$ with N unknowns in a computational complexity of $\mathcal{O}(N)$. The basic idea behind MG is to use a sequence of coarse grids to accelerate convergence of the fine grid solution. The building blocks of the multigrid method are the smoothing, restriction, and interpolation operators. These are usually 3-D stencil operations on a structured grid in the case of geometric multigrid (GMG) and sparse matrix-vector multiplications (SpMV) in algebraic multigrid (AMG). In the current study, multigrid refers to the geometric multigrid.

The V-cycle, shown in Fig. 4, is the standard process of a multigrid solver. Starting at the finest structured grid, a smoothing operation is applied to reduce high-frequency errors followed by a transfer of the residual to the next coarser grid. This process is repeated until the coarsest level is reached, at which point the linear system is solved with a direct solver. The error is then interpolated back to the finest grid. The V-cycle is mainly dominated by the smoothing and residual operations on each level.

A multigrid solver is constructed by repeated application of a V-cycle. The number of V-cycles required to reduce the norm of the error by a given tolerance ϵ is estimated by

$$itr_{MG} = \frac{\log \epsilon}{\log \rho}, \tag{4}$$

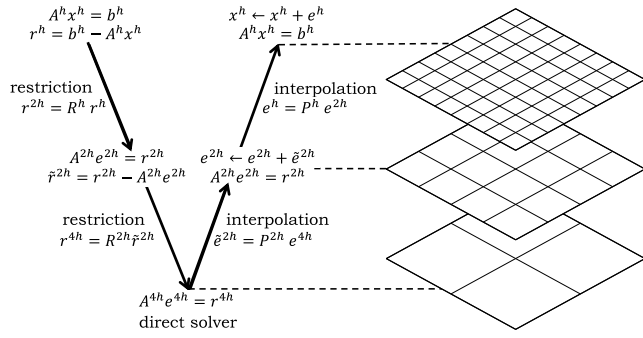


Fig. 4. Illustration of the multigrid V-cycle.

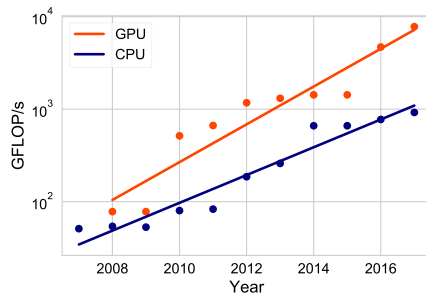
where ρ is the convergence rate. Generally, the convergence rate is bounded by $((\kappa - 1)/\kappa)^\mu$ where μ is the number of smoothing steps and κ is the condition number of the matrix A [4].

3. Exascale projection

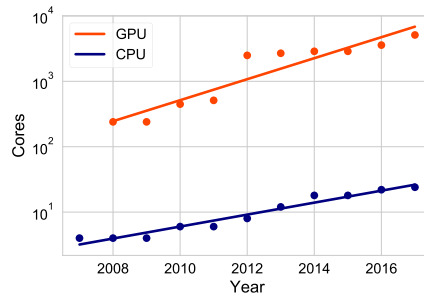
In this paper we consider exascale systems built from hypothetical processors based on extrapolating current technology trends. This section describes how we project these hypothetical CPU-based and GPU-based exascale systems. Similar concept was applied in 2010 by [13].

We collect CPUs and GPUs peak performance, memory bandwidth, and number of cores per processor for the period 2007–2017. Linear regression is then used to find the doubling-time estimate for each parameter, as shown in Fig. 5. For the network link bandwidth, we begin with the data collected in [13], which covers the period 1986–2012. We then collect the same data for systems that made the TOP500 list since 2012.

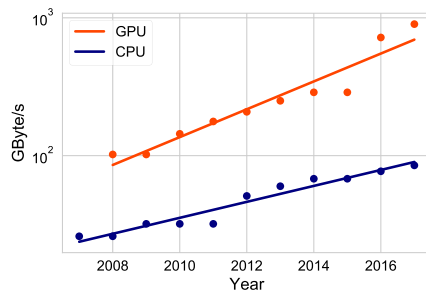
Table 2 shows processor architecture projections, from starting values on the Argonne National Laboratory’s Cooley (GPU-based) and KAUST’s Shaheen II system (CPU-based), both delivered in 2015. We assume that in 2025, we will be able to build a 7 exaFLOP/s (double-precision) system, as projected in



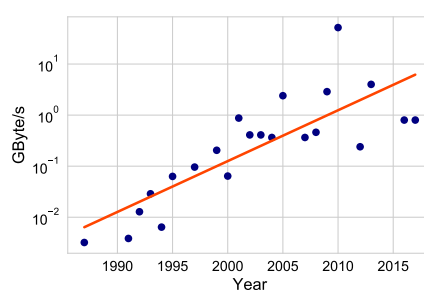
(a) Processor performance



(b) Cores per processor



(c) Memory bandwidth



(d) Link bandwidth

Fig. 5. Hardware trends.

Fig. 6. The value “Processors” count in Table 2 is scaled to reflect this performance.

4. Concurrency

To gain some insight into the solvers’ performance on the massively concurrent systems expected at exascale, we derive analytical performance models that include computation and both intra- and inter-node communication costs. The intra-node communication along with the computation cost account for the single node performance which is a critical building-block in scalable parallel programs, whereas the inter-node term reflects the impact of network communication on the scalability.

In this section we develop performance models for FFT, FMM, and MG on P nodes for a total problem size of $N = \sqrt[3]{N} \times \sqrt[3]{N} \times \sqrt[3]{N}$. Throughout, the computation time is defined as the total number of floating-point operations, multiplied by the time per floating-point operation, t_c , in seconds. Memory movement is modeled as the total data fetched into fast memory, multiplied by the memory bandwidth inverse (β_{mem}) in units of seconds per element. Assuming arithmetic and memory operations are not overlapped, the total execution time T_{exe} is given by

$$T_{\text{exe}} = T_{\text{comp}} + T_{\text{mem}}, \quad (5)$$

and with overlap, T_{exe} is given by

$$T_{\text{exe}} \approx \max(T_{\text{comp}}, T_{\text{mem}}), \quad (6)$$

where T_{comp} is the computation time and T_{mem} is the time spent transferring data in a two-level memory hierarchy between the main memory and cache. T_{exe} in (6) can be rewritten as

$$T_{\text{exe}} = n_{\text{FLOP}} \cdot t_c \cdot \max\left(1, \frac{B_\tau}{AI}\right), \quad (7)$$

where n_{FLOP} is the number of FLOPs, n_{mem} is the number of main memory operations, $B_\tau \equiv \beta_{\text{mem}}/t_c$ is the processor time balance, and $AI \equiv n_{\text{FLOP}}/n_{\text{mem}}$ is the arithmetic intensity. In order to minimize the execution time, AI must be larger than B_τ . This condition is referred to as the balance principle.

Table 2
Processor architecture projections.

Parameter		2015 values (in years)	Doubling time increase factor	10-year ^a	Value
Processor peak	$1/t_{\text{cpu}}$	588.8 GF/s	2.0	32×	18.8 TF/s
	$1/t_{\text{gpu}}$	1.45 TF/s	1.47	111.6×	161.8 TF/s
Memory bandwidth	$1/\beta_{\text{cpu}}$	68 GB/s	5.2	3.8×	258 GB/s
	$1/\beta_{\text{gpu}}$	240 GB/s	2.98	10.2×	2.4 TB/s
Cores	ρ_{cpu}	16	3.29	8.2×	132
	ρ_{gpu}	2496	1.87	40.7×	101.6k
Fast memory	Z_{cpu}	40 MB	2.0	32.0×	1.3 GB
	Z_{gpu}	1.5 MB			48 MB
Line size	L_{cpu}	64 B	10.2	2.0×	128 B
	L_{gpu}	128 B			256 B
Link bandwidth	$1/\beta_{\text{link}}$	10 GB/s	3.0	10×	100 GB/s
Machine peak	R_{peak}	7 PF/s	1.0	1000.0×	7 EF/s
Processors ($R_{\text{peak}} \times t$)	P_{cpu}	11,889	2.01	31.3×	372k
	P_{gpu}	4828	3.15	9×	43.3k

^aThe 10-year increase factor is calculated using $2^{10/\text{Doubling Time}}$.

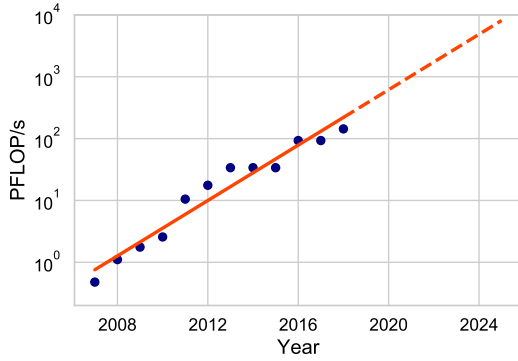


Fig. 6. Top machine on the TOP500 list by year [26].

Inter-node communication cost is modeled using the postal model or α - β_{link} model for communication, where α represents communication latency, β_{link} is the send time per element over the network (inverse the link bandwidth). Using this basic model, communication cost can be represented as

$$T_{\text{net}} = m\alpha + n\beta_{\text{link}}, \quad (8)$$

where m and n are the maximum number of messages and total number of elements sent by a process, respectively.

4.1. Fast Fourier transform

4.1.1. Computation costs

The 1-D Cooley–Tukey FFT of size $\sqrt[3]{N}$ consists of approximately $(5\sqrt[3]{N} \log \sqrt[3]{N})$ floating-point operations. Hence, the total computation time of the 3-D FFT is

$$T_{\text{comp,FFT}} = 3 \cdot \frac{5N \log \sqrt[3]{N}}{P} \cdot t_c, \quad (9)$$

This model accounts for the three computational phases where each phase consists of $\sqrt[3]{N} \times \sqrt[3]{N}$ 1-D FFTs performed in parallel.

4.1.2. Memory access costs

For a cache with size Z and cache-line length L in elements, a cache-oblivious $\sqrt[3]{N}$ -point 1-D FFT incurs $\Theta(1 + \frac{\sqrt[3]{N}}{L}(1 + \log_z \sqrt[3]{N}))$ cache misses, for each transferring line of size L [13,16]. This bound is optimal, matching the lower bound by Hong and Kung

[23] when $\sqrt[3]{N}$ is an exact power of two. Thus, the time spent moving data between the main memory and a processor in the 3-D FFT is given by

$$T_{\text{mem,FFT}} = 3 \cdot \frac{N \log_z \sqrt[3]{N}}{P} \cdot \beta_{\text{mem}}. \quad (10)$$

4.1.3. Network communication costs

In the pencil decomposed 3-D FFT, each processor performs two all-to-all communications with \sqrt{P} other processors sending a total of $\frac{N}{P}$ data points at each communication phase. Hence, the FFT inter-node communication time is approximated by

$$T_{\text{net,FFT}} = 2 \cdot (\sqrt{P} \cdot \alpha + \frac{N}{P} \cdot \beta_{\text{link}}), \quad (11)$$

where the factor of two accounts for the two communication phases. Since a fully connected network is unlikely at exascale, a more realistic estimation of the communication cost must include the topology of the interconnect [13]. For example, on a 3-D torus network without task-aware process placement, the communication time is bounded by the bisection bandwidth $\frac{P^{2/3}}{\beta_{\text{link}}}$. Thus

$$T_{\text{net,FFT}} = 2 \cdot (\sqrt{P} \cdot \alpha + \frac{N}{P^{2/3}} \cdot \beta_{\text{link}}). \quad (12)$$

4.2. Fast multipole method

In this section, we present analytical models for the two phases of FMM that consume most of the calculation time: P2P and M2L. We assume a nearly uniform points distribution and therefore a full oct-tree structure.

4.2.1. Computation costs

P2p. Assuming q points per leaf box, the computational complexity of the P2P phase is $27q^2 \frac{N}{q}$ where 27 is the number of neighbors including the box itself. This leads to a computation cost of

$$T_{\text{comp,P2P}} = 27 \cdot \frac{qN}{P} \cdot t_c. \quad (13)$$

M2l. The asymptotic complexity of the M2L phase depends on the order of expansion k and the choice of series expansion. Table 3 shows the asymptotic arithmetic complexity with respect to k for different expansions used in fast N -body methods [28,29].

Table 3

Asymptotic arithmetic complexity with respect to the order of expansion k for the different series expansions (3-D).

Type of expansion	Complexity
Cartesian Taylor	$\mathcal{O}(k^6)$
Cartesian Chebychev	$\mathcal{O}(k^6)$
Spherical harmonics	$\mathcal{O}(k^4)$
Spherical harmonics+rotation	$\mathcal{O}(k^3)$
Spherical harmonics+FFT	$\mathcal{O}(k^2 \log^2 k)$
Planewave	$\mathcal{O}(k^3)$
Equivalent charges	$\mathcal{O}(k^4)$
Equivalent charges+FFT	$\mathcal{O}(k^3 \log k)$

The kernel-independent FMM (KIFMM) [27], which uses equivalent charges and FFT, has a more precise operations count of $k^3 \log k + 189k^3$ [10]. Hence, the M2L phase of the KIFMM has a total computation cost of

$$T_{\text{comp},\text{M2L}} = \frac{Nk^3 \log k}{q \cdot P} \cdot t_c + 189 \cdot \frac{Nk^3}{q \cdot P} \cdot t_c, \quad (14)$$

where 189 is the number of well-separated neighbors per box ($6^3 - 3^3 = 189$).

Another state-of-the-art FMM implementation is exaFMM [28] which uses Cartesian series expansion. ExaFMM has operations count of $189k^6$. Hence

$$T_{\text{comp},\text{M2L}} = 189 \cdot \frac{Nk^6}{q \cdot P} \cdot t_c. \quad (15)$$

4.2.2. Memory access costs

As shown in [10], the outer loops of the P2P and M2L computations can be modeled as sparse matrix–vector multiplies. A cache-oblivious algorithm [8] for multiplying a sparse $H \times H$ matrix with h non-zeros by a vector establishes an upper bound on cache misses in the SpMV as

$$\mathcal{O}\left(\frac{h}{L} + \frac{H}{Z^{1/3}}\right), \quad (16)$$

for each transferring line of size L .

P2p. Applying (16) gives an upper bound on the number of cache misses for the P2P phase as follows

$$Q_{\text{P2P}} \leq 4 \cdot \frac{N}{L \cdot P} + b_{\text{P2P}} \cdot \frac{N/q}{L \cdot P} + 4 \cdot \frac{N}{L \cdot P} + \frac{N/q}{\left(\frac{Z}{4q}\right)^{1/3} \cdot P}, \quad (17)$$

where b_{P2P} is the average number of source boxes in the neighbor list of a target leaf box ($b_{\text{P2P}} = 26$ for an interior box in a uniform distribution). The first two terms on the right-hand side of (17) refer to read access for the source boxes and the neighbor lists for each target box, while the third term refers to the update access for the target leaf box potentials. In P2P communication, coordinates and values of every point belonging to the box must be sent, resulting in a multiplication factor of four. We model the dominant access time as

$$T_{\text{mem},\text{P2P}} = \frac{N}{P} \cdot \beta_{\text{mem}} + \frac{NL}{(Z^{1/3}q^{2/3}) \cdot P} \cdot \beta_{\text{mem}}. \quad (18)$$

M2L. Applying (16) for the M2L phase gives an upper bound on the number of cache misses as follows

$$Q_{\text{M2L}} \leq \frac{(b_t + bs)f(k)}{L} + \frac{b_{\text{M2L}}b_t}{L} + \frac{b_t}{\left(\frac{Z}{f(p)}\right)^{1/3}}, \quad (19)$$

where b_t is the number of target boxes, b_s is the number of source boxes, b_{M2L} is the average number of source boxes in the well-separated list of a target box ($b_{\text{M2L}} = 189$ for an interior box in a uniform distribution), $f(k)$ is the asymptotic complexity given in

Table 3, and \bar{Z} is the effective cache size. For \bar{Z} , we assume that all possible M2L translation operators ($7^3 - 3^3 = 316$) are computed and stored in the cache, leading to $\bar{Z} = Z - 316 \cdot f(k)$.

Considering the higher order terms, the memory access cost of the M2L phase can be approximated by

$$T_{\text{mem},\text{M2L}} = \frac{Nf(k)}{q \cdot P} \cdot \beta_{\text{mem}} + \frac{Nf(k)^{1/3}L}{q\bar{Z}^{1/3} \cdot P} \cdot \beta_{\text{mem}}. \quad (20)$$

4.2.3. Network communication costs

P2p. The P2P communication is executing only at the lowest level of the FMM tree where each node communicates with its 26 neighbors. In total, each node communicates one layer of halo boxes which create a volume of $(2^l + 2)^3 - 8^l$ where $l = \log_8(N/P)$. Using the α - β_{link} model, the inter-node communication cost of the P2P phase can be represented by

$$T_{\text{net},\text{P2P}} = 26\alpha + n_{\text{P2P}}\beta_{\text{link}}, \quad (21)$$

where n_{P2P} is the number of elements in $\left(\left(\left(\frac{N}{P}\right)^{\frac{1}{3}} + 2\right)^3 - \frac{N}{P}\right)$ leaf boxes.

M2L. Similar to the P2P phase, the number of communicating nodes in the M2L phase is always the 26 neighbors. To use the α - β_{link} model, we estimate the amount of data that is sent at each level of the FMM hierarchy. Table 1 shows the number of boxes that are sent at the ‘‘Global M2L’’, ‘‘Local M2L’’, and ‘‘Local P2P’’ phases where i refers to the level in the local tree. Thus, the communication cost of the M2L phase at level l is represented by

$$T_{\text{net},\text{M2L}}^l = 26\alpha + n_{\text{M2L}}^l \beta_{\text{link}}, \quad (22)$$

where n_{M2L}^l is the number of elements sent at level l .

4.3. Multigrid

The basic building blocks of the classic geometric multigrid algorithm are all essentially stencil computations. In this section, the multigrid solve time is modeled as the sum of the time spent smoothing, restricting, and interpolating at each level as follows

$$T_{\text{solve}}^{\text{MG}} = T_S + T_R + T_I, \quad (23)$$

where T_S , T_R , and T_I are the smoothing, restriction, and interpolation times, respectively.

In the classical multigrid there are more grid points than processors at fine levels. Hence, all processors are active. On coarse grids, however, there are fewer grid points than processors. Therefore, some processors execute on one grid point while others are idle. Some approaches to alleviate this problem include redistributing the coarsest problem to a single process and redundant data distributions. We assume a naive multigrid implementation where the number of points decreases by a constant factor γ in each dimension after each restriction operation. For simplicity of analysis, we assume restriction and interpolation require only communication with neighbors [18].

4.3.1. Computation costs

The smoother is a repeated stencil application. Each smoothing step is performed η_1 times before restriction and η_2 times after interpolation. Thus, the computation cost on a seven-point stencil can be approximated by

$$T_{\text{comp},S} = 7\eta \cdot \left(\sum_{i=0}^{\lfloor \log_{\gamma} \frac{N}{P} \rfloor} \frac{N}{\gamma^{3i}P} + \sum_{i=\lfloor \log_{\gamma} \frac{N}{P} \rfloor + 1}^{\lfloor \log_{\gamma} N \rfloor} 1 \right) \cdot t_c, \quad (24)$$

where the number of smoothing phases $\eta = \eta_1 + \eta_2$. In our analysis we assume one smoothing step before restricting and one smoothing step after interpolation.

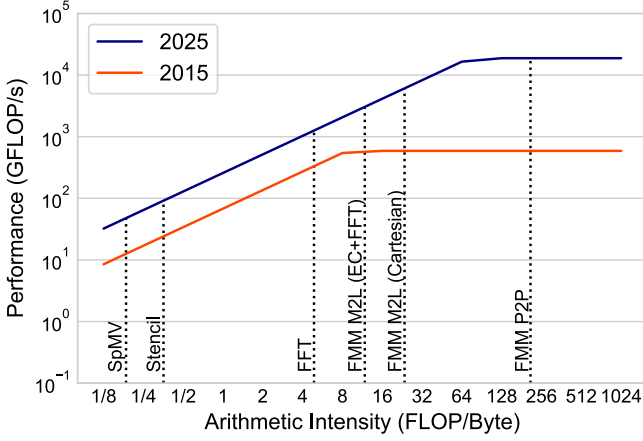


Fig. 7. Roofline model and computation intensity of various phases of the FFT, FMM, and MG methods with $N = (32K)^3$.

4.3.2. Memory access costs

A cache oblivious algorithm for 3-D stencil computations incurs at most $\mathcal{O}(N/Z^{1/3})$ cache misses for each transferring line of size L [17]. This number of cache misses matches the lower bound of Hong and Kung [23] within a constant factor. We apply this bound to the stencil computations within the multigrid method. Therefore, the memory access cost of smoothing can be represented by

$$T_{\text{mem},S} = 7\eta \cdot \left(\sum_{i=0}^{\lfloor \log_{\gamma} \frac{N}{P} \rfloor} \frac{N}{\gamma^{3i}P} + \sum_{i=\lfloor \log_{\gamma} \frac{N}{P} \rfloor + 1}^{\lfloor \log_{\gamma} N \rfloor} 1 \right) \frac{L}{Z^{1/3}} \cdot \beta_{\text{mem}}. \quad (25)$$

4.3.3. Network communication costs

Communication within the V-cycle takes the form of nearest-neighbor halo exchanges. In the 3-D multigrid, each processor communicates with its six neighbors where the amount of data exchanged decreases by a factor of c^2 on each subsequent level. Thus, the communication time at level l is given by

$$T_{\text{net},S}^l = T_{\text{net},R}^l = T_{\text{net},I}^l = 6\alpha + \frac{6(N/P)^{2/3}}{\gamma^{2l}} \cdot \beta_{\text{link}}. \quad (26)$$

4.4. Model interpretation

4.4.1. Roofline model

Arithmetic intensity (AI) is the ratio of total floating-point operations (FLOPs) to total data movement (Bytes). Applications with low arithmetic intensity are typically memory-bound. This means their execution time is limited by the speed at which data can be moved rather than the speed at which computations can be performed, as in compute-bound applications. Hence, memory-bound applications achieve only a small percentage of the theoretical peak performance of the underlying hardware.

The roofline model can be used to assess the quality of attained floating-point performance (GFLOP/s) by combining machine peak performance, machine sustained bandwidth, and arithmetic intensity as follows

$$\text{Attainable (GFLOP/s)} = \min(\text{Peak (GFLOP/s)}, \text{Memory BW} \times \text{AI}). \quad (27)$$

Fig. 7 shows a roofline model along with the arithmetic intensity of various phases of the FFT, FMM, and MG methods. The ridge point on the roofline model is the processor balance point. All intensities to the left of the balance point are memory bound,

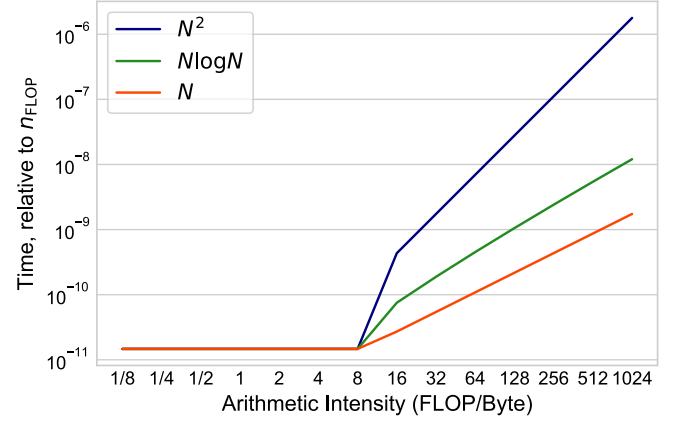


Fig. 8. Execution time (normalized to n_{FLOP}) for various computational complexities.

whereas all to the right are compute bound. Comparing the three methods shows that the FMM computations have higher arithmetic intensity due to its matrix-free nature. On the other hand, SpMV and stencil operations, which are the basic building blocks of the classic algebraic and geometric multigrid methods, have low arithmetic intensities. The 3-D FFT has an intermediate arithmetic intensity that grows slowly with the problem size.

In order to understand the relation between computation time and arithmetic intensity, the algorithmic efficiency must be taken into account. Using (7), Fig. 8 shows that computation time is independent of the algorithmic computational complexity up until the processor balance point. Beyond that point, it increases as a function of the algorithmic complexity.

4.4.2. Projecting forward

Fig. 7 also shows the roofline model of a possible future CPU processor. The characteristics of the processor are based on extrapolating historical technology trends. These trends are summarized in Table 2. From Fig. 7 we observe that although FMM is compute-bound on contemporary systems, it could become memory-bound at exascale.

5. Resiliency

The focus of this section is on the main memory and network errors. Therefore, to assess the resiliency of FFT, FMM, and MG, we quantify the vulnerability of the data structures and communication patterns used within these methods.

For data structures, we use the data vulnerability factor (DVF) introduced in [30]. The DVF for a specific data structure (DVF_d) is defined as

$$DVF_d = FIT \times T_{\text{exe}} \times S_d \times N_{ha}, \quad (28)$$

where FIT is the failure in time (failures per billion hours per Mbit), T_{exe} is the application execution time, S_d is the size of the data structure, and N_{ha} is the number of accesses to the hardware (main memory in this study). To estimate N_{ha} , we use the number of cache misses approximated for FFT, FMM, and MG in Section 4.

The DVF of an application (DVF_a) can be calculated by

$$DVF_a = \sum_{i=1}^D DVF_{d_i}, \quad (29)$$

where D is the number of major data structures in the application.

For communication, we introduce the communication vulnerability factor (CVF) which reflects the communication pattern and

Table 4
Data vulnerability factors of FFT, FMM, and MG.

	2015	2025
FFT	0.003	0.431
FMM	0.376	41.20
MG	0.037	2.101

Table 5
Communication vulnerability factors of FFT, FMM, and MG.

	2015	2025
FFT	0.020	2.934
FMM	3.4e−4	0.004
MG	1.2e−5	4.9e−5

network characteristics. The CVF for a specific kernel (CVF_k) is defined as

$$CVF_k = m \times T_{\text{net}} \times RF_n, \quad (30)$$

where m is the maximum number of messages sent, T_{net} is the application communication time, and RF_n is the network resilience factor defined for uniform deterministic traffic [24] as follows

$$RF_n = \bar{h}p_e + bp_b^2, \quad (31)$$

where \bar{h} is the average route length and p_e is the effective link failure probability given by

$$p_e = ap_a + 2bp_b - bp_b^2, \quad (32)$$

where a is the probability of the occurrence of an event A that can only affect the status of a link, b is the probability of the occurrence of an event B that can affect the status of all the links incident at a node, and p_a and p_b are the probability that events A and B , respectively, can lead to link failure. Here, we define \bar{h} as the diameter of the network formed by P nodes.

For multilevel methods, the CVF is calculated at each level individually. Thus, the application CVF_a is given by

$$CVF_a = \sum_{k=1}^K CVF_k, \quad (33)$$

where K is the number of key kernels in the application and CVF_k is given by

$$CVF_k = \sum_{l=1}^L CVF_k^l, \quad (34)$$

where L is the number of levels.

5.1. Model interpretation

5.1.1. Data structures vulnerability

Table 4 shows the DVF of the FFT, FMM, and MG methods. FMM has random memory access pattern as memory accesses to the tree are random. FFT and MG, on the other had, have template-based memory access pattern where accesses to elements of the data structure mesh follow specific topology or stencil information instead of arbitrarily constructed. Table 4 shows that algorithms with random memory access pattern, such as FMM, have higher DVF than algorithms with template-based memory access pattern, such as FFT and MG. Therefore, the FMM data structures are more sensitive to memory errors compared to FFT and MG. These observations are consistent with the ones in [30].

5.1.2. Communication vulnerability

The communication vulnerability factors of FFT, FMM, and MG are shown in Table 5. As expected, the all-to-all communication pattern of FFT makes it more sensitive to network failures compared to the hierarchical methods, FMM and MG. The hierarchical nature of FMM and MG reduces the $\mathcal{O}(\sqrt{P})$ communication complexity of FFT to $\mathcal{O}(\log P)$. This communication complexity is likely to be optimal for elliptic problems, since an appropriately coarsened representation of a local forcing must somehow arrive at all other parts of the domain for the elliptic equation to converge.

5.1.3. Projecting forward

For exascale projections, we scale the problem size by the “Cores” 10-year increase factor from Table 2. In Tables 4 and 5, the problem size per processor is $N/P = 32^3$ for 2015 and $N/P = 65^3$ for 2025. We also scale the number of processes by the “Processors” increase factor from $P = 11,889$ to $P = 372k$ and assume that the effective link failure probability p_e remains constant over time. The results show that both the DVF and CVF are expected to increase on future exascale systems with more drastic increase in the DVF.

6. Heterogeneity

In this section we adapt the FFT, FMM, and MG execution models introduced in Section 4 to accelerators. In particular, we consider NVIDIA GPUs. One of the main architectural differences between GPUs and CPUs is the relatively small caches on GPUs which makes reusing data in the fast memory more difficult. Another bottleneck to consider on heterogeneous systems is the PCIe bus. Due to the high compute capability of the GPU, the PCIe bus can have a significant impact on performance. The PCIe transfer time for n elements is given by

$$T_{\text{PCIe}}(n) = n\beta_{\text{PCIe}}, \quad (35)$$

where β_{PCIe} is the I/O bus bandwidth inverse in seconds per element. For FFT, FMM, and MG, the PCIe transfer time is given by

$$T_{\text{PCIe}} = 2 \times \frac{N}{P} \beta_{\text{PCIe}}, \quad (36)$$

where the factor of two accounts for the two ways transfer. Here, we assume that each processor has a direct network connection, optimistically avoiding PCIe channels.

6.1. Model interpretation

6.1.1. Roofline model

Fig. 9 shows roofline models of NVIDIA Tesla GPU and of a possible future GPU processor that is based on extrapolating historical technology trends. Similar to the CPU exascale projection results, Fig. 9 shows that kernels that are compute-bound on contemporary systems could become memory-bound at exascale.

6.1.2. Projecting forward

Using the analytical models, we predict the communication time of FFT, FMM, and MG for large-scale problems on possible future GPU-only and CPU-only exascale systems. The machine characteristics of the exascale systems are based on the trends summarized in Table 2. Fig. 10 shows the communication time of FFT, FMM, and MG split into memory and network access costs. We consider extrapolated GPU-only and CPU-only systems that have the same peak performance of 7 exaFLOPS. The GPU-only system has 43.3 k processors while the CPU-only system has 372 k processors. The results show that all methods spend less

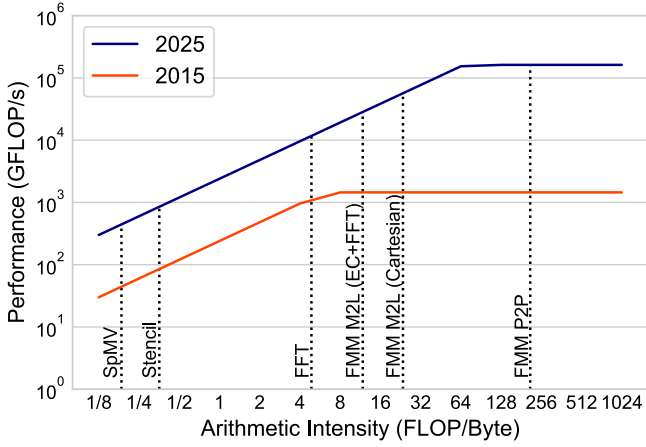


Fig. 9. Roofline model of NVIDIA Tesla GPU and computation intensity of various phases of the FFT, FMM, and MG methods with $N = (32 \text{ k})^3$.

time in both forms of communication on the CPU-only system. However, this system requires almost $8.7\times$ as many processors as the GPU-only system. This cost could be prohibitive.

Fig. 10 shows that the FFT communication time is dominated by the network access cost which is expected given the all-to-all communication pattern of FFT. On the other hand, memory access cost dominates MG communication time. This implies that intra-node communication could become the bottleneck that limits the scalability of MG on exascale systems.

7. Energy

To characterize power and energy efficiency of the FFT, FMM, and MG methods, we use the energy roofline model introduced in [11]. This model bounds power consumption as a function of the total floating-point operations and total amount of data moved. The energy cost (Joules) is defined by

$$E \equiv E_{\text{FLOP}} + E_{\text{mem}} + E_0(T_{\text{exe}}), \quad (37)$$

where E_{FLOP} is the total energy consumption of the computation, E_{mem} is the total energy consumption of memory traffic, and E_0 is a measure of energy leakage as a function of execution time, T_{exe} .

Suppose the energy cost is linear in T_{exe} with a fixed constant power π_0 , (37) can be written as

$$E = n_{\text{FLOP}} \cdot \epsilon_{\text{FLOP}} + n_{\text{mem}} \cdot \epsilon_{\text{mem}} + \pi_0 \cdot T_{\text{exe}}, \quad (38)$$

where n_{FLOP} is the number of FLOPs, n_{mem} is the number of main memory operations, and ϵ_{FLOP} and ϵ_{mem} are fixed energy per computation and per memory operation, respectively. Defining the energy balance $B_\epsilon \equiv \epsilon_{\text{mem}}/\epsilon_{\text{FLOP}}$, the above equation becomes

$$E = n_{\text{FLOP}} \cdot \epsilon_{\text{FLOP}} \cdot \left(1 + \frac{B_\epsilon}{AI} + \frac{\pi_0 \cdot T_{\text{exe}}}{\epsilon_{\text{FLOP}} \cdot n_{\text{FLOP}}}\right), \quad (39)$$

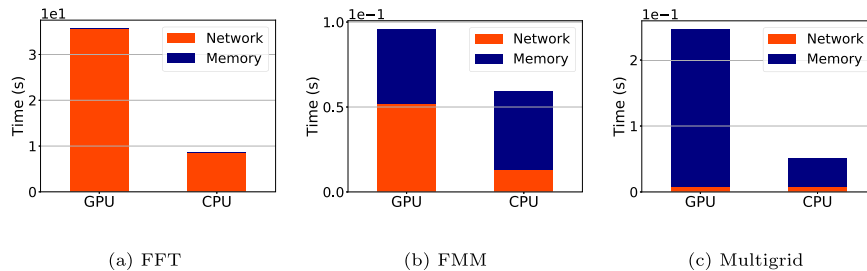


Fig. 10. Exascale projections of the FFT, FMM, and MG methods on GPU- and CPU-only systems with $N = (65 \text{ k})^3$.

Table 6

Component scaling with node size. Ratios are given in reference to 45 nm [15].

Tech node (nm)	Frequency	Voltage	Capacitance	Power
45	1.00	1.00	1.00	1.00
32	1.10	0.93	0.75	0.71
22	1.19	0.88	0.56	0.52
16	1.25	0.86	0.42	0.39
11	1.30	0.84	0.32	0.29
8	1.34	0.84	0.24	0.22

Using (39), Fig. 12 shows the energy roofline model along with the arithmetic intensity of various phases of FFT, FMM, and MG. The figure shows that algorithms with higher arithmetic intensity have better energy efficiency. However, total energy consumption depends heavily on the algorithmic efficiency. Similar to Fig. 8, Fig. 13 shows that total energy consumption is independent of the algorithmic computational complexity for all intensities to the left of the processor balance point whereas energy consumption increases as a function of the algorithmic complexity beyond that point.

7.1. Projecting forward

The classic equation for dynamic power is given by

$$\text{Power}_{\text{dyn}} = \alpha CV^2f, \quad (40)$$

where C is the load capacitance, a physical property of the material, V is the supply voltage, and f is the clock frequency. Hence, ϵ_{FLOP} and ϵ_{mem} become

$$\epsilon_{\text{FLOP}} = t_c \cdot \text{Power}_{\text{dyn}}, \quad (41)$$

and

$$\epsilon_{\text{mem}} = \beta_{\text{mem}} \cdot \text{Power}_{\text{dyn}}. \quad (42)$$

To estimate the energy efficiency of future multicore chips, we use the transistor scaling projection model presented in [15]. This model provides the area, voltage, and frequency scaling factors for technology nodes from 45 nm through 8 nm. These factors are summarized in Table 6. Fig. 11 shows how the energy efficiency is predicted to improve as the node size decreases. Nevertheless, the per-transistor power efficiency improvements have slowed in comparison to the historic rates. Microarchitecture innovations are needed to improve the energy efficiency.

8. Memory

Memory hierarchy is expected to change at exascale based on both new packaging capabilities and new technologies to provide the required bandwidth and capacity. Local RAM and non-volatile-memory (NVRAM) will be available either on or very close to the nodes to reduce wire delay and power consumption.

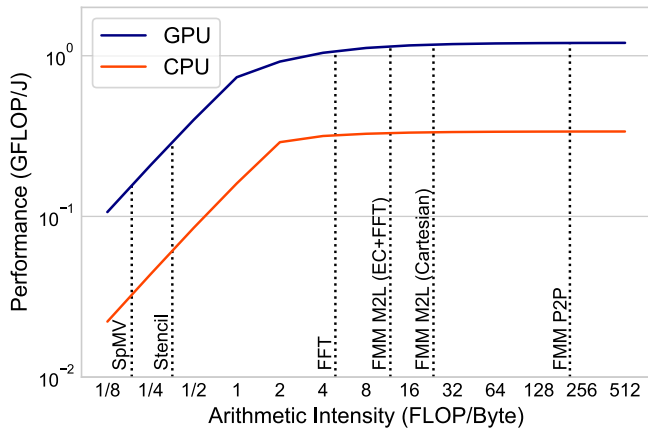


Fig. 11. Energy roofline model of various technology nodes along with the computation intensity of FFT, FMM, and MG.

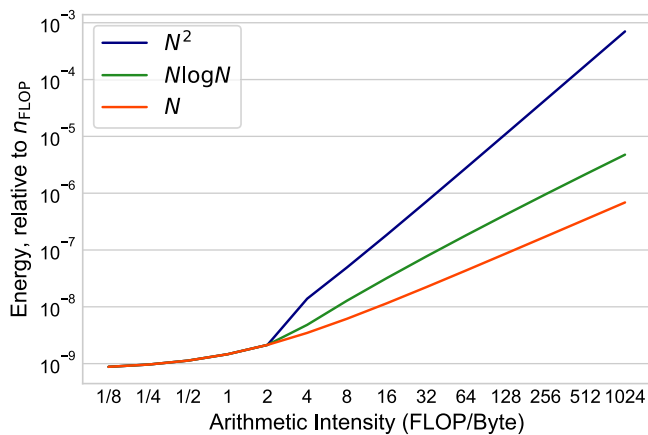


Fig. 12. Energy roofline model of NVIDIA GTX 580 GPU and Intel i7-950 (double-precision) along with the computation intensity of various phases of FFT, FMM, and MG (GPU: $\epsilon_{\text{FLOP}} \approx 212$ pJ per FLOP, $\epsilon_{\text{mem}} \approx 513$ pJ per Byte, $\pi_0 \approx 122$ W, $t_c \approx 5.1$ ps per FLOP, and $\beta_{\text{mem}} \approx 5.2$ ps per Byte; CPU: $\epsilon_{\text{FLOP}} \approx 670$ pJ per FLOP, $\epsilon_{\text{mem}} \approx 795$ pJ per Byte, $\pi_0 \approx 122$ W, $t_c \approx 18$ ps per FLOP, and $\beta_{\text{mem}} \approx 39$ ps per Byte [12]).

One of the leading proposed mechanism to emerge in the memory hierarchy is the 3-D stacked memory which enables DRAM devices with much higher bandwidths than traditional DIMMs (dual in-line memory module).

Deeper memory hierarchy is expected at exascale with each level composed of a different memory technology. One proposed memory hierarchy for exascale systems consists of [2]: a high-bandwidth 3-D stacked memory, such as high bandwidth memory (HBM) standard or hybrid memory cube (HMC) technology, a standard DRAM, and NVRAM memory. Approximate bandwidths and capacities of the proposed memory subsystem are shown in Table 7.

Fig. 14 shows memory-aware roofline models of the different memory technologies. The roofline models can be derived by substituting the bandwidth values from Table 7 into (27). Emerging 3-D stacked DRAM devices, such as HBM and HMC, will significantly increase available memory bandwidth. However, with the exponential increase in core counts, stacked DRAM will only move the memory wall and is unlikely to break through it [25]. Fig. 14 also shows the large difference in attainable performance between different levels of the memory hierarchy. Exascale applications need to exploit data locality and explicitly manage data movement to minimize the cost of memory accesses and to make the most effective use of available bandwidth.

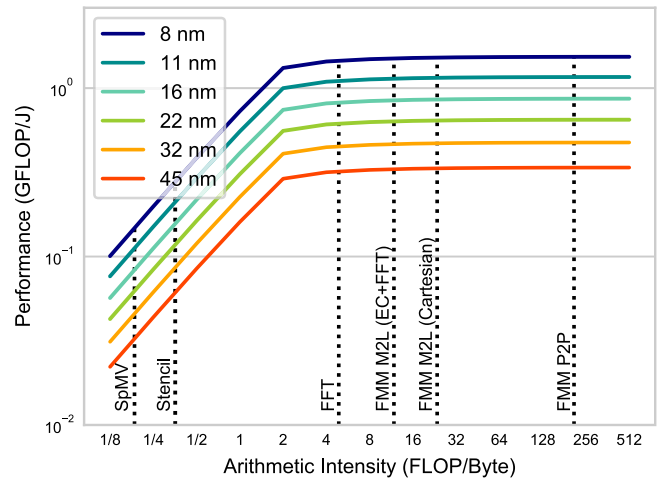


Fig. 13. Energy consumption (normalized to n_{FLOP}) for various computational complexities.

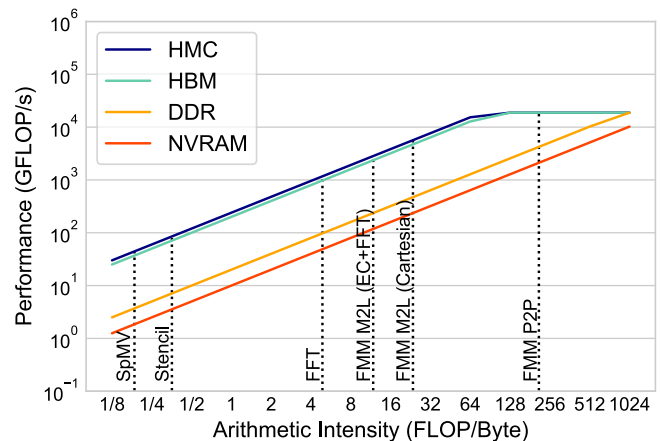


Fig. 14. Memory-aware roofline model of a possible exascale machine along with the computation intensity of various phases of FFT, FMM, and multigrid methods with $N = (32 \text{ k})^3$.

9. Observations

- Algorithms that are known to be compute-bound on current architectures, such as the FMM, could become memory-bound on future CPU- and GPU-based exascale systems.
- Execution time and energy consumption are independent of the algorithmic computational complexity until the processor balance point. They increase as a function of the algorithmic complexity beyond that point.
- Heterogeneous systems are important for energy efficient scientific computing.
- It is well known that GPUs deliver more peak performance and bandwidth relative to high-end CPUs. This performance gap is likely to increase towards exascale, as shown in Fig. 15.
- Emerging 3-D stacked DRAM devices will significantly increase available memory bandwidth. However, with the exponential increase in core counts, stacked DRAM will only move the memory wall and is unlikely to break through it.

10. Conclusions

Recent efforts [14] have identified several constraints in the design of exascale software that include massive concurrency,

Table 7
Approximate bandwidths and capacities of memory subsystem.

	Configuration	Bandwidth	Capacity
Single-Level	HMC	240 GB/s per stack	16 GB per stack
Multi-Level DRAM	HBM	200 GB/s per stack	16 GB per stack
	DDR	20 GB/s per channel	64 GB per DIMM
High capacity memory	NVRAM	10 GB/s	4 – 8 × DRAM

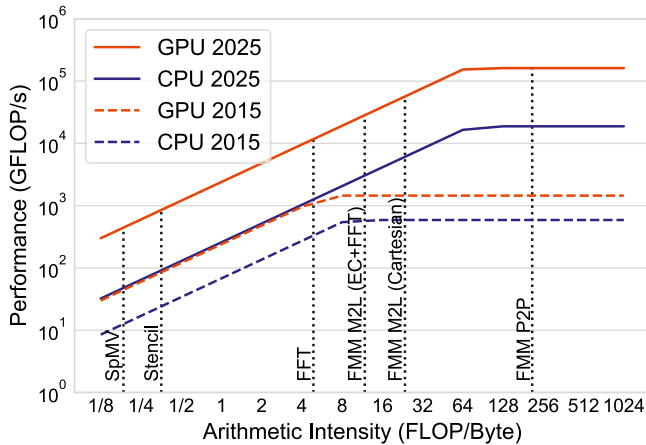


Fig. 15. CPU- and GPU-based roofline models.

resilience management, exploiting the high performance of heterogeneous systems, energy efficiency, and utilizing the deeper and more complex memory hierarchy expected at exascale. In this manuscript, we perform model-based comparison of the FFT, FMM, and MG methods vis-à-vis these challenges. We believe that the importance of each of these challenges is application dependent. This paper provides metrics for researchers to quantify these challenges and their importance relative to the applications of interest.

Modeling FFT, FMM, and MG relative to these challenges has contributed to our understanding of the main steps that must be taken on both application and architecture sides to help overcoming these challenges.

On the application side:

- *Rethink algorithms to reduce memory requirements.* Data movement is the dominant factor that limits performance and efficiency on contemporary architecture. Attainable floating-point performance of memory-bound applications is limited by the memory bandwidth. Furthermore, a significant portion of the energy consumption of modern supercomputers is caused by memory operations. Reducing data movements leads to higher arithmetic intensity, lower memory bandwidth usage, lower energy consumption, and better scalability with the number of cores.
- *Rethink algorithms to improve arithmetic intensity.* High arithmetic intensity is essential for achieving good performance and efficiency. Possible approaches to increase the arithmetic intensity include improving data locality, combining multiple kernels into a single high arithmetic intensity kernel, and reducing the memory footprint by, for example, using matrix-free approaches as in the FMM.
- *Design for sustainability.* Resilience is a major obstacle on the road to exascale. Our projections show that resilience is expected to be a much larger issue on exascale systems than it is on current petascale computers. New resilience paradigms are required.

- *Enable energy-efficient software.* In addition to reducing memory operations and improving arithmetic intensity, power consumption can be reduced at the software side by efficiently exploiting thread level parallelism to ensure balance between performance gains and increases in energy consumption.

On the architectural side:

- *Increasing memory bandwidth.* Radical technology advances are needed to improve local memory bandwidth.
- *Enable energy-efficient computers.* Ongoing research efforts to improve energy efficiency include [19]: dynamic frequency scaling, power-aware applications, energy management throughout the hardware/software stack, and optimization techniques for balancing performance and power. Nevertheless, disruptive technology breakthroughs are still needed to enable energy efficient computers.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

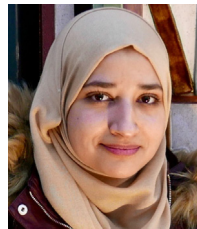
Acknowledgments

The authors would like to thank Prof. David Keyes (KAUST) for many insightful discussions. This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number DE-NA0002374.

References

- [1] M. Abduljabbar, G.S. Markomanolis, H. Ibeid, R. Yokota, D. Keyes, *Communication reducing algorithms for distributed hierarchical n-body problems with boundary distributions*, in: J.M. Kunkel, R. Yokota, P. Balaji, D. Keyes (Eds.), *High Performance Computing*, Springer International Publishing, Cham, 2017, pp. 79–96.
- [2] J.A. Ang, R.F. Barrett, R.E. Benner, D. Burke, C. Chan, J. Cook, D. Donofrio, S.D. Hammond, K.S. Hemmert, S.M. Kelly, H. Le, V.J. Leung, D.R. Resnick, A.F. Rodrigues, J. Shalf, D. Stark, D. Unat, N.J. Wright, *Abstract Machine Models and Proxy Architectures for Exascale Computing*, in: 2014 Hardware-Software Co-Design for High Performance Computing, 2014, pp. 25–32, <http://dx.doi.org/10.1109/Co-HPC.2014.4>.
- [3] A. Arnold, F. Fahrenberger, C. Holm, O. Lenz, M. Bolten, H. Dachsel, M. Halver, I. Kabadshow, F. Gähler, F. Heber, J. Iseringhausen, M. Hofmann, M. Pippig, D. Potts, G. Sutmann, *Comparison of scalable fast methods for long-range interactions*, *Phys. Rev. E* 88 (2013) 063308, <http://dx.doi.org/10.1103/PhysRevE.88.063308>, URL <https://link.aps.org/doi/10.1103/PhysRevE.88.063308>.
- [4] R.E. Bank, C.C. Douglas, *Sharp estimates for multigrid rates of convergence with general smoothing and acceleration*, *SIAM J. Numer. Anal.* 22 (4) (1985) 617–633, <http://dx.doi.org/10.1137/0722038>, arXiv:<https://doi.org/10.1137/0722038>.
- [5] J. Barnes, P. Hut, *A hierarchical $O(N \log N)$ force-calculation algorithm*, *Nature* 324 (1986) 446–449, <http://dx.doi.org/10.1038/324446a0>.
- [6] N. Beams, L. Olson, J. Freund, *A finite element based P³M method for N-body problems*, *SIAM J. Sci. Comput.* 38 (3) (2016) A1538–A1560, <http://dx.doi.org/10.1137/15M1014644>.
- [7] A. Bienz, R.D. Falgout, W. Gropp, L.N. Olson, J.B. Schroder, *Reducing parallel communication in algebraic multigrid through sparsification*, *SIAM J. Sci. Comput.* 38 (5) (2016) S332–S357.

- [8] G.E. Blelloch, P.B. Gibbons, H.V. Simhadri, Low depth Cache-oblivious algorithms, in: Proceedings of the Twenty-Second Annual ACM Symposium on Parallelism in Algorithms and Architectures, in: SPAA '10, ACM, New York, NY, USA, 2010, pp. 189–199, <http://dx.doi.org/10.1145/1810479.1810519>, URL <http://doi.acm.org/10.1145/1810479.1810519>.
- [9] C. Burstedde, L.C. Wilcox, O. Ghattas, P4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, *SIAM J. Sci. Comput.* 33 (3) (2011) 1103–1133, <http://dx.doi.org/10.1137/100791634>.
- [10] A. Chandramowlishwaran, J. Choi, K. Madduri, R. Vuduc, Brief announcement: Towards a communication optimal fast multipole method and its implications at exascale, in: Proceedings of the Twenty-Fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures, in: SPAA '12, ACM, New York, NY, USA, 2012, pp. 182–184, <http://dx.doi.org/10.1145/2312005.2312039>, URL <http://doi.acm.org/10.1145/2312005.2312039>.
- [11] J.W. Choi, D. Bedard, R. Fowler, R. Vuduc, A roofline model of energy, in: Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, in: IPDPS '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 661–672, <http://dx.doi.org/10.1109/IPDPS.2013.77>.
- [12] J. Choi, M. Dukhan, X. Liu, R. Vuduc, Algorithmic time, energy, and power on Candidate HPC compute building blocks, in: 2014 IEEE 28th International Parallel and Distributed Processing Symposium, 2014, pp. 447–457, <http://dx.doi.org/10.1109/IPDPS.2014.54>.
- [13] K. Czechowski, C. Battaglini, C. McClanahan, K. Iyer, P.-K. Yeung, R. Vuduc, On the communication complexity of 3d FFTs and its implications for exascale, in: Proceedings of the 26th ACM International Conference on Supercomputing, in: ICS '12, ACM, New York, NY, USA, 2012, pp. 205–214, <http://dx.doi.org/10.1145/2304576.2304604>, URL <http://doi.acm.org/10.1145/2304576.2304604>.
- [14] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, X. Chi, A. Choudhary, S. Dosanji, T. Dunning, S. Fiore, A. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Z. Jin, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M.S. Mueller, W.E. Nagel, H. Nakashima, M.E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Valero, A. van der Steen, J. Vetter, P. Williams, R. Wisniewski, K. Yelick, The international exascale software project roadmap, *Int. J. High Perform. Comput. Appl.* 25 (1) (2011) 3–60, <http://dx.doi.org/10.1177/1094342010391989>, arXiv: <https://doi.org/10.1177/1094342010391989>.
- [15] H. Esmailzadeh, E. Blem, R.S. Amant, K. Sankaralingam, D. Burger, Power challenges may end the multicore era, *Commun. ACM* 56 (2) (2013) 93–102, <http://dx.doi.org/10.1145/2408776.2408797>, URL <http://doi.acm.org/10.1145/2408776.2408797>.
- [16] M. Frigo, C.E. Leiserson, H. Prokop, S. Ramachandran, Cache-oblivious algorithms, *ACM Trans. Algorithms* 8 (1) (2012) 4:1–4:22, <http://dx.doi.org/10.1145/2071379.2071383>, URL <http://doi.acm.org/10.1145/2071379.2071383>.
- [17] M. Frigo, V. Strumpen, CaChe oblivious stencil computations, in: Proceedings of the 19th Annual International Conference on Supercomputing, in: ICS '05, ACM, New York, NY, USA, 2005, pp. 361–366, <http://dx.doi.org/10.1145/1088149.1088197>, URL <http://doi.acm.org/10.1145/1088149.1088197>.
- [18] H. Gahvari, W. Gropp, An introductory exascale feasibility study for FFTs and multigrid, in: 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS), 2010, pp. 1–9, <http://dx.doi.org/10.1109/IPDPS.2010.5470417>.
- [19] V. Getov, A. Hoisie, P. Bose, New frontiers in energy-efficient computing [guest editors' introduction], *Computer* 49 (10) (2016) 14–18, <http://dx.doi.org/10.1109/MC.2016.315>.
- [20] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, *J. Comput. Phys.* 135 (2) (1997) 280–292, <http://dx.doi.org/10.1006/jcph.1997.5706>, URL <http://www.sciencedirect.com/science/article/pii/S0021999197957065>.
- [21] H. Ibeid, R. Yokota, D. Keyes, A performance model for the communication in fast multipole methods on high-performance computing platforms, *Int. J. High Perform. Comput. Appl.* 30 (4) (2016) 423–437, <http://dx.doi.org/10.1177/1094342016634819>, arXiv: <https://doi.org/10.1177/1094342016634819>.
- [22] H. Ibeid, R. Yokota, J. Pestana, D. Keyes, Fast multipole preconditioners for sparse matrices arising from elliptic equations, *Comput. Vis. Sci.* 18 (6) (2018) 213–229, <http://dx.doi.org/10.1007/s00791-017-0287-5>.
- [23] H. Jia-Wei, H.T. Kung, I/O complexity: The red-blue pebble game, in: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing, in: STOC '81, ACM, New York, NY, USA, 1981, pp. 326–333, <http://dx.doi.org/10.1145/800076.802486>, URL <http://doi.acm.org/10.1145/800076.802486>.
- [24] G. Liu, C. Ji, Scalability of network-failure resilience: analysis using multi-layer probabilistic graphical models, *IEEE/ACM Trans. Netw.* 17 (1) (2009) 319–331, <http://dx.doi.org/10.1109/TNET.2008.925944>, URL doi.ieeecomputersociety.org/10.1109/TNET.2008.925944.
- [25] M. Radulovic, D. Zivanovic, D. Ruiz, B.R. de Supinski, S.A. McKee, P. Radojković, E. Ayguadé, Another trip to the wall: How much will stacked DRAM benefit HPC?, in: Proceedings of the 2015 International Symposium on Memory Systems, in: MEMSYS '15, ACM, New York, NY, USA, 2015, pp. 31–36, <http://dx.doi.org/10.1145/2818950.2818955>, URL <http://doi.acm.org/10.1145/2818950.2818955>.
- [26] TOP500.org, TOP500 Supercomputer Site, 2016. URL www.top500.org.
- [27] L. Ying, G. Biros, D. Zorin, A kernel-independent adaptive fast multipole algorithm in two and three dimensions, *J. Comput. Phys.* 196 (2) (2004) 591–626, <http://dx.doi.org/10.1016/j.jcp.2003.11.021>, URL <http://www.sciencedirect.com/science/article/pii/S0021999103006090>.
- [28] R. Yokota, An FMM based on dual tree traversal for many-core architectures, *J. Algorithms Comput. Technol.* 7 (3) (2013) 301–324, <http://dx.doi.org/10.1260/1748-3018.7.3.301>, arXiv: <https://doi.org/10.1260/1748-3018.7.3.301>.
- [29] R. Yokota, H. Ibeid, D. Keyes, Fast multipole method as a matrix-free hierarchical low-rank approximation, in: T. Sakurai, S.-L. Zhang, T. Imamura, Y. Yamamoto, Y. Kuramashi, T. Hoshi (Eds.), *Eigenvalue Problems: Algorithms, Software and Applications in Petascale Computing*, Springer International Publishing, Cham, 2017, pp. 267–286.
- [30] L. Yu, D. Li, S. Mittal, J.S. Vetter, Quantitatively modeling application resilience with the data vulnerability factor, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, in: SC '14, IEEE Press, Piscataway, NJ, USA, 2014, pp. 695–706, <http://dx.doi.org/10.1109/SC.2014.62>.



Huda Ibeid is a postdoctoral researcher at the University of Illinois at Urbana-Champaign. She received her B.Sc. degree in Computer Engineering from the University of Jordan and her Ph.D. degree in Computer Science from the King Abdullah University of Science and Technology (KAUST) in 2016. Her research interests include fast algorithms for particle-based simulations, fast solvers for large-scale linear systems, the design of parallel numerical algorithms, and performance modeling.



Luke Olson is a professor in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He received his Ph.D. in applied mathematics from the University of Colorado at Boulder in 2003 and his research interests include sparse matrix computations, multigrid methods, finite elements methods, and parallel numerical algorithms.



William Gropp is a director and chief scientist of the National Center for Supercomputing Applications and holds the Thomas M. Siebel Chair in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He received his Ph.D. in Computer Science from Stanford University in 1982. He was on the faculty of the Computer Science Department of Yale University from 1982 to 1990 and from 1990 to 2007, and he was a member of the Mathematics and Computer Science Division at Argonne National Laboratory. His research interests are in parallel computing,

software for scientific computing, and numerical methods for partial differential equations. He is a fellow of ACM, IEEE, and SIAM and a member of the National Academy of Engineering.