On the Communication Complexity of 3D FFTs and its Implications for Exascale

Kenneth Czechowski[†], Casey Battaglino[†], Chris McClanahan[*],
Kartik Iyer[‡], P.-K. Yeung[†,‡], Richard Vuduc[†]
Georgia Institute of Technology, Atlanta, GA
[*] School of Computer Science
[†] School of Computational Science and Engineering
[‡] School of Aerospace Engineering
{kentcz,cbattaglino3,chris.mcclanahan,kartik.iyer,pk.yeung,richie}@gatech.edu

ABSTRACT

This paper revisits the communication complexity of large-scale 3D fast Fourier transforms (FFTs) and asks what impact trends in current architectures will have on FFT performance at exascale. We analyze both memory hierarchy traffic and network communication to derive suitable analytical models, which we calibrate against current software implementations; we then evaluate models to make predictions about potential scaling outcomes at exascale, based on extrapolating current technology trends. Of particular interest is the performance impact of choosing high-density processors, typified today by graphics co-processors (GPUs), as the base processor for an exascale system. Among various observations, a key prediction is that although inter-node all-to-all communication is expected to be the bottleneck of distributed FFTs, *intra*-node communication—expressed precisely in terms of the relative balance among compute capacity, memory bandwidth, and network bandwidth—will play a critical role.

Categories and Subject Descriptors

D.1 [**Concurrent Programming**]: Distributed programming

General Terms

Algorithms, Performance

Keywords

FFT, Exascale, Performance Model

1. INTRODUCTION

As we progress toward exascale computing, new challenges, such as power and energy constraints, limit the scalability of traditional high performance computing systems. This has spurred the development of various alternative architectures. One noteworthy example is the recent interest in systems comprised of dense, massively parallel processors, such as GPUs. Yet despite the attention surrounding exascale, there is no consensus on which architectural strategies will win out, both on the processor level and on the system-wide level. Furthermore, this uncertainty raises important questions about how these design choices impact algorithms and software implementations.

This paper studies the ways in which the design of future exascale systems will affect applications targeted to run on these machines. To analyze potential architecture and algorithm scenarios, we propose the use of detailed, principled performance models, incorporating both algorithmic and architectural characteristics. This takes a unique approach towards performance modeling by integrating algorithmic communication complexity analysis—including traffic both within the memory hierarchy and across the network—with cost models that are grounded in technology trends. Our overall goal is to use such models to better understand the strengths and weaknesses of a particular algorithmic or architectural approach, and to apply such insights to quantitatively predict performance on future architectures.

To demonstrate this approach, we consider the problem of implementing a large-scale 3-dimensional fast Fourier transform (3D FFT) on a hypothetical future exascale system. We derive and calibrate a suitable analytical performance model, then use it to make predictions about potential scaling outcomes at exascale, based on the extrapolation of current technology trends. Of particular interest is the performance impact of building systems comprised of high-density compute units, as typified today by graphics co-processors (GPUs).

Contributions.

We make what we believe to be two contributions to our current understanding of large-scale FFT computations and the implications for exascale.

1. **Modeling** (§ 3): We derive an analytical performance model of a 3D FFT that includes computation and *both* inter- and intra-node communication costs. The inter-node terms account for topology; the intra-node terms include memory bandwidth, cache, and I/O-bus (PCIe) effects. This gives us a basis for studying

how performance changes as machine parameters vary; how alternative 3D FFT algorithms might behave; and what the future may hold. We instantiate and validate this model experimentally on existing systems. An intuitive interpretation of the model in terms of *processor balance* [6, 8, 11, 33, 35] is given in § 4.

2. **Predictions** (§ 6): Using our model, we consider trends in architecture and FFT performance over the past 30 years and predict what might happen in the year 2020, at exascale. Current debates about exascale architectures suggest two competing designs, or "swim lanes." One design is based on an embedded-CPU-like processor and the other based on a GPU-like design. Barring certain memory technology changes as discussed below, our analysis yields what may be a surprise to some, if not many: the relatively high compute-density of a GPU-like node could cause a global system imbalance. Consequently, a one exaflop/s (EF/s) system based on an extrapolated CPU-like design might actually outperform a one EF/s GPU-like design for 3D FFTs, or any similarly communication bandwidth-bound computation.

Limitations.

Our work has several limitations and qualifications.

First and foremost, we are not oblivious to the notion that, "Prediction is very difficult, especially about the future."[1] Indeed, we rely critically on assumptions that are subject to considerable debate and dramatic shifts. We do discuss some specific threats to validity, including, for instance, viability and impact of stacked memory on intra-node FFT performance. Our real intent in making any predictions at all is to foster discussion and perhaps influence the future, rather than to only obtain the strictly "correct" answer.

Second, our FFT analysis is far from exhaustive. For example, we only consider the pencil decomposition of the transpose method for computing a 3D FFT. Other distributed FFT algorithms exist, but for realistic problem sizes on current and future large-scale systems the pencil decomposition is the best option [17, 22]. We also assume ideal problem sizes: n is a power of two and all dimensions are equally sized. Interested readers are directed elsewhere for examples of how irregular problem sizes can impact performance [9]. Our analysis also omits several factors that could play key roles in future systems. Chief among these are microscopic memory and network contention effects, which we account for implicitly through parameter calibration. These omissions imply that our estimates are optimistic, as we show when we try to validate the model. Nevertheless, our modeling goal is to *bound* performance, in order to understand scalability independent of implementation artifacts that might reasonably disappear by the time we reach exascale. In that sense, the model may still permit interesting conclusions.

Finally, we have framed our Prediction section (§ 6) around "CPUs vs. GPUs," mostly to be topical and provocative. Such comparisons for exascale are in our view a red herring. In fact, our analysis abstracts away the details of CPU and GPU structures and characterizes them simply by balance. The predominant trend in nearly all processor designs is toward greater performance at the cost of increased imbalance. Our paper serves as a cautionary tale about the consequences of increasing imbalance too aggressively.

[1] Attribution is disputed though often credited to physicist Niels Bohr.

2. RELATED WORK

There is a flurry of current research activity in performance analysis and modeling, both for exascale in general and in particular for 3D FFT algorithms and software at all scales of parallelism. Our paper most closely follows three recent studies.

The first study is by Pennycook et al., who also consider inter- vs. intra-node communication for the NAS-LU benchmark (parallel wavefront stencil), leveraging their earlier empirical modeling work [37]. Our model is by contrast more explicit about particular intra-node parameters, such as bandwidth, cache size, and I/O bus factors, and so our model adds improved algorithm-architecture understanding relative to this prior work.

The second study is Gahvari's and Gropp's theoretical analysis of feasible latency and bandwidth regimes at exascale, using LogGP modeling and pencil/transpose-based FFTs as one benchmark [9, 22]. Their model is more general than ours in that it is agnostic about specific architectural forms at exascale; however, ours may be more *prescriptive* about the necessary changes by explicitly modeling particular architectural features.

The third study is Kogge's and Dysart's exascale projection paper [30]. Their paper tracks system characteristics of Top500 supercomputers over the past 18 years to establish technology and architecture trends. Like our analysis, they use historical data to predict exascale technologies, but their paper primarily focuses on classifying supercomputers rather than evaluating the impact of these design choices. We take these projections a step further by combining them with our performance models to make quantitative performance predictions.

Beyond these key studies, there is a vast literature on 3D FFTs [2, 3, 5, 10, 12–15, 19, 20, 24, 25, 27, 32, 39, 40, 43]. We call attention to just a few of these. For large problem sizes, the speed record is 34.7 teraflop/s (TF/s) in 1D set by the 88,126 processor "K computer" manufactured by Fujitsu [1]. For relatively small problem sizes, the most impressive strong scaling demonstration is the 32^3 run on the Anton system, which uses custom ASIC network chips and fixed-point arithmetic, completing a 32^3 3D FFT in 4 μs (614 GF/s) [43], which our codes can only attain for relatively much larger problem sizes (see § 5).

3. 3D FFT PERFORMANCE MODEL

In this section we develop a performance model for a distributed 3D FFT on P nodes and total problem size $N = n^3$.The algorithm is illustrated in Figure 1. It consists of three computation phases which are separated by two communication phases. Each computation phase computes n^2 1D FFTs of size n in parallel. Each communication phase involves $\sqrt{P}$ independent P-node personalized all-to-all exchanges. We describe our model in terms of the time to compute (§ 3.1) and time to communicate (§ 3.2), but focus on dominant terms T_{mem} and T_{net}. In § 5 we identify artifacts of modern architectures, such as the CPU$\leftrightarrows$GPU memory transfer (§ 5.3.1) and local transpose costs (§ 5.2.2), that do not appear in the algorithmic analysis yet can have an impact on performance.

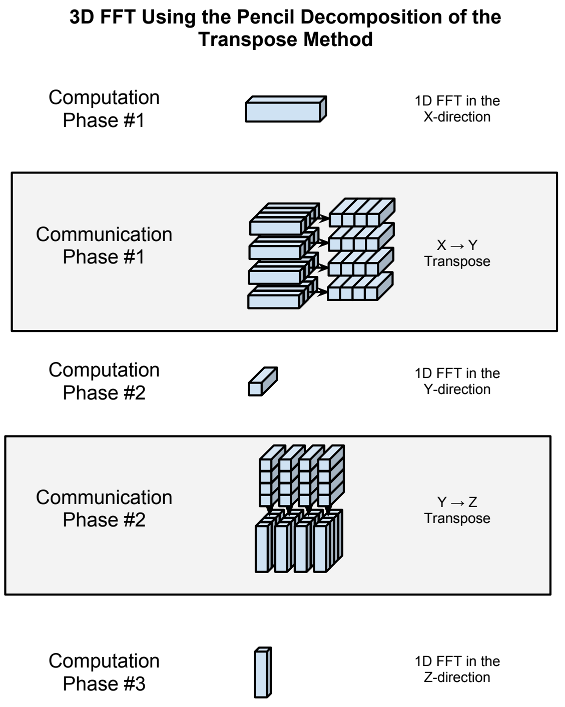
3D FFT Using the Pencil Decomposition of the Transpose Method
Computation Phase #1
1D FFT in the X-direction
Communication Phase #1
X → Y Transpose
Computation Phase #2
1D FFT in the Y-direction
Communication Phase #2
Y → Z Transpose
Computation Phase #3
1D FFT in the Z-direction

Figure 1: An illustration of the Computation and Communication phases in a 3D FFT using a pencil decomposition of the transpose method.

We make an additional simplifying assumption: we assume just one processor per node. The reason is that the cost of communicating between processors within a node versus the cost between nodes may actually be decreasing over time, as suggested in the trend data that appears in § 6.

Note about units of measure.

When counting flops, we assume *scalar flops*. That is, for the multiplication of two complex numbers, we would count 6 flops when using the classical method (4 scalar multiplies and 2 scalar adds). When counting volumes of data, however, we assume each *word* is a *complex* value. We use double-precision flops and double-complex words (16 bytes per word) unless otherwise specified.

3.1 Computation Costs

The 3D FFT is decomposed into $3n^2$ 1D FFTs, each of length n, distributed evenly among the P nodes. To approximate the cost of a local 1D FFT, we must consider both the cost of the floating point operations (T_{flops}) as well as the memory operations (T_{mem}).

3.1.1 Flop Costs

The 1D FFT of size n is computed using $\Theta(n \log n)$ floating point operations. The "radix-2" algorithm, which was originally presented by Cooley and Tukey in 1965, consists of approximately $5n \log_2 n$ flops. Since then numerous FFT algorithms have been presented, most of which reduce the absolute flop count slightly ($\approx 20\%$), but the practical performance benefits come from algorithms that allow the computation to be structured in such a way that fully exploits the caches and SIMD lanes of the processor rather than the improved work load. Therefore, for simplicity we will rely on the conventional constant, which closely matches the observed flop count in § 5.2.1.

Using this approximation of the 1D FFT, we can approximate the total cost of all 1D FFTs during the three computation phases ($3 \times \frac{n^2}{P}$ 1D FFTs per node). If each node can perform C_{node} flops per unit time, then the total time spent on computation is

$$T_{\text{flops}} = 3 \times \frac{n^2}{P} \times \frac{5n \log n}{C_{\text{node}}} \ . \qquad (1)$$

3.1.2 Memory Operation Costs

Within each node, we must load and store each data point from memory at least once during a 1D FFT computation. Caches are exploited to prevent an additional DRAM access for each operand of each floating point operation. For a local 1D n-point cache-oblivious FFT in a two-level memory hierarchy, with cache size Z and line size L in words, the number of cache misses grows on the order of $\Theta(1 + (n/L)(1 + \log_Z n))$ [21]. This result is I/O-optimal [29], and so represents the best case asymptotic performance for any algorithm and implementation. Note that this bound counts cache misses, each transferring lines of size L, hence the additional factor of L in the equation.

Thus, for some constant A and sufficiently large n, the time spent moving data between main memory and the processor is

$$T_{\text{mem}} \quad \approx \quad 3 \times \frac{n^2}{P} \cdot \frac{A \times n(\max(\log_Z n, 1.0))}{\beta_{\text{mem}}} \ , \qquad (2)$$

where β_{mem} is the node's memory bandwidth in words per unit time. The *max* function ensures that the transfers include at least the compulsory misses.

Unfortunately, the relative complexity of the cache hierarchy on modern processors (e.g., multi-levels, replacement policies, address translation, etc.) makes an analytical derivation of the constant prohibitively difficult. Instead, in § 5.2.1 we will approximate the constant $A = 6.3$ by using hardware counters to track the number of DRAM accesses during a 1D FFT computation.

3.1.3 Flops:Byte

Assuming arithmetic and memory operations can be overlapped, $T_{comp} \approx \max(T_{\text{flops}}, T_{\text{mem}})$. However, the computational intensity (flops:byte) of a 1D FFT is generally lower than the machine balance of modern processors, making the computation memory bound. Furthermore, our analysis in § 6 suggests processors will be even more imbalanced in the future. This leads us to focus on T_{mem} instead of T_{flops}.

3.2 Communication Costs

During the communication phase, each node must perform a personalized all-to-all exchange of its data points with $\sqrt{P}$

other nodes on the network. In total, each node sends approximately $\frac{n^3}{P}$ data points.[2]

Ideally, on a fully connected network with link bandwidth β_{link}, the time to perform this exchange is

$$T_{\text{net}} \quad \approx \quad 2 \times \frac{n^3}{P \cdot \beta_{\text{link}}} \;, \tag{3}$$

where the factor of 2 accounts for the two communication phases.

Since a fully connected network is unlikely at exascale, we assume a more realistic 3D torus topology for the analytical sections of this paper. We consider two scenarios. First, a lower-bound on the communication phase on a 3D torus is

$$T_{\text{net}} = \Omega\left(\frac{n^3}{P^{\frac{5}{6}} \cdot \beta_{\text{link}}}\right) \;. \tag{4}$$

This models the communication phase as simultaneous $\sqrt{P}$-node all-to-all exchanges within $P^{\frac{1}{6}} \times P^{\frac{1}{6}} \times P^{\frac{1}{6}}$ subblocks of the 3D torus (see § 9 for a detailed derivation). For the problem size and machine parameter values we consider in this paper, the latency term will be negligible, and so does not appear in subsequent uses of this formula.

However, the bound of Equation (4) makes strong assumptions about task placement that are not always feasible on shared high-end systems [28]. Thus, we will also consider a second, more realistic approximation by using the cost of a global all-to-all. The communication time is then bound by the bisection bandwidth of a 3D torus, which is $\mathcal{O}\left(P^{\frac{2}{3}} \cdot \beta_{\text{link}}\right)$; thus,

$$T_{\text{net}} \quad \approx \quad 2 \times \frac{n^3}{P^{\frac{2}{3}} \beta_{\text{link}}} \;. \tag{5}$$

4. INTERPRETING THE MODEL: BALANCED PROCESSORS

This section gives a more intuitive explanation of the high-level features of the model described in § 3.

Suppose we wish to build a machine with a particular peak performance of R_{peak} flops per unit time. Given a node of peak C_{node}, we will choose the number of nodes $P = R_{\text{peak}}/C_{\text{node}}$. Thus, T_{mem} from Equation (2) becomes, in the limit of large n,

$$T_{\text{mem}} \approx \mathcal{O}\left(\frac{1}{R_{\text{peak}}} \cdot \frac{C_{\text{node}}}{\beta_{\text{mem}}} \cdot n^3 \log_Z n\right) \;. \tag{6}$$

Note the factor $C_{\text{node}}/\beta_{\text{mem}}$. This factor is the classical definition of *balance* [6, 8, 11, 33, 35], which has units of flops / word (or byte), applied here to node performance. When this factor is large (in this case relative to the inherent flop:byte requirements of an FFT), we say the node is *imbalanced*. Thus, the intra-node communication time depends not on the absolute performance of a node but instead on this balance ratio.

A similar argument applies to the inter-node time, T_{net}, yielding from Equation (5),

$$T_{\text{net}} \approx \mathcal{O}\left(\frac{1}{R_{\text{peak}}^{\kappa}} \cdot \frac{C_{\text{node}}^{\kappa}}{\beta_{\text{link}}} \cdot n^3\right) \;, \tag{7}$$

where $\kappa = \frac{5}{6}$, using the optimal task-mapping lower-bound of Equation (4), or $\kappa = \frac{2}{3}$, using the bisection bound of Equation (5). Again, a similar kind of balance factor appears, here relative to network bandwidth. Evidently, inter-node time also depends on balance. However, since $\kappa < 1$, T_{net} is less sensitive to C_{node} than is T_{mem}.

Thus, we may conclude the following: balanced nodes reduce both T_{mem} and T_{net}; and T_{mem} depends more sensitively on C_{node} than T_{net}. Put another way, intra-node design is critical, but in perhaps an unintuitive way: a supercomputer composed of many weak but balanced nodes could perform better than one with the same peak and fewer more powerful but imbalanced nodes.

5. MODEL CALIBRATION

Empirical tests are important for calibrating the analytical model as well as identifying additional hardware and software artifacts that impede performance. In the case of the FFT, we use this opportunity to

- use timing data to estimate effective throughput of both memory bandwidth and network bandwidth;
- use hardware counters to measure the analytical constants in Equation 1 and Equation 2;
- explore hardware and software artifacts that influence performance;
- compare and contrast performance on systems with different underlying architectures —a CPU-based system and a GPU-based system.

5.1 Experiment Setup

The experiments were run on 4,096 nodes (98,304 cores) of "Hopper," a 1.288 PF/s Cray XE6 housed at the the National Energy Research Scientific Computing Center. Each node has two Magny-Cours chips[3] running at 2.1 GHz and 32 GB DDR3 1333-MHz memory. Hopper's 6,392 nodes form a 17×8×24 3D torus with a Gemini interconnect.

In addition to Hopper, the GPU experiments were run on "Keeneland," which consists of HP SL390 servers accelerated with NVIDIA Tesla M2070 GPUs. Keeneland has 120 compute nodes, each with dual-socket, six-core Intel X5660 2.8 GHz Westmere processors and 3 GPUs per node, with 24GB of DDR3 host memory. The interconnect is single rail, QDR Infiniband [41].

We use the P3DFFT (Parallel Three-Dimensional Fast Fourier Transform), an off-the-shelf distributed 3D FFT library, to perform the computations. It implements the distributed memory transpose algorithm using a pencil decomposition [36]. P3DFFT is freely available under a GPL license.[4] A major use of P3DFFT, which could be considered to be our motivating application, is a direct numerical turbulence simulation.

On each node P3DFFT computes local 1D FFTs using third party libraries. We used FFTW 3.2.2, though IBM's

[2]Each node need only send $\frac{(\sqrt{P}-1)\cdot n^3}{P^{\frac{3}{2}}}$ data points, which is approximately $\frac{n^3}{P}$ (all of its data) for the relatively large values of P considered in this paper.

[3]the Magny-Cours chip contains two six-core Istanbul processors on a single die

[4]`http://code.google.com/p/p3dfft/`

ESSL and Intel's MKL libraries can serve as drop-in replacements. For the GPU experiments we developed DiGPUFFT, [5] a custom wrapper around CUFFT, Nvidia's FFT library for GPUs, making CUFFT an additional option.

5.2 Hopper Performance Results

Here we present our empirical performance results from a large-scale FFT, run on 4096 nodes of Hopper. Figure 2 shows the time spent during the Communication and Computation phases over varying problem sizes. To quantify the additional overhead of the 3D FFT, we timed components in an isolated setting. This involved benchmarking the "MPI_AlltoAll()" and FFTW calls individually. The results are shown in Figure 2. Observe that

- network communication time dominates computation time. However, as the problem size increases the computation time grows faster than communication time. This can be explained by the asymptotic complexities of Equation (2) and Equation (5).
- the computation phase incurs a large overhead that is not explained by the analytical model;
- total time is the sum of the computation and communication phases. There is no overlap.

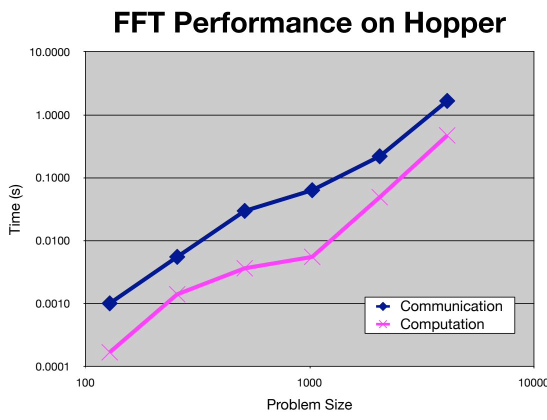

Figure 2: P3DFFT performance on 4096 nodes of Hopper. The plot shows the time spent during the Computation and Communication phases of the 3D FFT.

5.2.1 *Measuring Computational Constants*

The computational complexity of the FFT and its constant are described in Section 3.1.2. To measure the constant for the FFT's cache complexity (to allow us to model its bandwidth load), we executed FFTW on all arrays of power-of-two size that were too large to fit in L3 cache but small enough to fit in DRAM. We used timing and profiling results and divided by the cache complexity equation to determine the constant A in Equation (2), assuming that the memory bottleneck exists between DRAM and L3 cache.

5.2.2 *Local Transpose Artifact*

[5]DiGPUFFT is freely available as a set of patches to P3DFFT at `http://code.google.com/p/digpufft/`

Prior to the personalized all-to-all exchange, each node must compute a *local transpose* to reshuffle the data in addition to performing the local FFT computation. This shuffle can be costly because it involves loading and storing nearly every value in the local data set. For a 3D FFT, where each of the P processors owns $\frac{n^3}{P}$ complex words, the total time spent reshuffling by each processor is

$$T_{\text{shuffle}} \approx 2 \times \frac{2n^3}{P \cdot \beta_{\text{mem}}}, \tag{8}$$

where β_{mem} is the local main memory bandwidth in words per unit time. There are two factors of 2 here: one accounts for loads and stores, and the other for the fact that in a 3D FFT there will be two all-to-all exchanges. Figure 4 shows that in DiGPUFFT the local transpose (16%) is nearly as costly as the FFT computation (21%) itself. Other FFT implementations have shown similar relative costs [19, 43].

5.3 Keeneland Performance Results

Here we present our 3D FFT performance results for a GPU cluster. Rather than comparing this data with the results from Hopper, which is much larger and has a more sophisticated interconnect than Keeneland, we also include P3DFFT results from experiments that were run on the CPU processors in Keeneland. The results are shown in Figure 3.

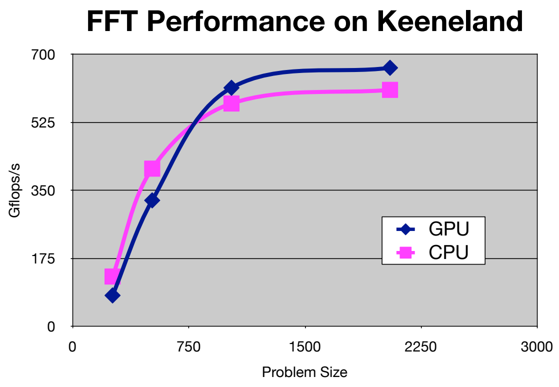

Figure 3: DiGPUFFT performance on Keeneland.

5.3.1 *The I/O Bus (PCIe) Bottleneck Artifact*

An important observation from Figure 3 is that despite the GPU's higher memory bandwidth and floating-point throughput, we observe only a modest overall win from GPU over CPU, showing just roughly 10% to 20% improvements. Further inspection (see Figure 4) reveals that 27% of the time is spent in CPU$\leftrightarrows$GPU communication.

In the case of a 3D FFT of size n^3 evenly distributed across P nodes, this can result in a transfer penalty of

$$T_{\text{PCIe}} \approx \frac{2n^3}{P \cdot \beta_{\text{PCIe}}} \tag{9}$$

during each computation phase, where β_{PCIe} is the I/O bus bandwidth in words per unit time. The factor of two accounts for both CPU$\rightarrow$GPU and CPU$\leftarrow$GPU transfers. Since the PCIe bandwidth (β_{PCIe}=8 GB/s) is an order of

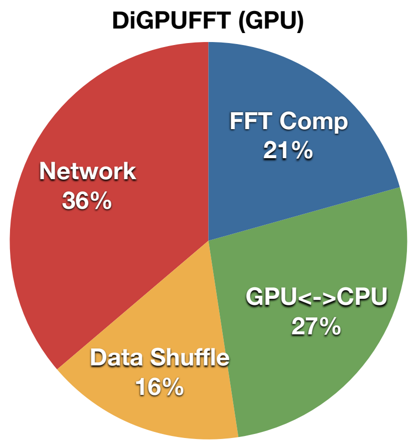

Figure 4: Breakdown of execution time in DiGPUFFT (GPU-based 3D FFT). Results are from a 64 node run on the Keeneland cluster with a problem size of 2048^3, and 3 MPI tasks per node.

magnitude slower than the memory bandwidth on the GPU (144 GB/s or more), the PCIe bus can have a significant impact on performance.

Fortunately, the PCIe bottleneck is likely to improve in the near future. The gap in bandwidth between β_{PCIe} and main memory bandwidth is, as it happens, decreasing. Additionally, several approaches are underway to circumvent the CPU⇆GPU memory transfer cost altogether. Early development in MVAPICH-GPU, a MPI implementation that improves the connection between the GPU and network interface, has already shown a 45% performance improvement over the indirect cudaMemcpy() + MPI_Send [42]. Or in a more extreme case, AMD's Fusion architecture discards the notion of a discrete GPU by incorporating CPU and GPU cores on the same processor die. These observations reaffirm generally held views that the effect of PCIe is a short-term artifact, not a long-term barrier to scalability.

6. PROJECTING FORWARD

Recent discussions surrounding the direction of high-end systems has separated into two strategies, sometimes called the exascale processor "swim lanes." Others use the terms *Many-Core* (MC) and *Many-Thread* (MT) designs [16, 23, 26, 30]. MC designs are "CPU-like," in that they replicate embedded CPU-cores, and emphasize latency reduction through traditional memory hierarchy and other techniques that boost instruction-level parallelism. By contrast, MT designs replicate GPU-style processors, with an emphasis on hiding latency via massive multithreading. In this paper, the essential difference is that MC designs have lower absolute performance but may be better balanced than MT designs. Recall that the key design consideration for 3D FFTs, as suggested in § 4, is not absolute per-processor performance, but rather the relative balance among compute capacity per node, intra-node bandwidth, and inter-node bandwidth.

Predictions.

We use our model to see how performance and scaling could change in light of current technology trends. We summarize these trends with respect to various machine parameters in Table 1, which extrapolates from the current configuration of the Keeneland cluster using our own derived trends.[6] We present these extrapolations in terms of the time (in years) for a particular parameter to double and the factor by which current values might increase in ten years. For several parameters, we separate extrapolated CPU-like vs. GPU-like processors. However, since the two are fundamentally based on similar process technologies (e.g., silicon and manufacturing processes), we hypothesize that the *rates* of growth will be identical through 2020 though they start from different values in 2010.

Table 1: Processor architecture projections, from starting values on the US National Science Foundation's "Keeneland" (GPU-based) and NERSC's Hopper system (CPU-based), both delivered in 2010. Processor/Node counts are scaled to reflect a 4 PF/s machine

Parameter		2010 values	Doubling time (in years)	10-year increase factor	value
Peak:	C_{CPU}	50.4 GF/s	1.7	59.0×	3.0 TF/s
	C_{GPU}	515 GF/s			30 TF/s
Cores:[a]	ρ_{CPU}	6	1.87	40.7×	134
	ρ_{GPU}	448			18k
Memory bandwidth:	β_{CPU}	21.3 GB/s	3.0	9.7×	206 GB/s
	β_{GPU}	144 GB/s			1.4 TB/s
Fast memory	Z_{CPU}	6 MB	2.0	32.0×	192 MB
	Z_{GPU}	2.7 MB[b]			86.4 MB
Line size:	L_{CPU}	64 B	10.2	2.0×	128 B
	L_{GPU}	128 B			256 B
Link bandwidth:	β_{link}	10 GB/s	2.25	21.8×	218 GB/s
Machine peak:	R_{peak}	4 PF/s	1.0	1000×	4 EF/s
System memory:	E	635 TB	1.3	208×	132 PB
Nodes	P_{CPU}	79,400	2.4	17.4×	1.3M
$(\frac{R_{\text{peak}}}{C})$:	P_{GPU}	7,770			135,000

[a] "Cores" refers to the processor manufacturer's own usage rather than, say, floating-point functional units.
[b] Fast Memory refers to the capacity of on-chip cache. In practice, we usually only measure the last-level cache because it dominates the total cache size. However, on the M2070 GPU, the sum of the L1 + registerfiles (2.7 MB) is larger than the L2 (512 KB).

Prediction 1: Under business-as-usual assumptions (Table 1), a 3D FFT will achieve 2.8 petaflop/s (PF/s) on a high-density GPU-like exascale machine in 2020. This value is 0.08% of the 4 EF/s peak, compared to today's best fraction of peak for 3D FFTs, which is about 0.5%.

To obtain this estimate, we extrapolated the various system parameters of Table 1, used them to determine the form (e.g., number of nodes) required to get a system running at 4 EF/s, selected a problem size according to the methodology of Gahvari and Gropp [22], and then evaluated our

[6]Our precise methodologies for deriving these trends appear in the extended appendix. See § 9.

performance model to estimate execution time. Prediction 2 below further elaborates on this calculation.

Prediction 2: On 4 EF/s systems that are expected in 2020, we predict that the MC or "CPU-like" swim lane will deliver higher performance for a 3D FFT. Our quantitative prediction is that, given MC-style and MT-style systems of 4 EF/s peak each, the MC-style system will deliver about 3.2× better performance.

Figure 5 summarizes our quantitative prediction and shows why we think an MC-style design wins. We consider three scenarios: (i) a GPU-like system vs. a CPU-like system matched on peak performance (GPU vs. CPU-1); (ii) a GPU-like system vs. a CPU-like system which both perform the FFT in the same time, assuming inter- and intra-node communication time *cannot* be overlapped (GPU vs. CPU-2); and (iii) a GPU-like system vs. a CPU-like system which both perform the FFT in the same time, assuming inter- and intra-node communication time *can* be overlapped (GPU vs. CPU-3). We fix the problem size to be an n^3 volume with $n = 21,000$. Figure 5 further breaks down the execution time into inter- vs. intra-node time. For the inter-node time, T_{net}, we assume Equation (5) for T_{net}; we revisit this choice in Prediction 3 below. Note that flop-time does not appear; it is practically negligible, as communication time dominates it by roughly three orders of magnitude.

First, consider the case in which the GPU-like and CPU-like systems have the same peak. For the GPU-like design, the values of T_{mem} and T_{net} appear in the leftmost bar of Figure 5. Network time dominates memory time by 2.8×. The CPU-like system, labeled CPU-1, appears as the second bar of Figure 5. This system spends less time than the GPU system in both forms of communication, with its network communication time nearly $\frac{1}{5}$ the time. From the discussion of § 4, the essential reason is better processor balance, which translates into lower memory and network time. However, this advantage is not free: CPU-1 requires over 10× as many processors as the GPU system. The price of such a system could be prohibitive, if dollar cost is proportional to the number of processors.

A more sensible CPU-like configuration might be one in which we match not on *peak* performance but rather on *actual* performance, that is, the actual time to perform the FFT. In the third bar of Figure 5, we consider a different CPU-based design, CPU-2, in which the total communication time for the FFT, $T_{\text{mem}} + T_{\text{net}}$, exactly matches that of the 4 EF/s GPU system. This change requires fewer processors than CPU-1 (350k vs. 1M) and has a lower overall peak of 1 EF/s, making it perhaps significantly cheaper. More interestingly, observe that relative to the GPU system, CPU-2 trades higher on-chip memory communication time for lower off-chip network communication.

The CPU-2 system may be pessimistic in its execution time, because it assumes that we cannot overlap T_{mem} and T_{net}. The rightmost bar of Figure 5 considers CPU-3, where we assume that the total time to perform the FFT is not the sum, $T_{\text{mem}} + T_{\text{net}}$, but rather $\max\{T_{\text{mem}}, T_{\text{net}}\} = 0.528$ seconds when the two communication phases perfectly overlap. The CPU-3 system loses in total time but reduces off-chip network communication time. This situation can be beneficial if off-chip communication consumes much more energy than on-chip communication [31]. Also, this system uses 295k CPUs, putting it within 2.25× of the total number of processors in the GPU system.

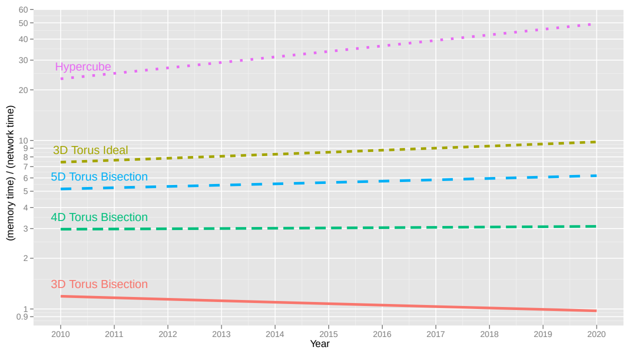

Figure 6: Projected ratio of $T_{\text{mem}}/T_{\text{net}}$ for weakly scaled 3D FFTs. The problem size n^3 starts at 4096^3 and is scaled as the same rate as P.

Prediction 3: Time spent moving data within the node could dominate time spent communicating in the network. The implication is that the all-to-all exchange will *not* be the factor that limits FFT scalability; rather, intra-node design will.

To get some intuition for the conditions under which this prediction could come true, consider the ratio $T_{\text{mem}}/T_{\text{net}}$, which is greater than one if main memory communication time dominates network time. Using the simplified analysis of § 4, we have

$$\frac{T_{\text{mem}}}{T_{\text{net}}} \propto \left(\frac{C_{\text{node}}}{R_{\text{peak}}}\right)^{1-\kappa} \cdot \frac{\beta_{\text{link}}}{\beta_{\text{mem}}} \cdot \log_Z n \,. \qquad (10)$$

The log factor grows slowly, so we can ignore it for the moment. Then, according to Table 1, $C_{\text{node}}/R_{\text{peak}}$ decreases over time, while $\beta_{\text{link}}/\beta_{\text{mem}}$ increases over time.[7] If we wish to control whether the overall product of these two factors increases or decreases, our main "tuning knob" is the network topology, which is captured by κ (recall that $\kappa \leq 1$).

We show how this ratio might grow over time for various topologies in Figure 6, where the base processor is our extrapolated CPU system (as it varies over time). Using the bisection bandwidth estimate given by Equation (5), we see that the overall ratio of $T_{\text{mem}}/T_{\text{net}}$ is less than one for a 3D torus and, furthermore, actually decreases over time, which means the network becomes more and more of a bottleneck. However, under an optimal mapping ("3D Torus Ideal" in Figure 6), a 3D torus could also be as much as 10× faster than suggested by the bisection estimate (compare Equation 4 with Equation 5), and the ratio actually increases over time. Thus, the extent to which the network limits scalability depends on the topology. Figure 6 includes bisection estimates for higher dimensional torii as well, in which cases $T_{\text{mem}}/T_{\text{net}}$ also increase. The slopes of these lines suggest that a 4D torus is well-balanced for an FFT, and that higher-dimensional networks are likely to be over-engineered. However, if there is a severe energy penalty for

[7]For us, the fact that β_{link} and β_{mem} are converging is interesting. One explanation for why this happens is that the physical footprint of pins going into a processor or DIMM cannot grow as quickly as the width of, say, wires going into a router or link. This notion is consistent with recent suggestions by interconnect architects, who try to keep the growth in network bandwidth as close to compute capacity increases as costs will permit [7].

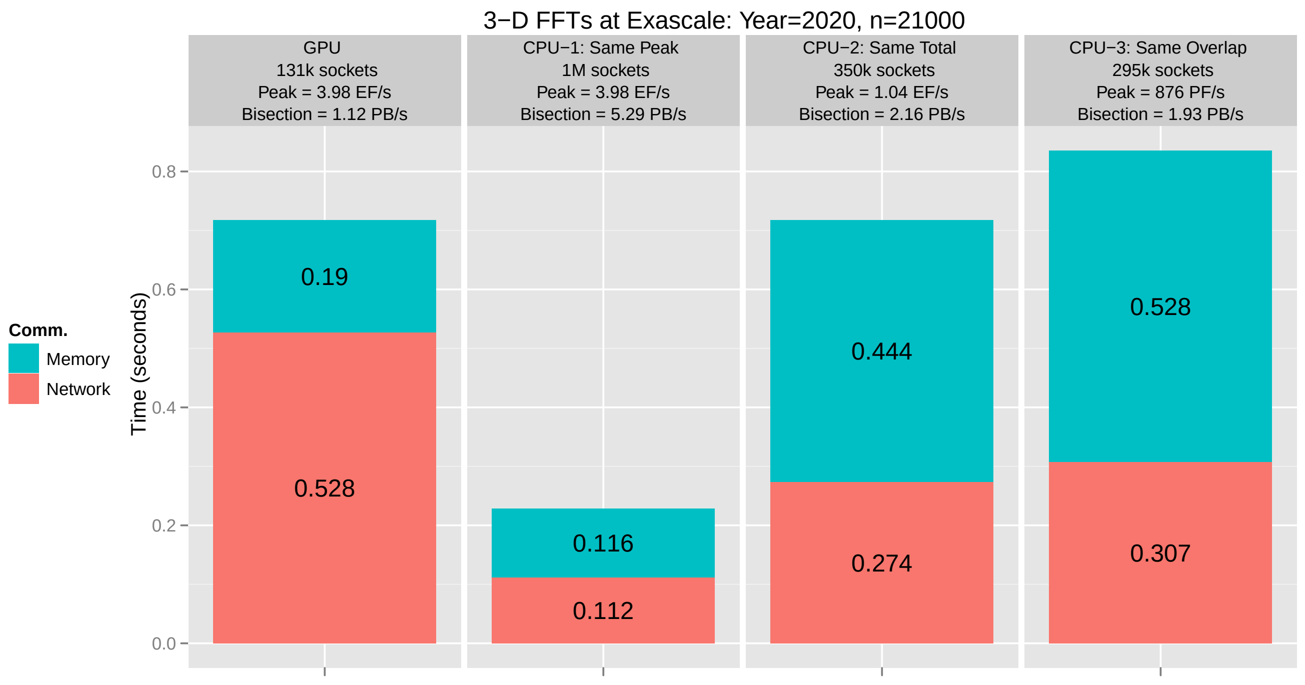

Figure 5: We consider three extrapolated CPU-like systems vs. an extrapolated GPU-like system. CPU-1 has the same peak as GPU; CPU-2 computes the FFT in the same total time as GPU, assuming no overlap of communication; and CPU-3 also yields the same time as GPU, but assuming full overlap. In all cases, the CPU systems actually perform *less* network communication.

network communication, then 5D and higher-dimensional FFTs will help provide energy scaling over time, since the $T_{\text{mem}}/T_{\text{net}}$ is tending to increase in those cases.

Note on xPU-memory stacking.

One technology that could disrupt these analyses is xPU-memory die stacking, the leading proposed mechanism for enabling memory bandwidth to scale at the same rate as compute capacity [34]. Applying our model, which is based on the known I/O complexity estimates for the FFT, we can establish that a node's computation and memory transfers will be balanced when $T_{\text{mem}} \le T_{\text{flops}}$ [11, 33]. Given a chip with ρ cores, C flop/s per core, β byte/s bandwidth to the chip, and a shared cache of size Z, this balance constraint yields [11]

$$\frac{\rho \cdot C}{\beta} \le \mathcal{O}\left(\log \frac{Z}{\rho}\right) . \tag{11}$$

In theory, stacked memory makes it possible to keep the left-hand side constant over time. Although ρ grows faster than Z, it enters into this inequality through the log and so will not decrease too quickly. Thus, stacked memory would keep the processing system balanced for FFTs. However, if it is not possible to keep $\rho = \Theta(\beta)$, then stacked memory only delays rather than solves the processor imbalance problem.

7. CONCLUSIONS AND FUTURE WORK

One claim of this paper is that I/O-bus (PCIe) and network bandwidth will not be the true limiters of performance for parallel 3D FFTs. Instead, it is intra-node communication due to main memory bandwidth that will have the biggest impact at exascale. In fact, our key prediction is that if we ignore intra-node balance, as a naïvely extrapolated GPU-style design would do, then we will hurt overall system balance by not taking advantage of the better scaling of network bandwidth relative to memory bandwidth. Thus, more architectural emphasis on intra-node balance will have the biggest pay-off in the long-run for communication bandwidth-bound computations on high-end systems.

In terms of absolute bandwidth values, high-density (GPU-based) compute nodes extend the time until which network bandwidth will outpace main memory bandwidth, but do not fundamentally solve the problem. The most interesting solution for FFT-like computations, which include sorting, is most likely stacked memory if it can indeed deliver proportional scaling of memory bandwidth to core counts.

Interestingly, the most common weak-but-balanced processor designs are those of mobile processors. Our analysis hints strongly that leveraging the volume of production of such processors, combined with high-quality integrated networking, die stacking, and a balanced-throughput design, could be an excellent building block for an exascale system. This notion may, in fact, be part of an emerging view in other large-scale systems, such as data centers [4, 38].

Algorithmically, perhaps the only way forward for FFT type computations is through much more aggressive data or numerical (e.g., low-rank) compression [18].

We believe our basic modeling methodology and its level of detail could provide similar kinds of insights for other computations, particularly if enriched with additional parameters and an explicit accounting of power, energy, and even dollar costs. This style of analysis could be especially useful in the context of algorithm-architecture co-design, a notion that has been outlined for intra-node designs elsewhere [11].

Finally, we would like to emphasize the methodological contribution of this paper over the analytical analysis. While our predictions are provoking and offer a fresh perspective, the results are suggestive rather than conclusive. The intention of this paper is not to fuel the CPU vs GPU debate or dictate microarchitecture design. Instead we are trying to offer a quantitative approach for reasoning about the relationship between hardware and software in future supercomputers. In time, as the path toward exascale computing becomes clear, a more rigorous analysis will be possible.

8. ACKNOWLEDGEMENTS

We thank the reviewers for their suggestions. This work was supported in part by the National Science Foundation (NSF) under award number 0833136, NSF CAREER award number 0953100, NSF I/UCRC award number 0934114, NSF TeraGrid allocations CCR-090024 and ASC-100019, joint NSF 0903447 and Semiconductor Research Corporation (SRC) Award 1981, and grants from the U.S. Dept. of Energy (DOE) and the Defense Advanced Research Projects Agency (DARPA). This research also used resources of the Keeneland Computing Facility at the Georgia Institute of Technology, which is supported by the National Science Foundation under Contract OCI-0910735. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of NSF, SRC, DOE, or DARPA.

9. APPENDIX

We have released a tech report version of this paper which includes an extended appendix with: (1) a more detailed discussion of the the performance model, (2) raw data used to generate the technology tends in Table 1, and (3) code that can be used to reproduce the calculations found in Section 6.

References

[1] The HPC Challenge benchmark. `http://icl.cs.utk.edu/hpcc`.

[2] R. Agarwal, F. Gustavson, and M. Zubair. An efficient parallel algorithm for the 3-D FFT NAS parallel benchmark. In *Proceedings of IEEE Scalable High Performance Computing Conference*, pages 129–133. IEEE Comput. Soc. Press, 1994.

[3] G. Almasi et al. Cellular supercomputing with system-on-a-chip. In *2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.02CH37315)*, pages 196–197. Ieee, 2002.

[4] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A fast array of wimpy nodes. *Communications of the ACM*, 54(7):101–109, July 2011.

[5] C. Bell, D. Bonachea, R. Nishtala, and K. Yelick. Optimizing Bandwidth Limited Problems Using One-Sided Communication and Overlap. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pages 1–10. IEEE, 2006.

[6] G. E. Blelloch, B. M. Maggs, and G. L. Miller. The hidden cost of low bandwidth communication. In U. Vishkin, editor, *Developing a Computer Science Agenda for High-Performance Computing*, pages 22–25. ACM, New York, NY, USA, 1994.

[7] R. Brightwell, K. T. Pedretti, K. D. Underwood, and T. Hudson. Seastar interconnect: Balanced bandwidth for scalable performance. *IEEE Micro*, 26:41–57, May 2006.

[8] D. Callahan, J. Cocke, and K. Kennedy. Estimating interlock and improving balance for pipelined architectures. *Journal of Parallel and Distributed Computing*, 5(4):334–358, Aug. 1988.

[9] A. Chan, P. Balaji, W. Gropp, and R. Thakur. Communication analysis of parallel 3d fft for flat cartesian meshes on large blue gene systems. In *Proceedings of the 15th international conference on High performance computing*, pages 350–364. Springer-Verlag, 2008.

[10] C. E. Cramer and J. Board. The development and integration of a distributed 3D FFT for a cluster of workstations. In *Proceedings of the 4th Annual Linux Showcase & Conference*, Atlanta, GA, USA, 2000.

[11] K. Czechowski, C. Battaglino, C. Mcclanahan, A. Chandramowlishwaran, and R. Vuduc. Balance principles for algorithm-architecture co-design. In *USENIX Wkshp. Hot Topics in Parallelism (HotPar)*, pages 1–5, Berkeley, CA, USA, 2011. Usenix Association.

[12] K. Datta, D. Bonachea, and K. Yelick. Titanium Performance and Potential: An NPB Experimental Study. In *Proceedings of the Languages and Compilers for Parallel Computing (LCPC) Workshop*, volume LNCS 4339, pages 200–214, 2006.

[13] H. Q. Ding, R. D. Ferraro, and D. B. Gennery. A Portable 3D FFT Package for Distributed-Memory Parallel Architectures. In *Proceedings of 7th SIAM Conference on Parallel Processing*, pages 70—-71. SIAM Press, 1995.

[14] P. Dmitruk, L.-P. Wang, W. H. Mattaeus, R. Zhang, and D. Seckel. Scalable parallel FFT for spectral simulations on a Beowulf cluster. *Parallel Computing*, 27(14):1921–1936, Dec. 2001.

[15] J. Doi and Y. Negishi. Overlapping Methods of All-to-All Communication and FFT Algorithms for Torus-Connected Massively Parallel Supercomputers. In *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, number November, pages 1–9. IEEE, Nov. 2010.

[16] J. Dongarra et al. The international exascale software project roadmap. *IJHPCA*, 25(1):3–60, 2011.

[17] D. Donzis, P. Yeung, and D. Pekurovsky. Turbulence simulations on $o(10^4)$ processors. 2008.

[18] A. Edelman, P. McCorquodale, and S. Toledo. The Future Fast Fourier Transform? *SIAM Journal on Scientific Computing*, 20(3):1094, 1998.

[19] M. Eleftheriou, B. Fitch, A. Rayshubskiy, T. Ward, and R. Germain. Scalable framework for 3D FFTs on the Blue Gene/L supercomputer: implementation and early performance measurements. *IBM Journal of Research and Development*, 49(2.3):457–464, 2005.

[20] B. FANG, Y. DENG, and G. MARTYNA. Performance of the 3D FFT on the 6D network torus QCDOC parallel supercomputer. *Computer Physics Communications*, 176(8):531–538, Apr. 2007.

[21] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 285–, Washington, DC, USA, 1999. IEEE Computer Society.

[22] H. Gahvari and W. Gropp. An introductory exascale feasibility study for FFTs and multigrid. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–9, Atlanta, GA, USA, Apr. 2010. IEEE.

[23] M. Garland and D. B. Kirk. Understanding throughput-oriented architectures. *Communications of the ACM*, 53(11):58, Nov. 2010.

[24] L. Giraud, R. Guivarch, and J. Stein. Parallel Distributed FFT-Based Solvers for 3-D Poisson Problems in Meso-Scale Atmospheric Simulations. *International Journal of High Performance Computing Applications*, 15(1):36–46, Feb. 2001.

[25] L. Gu, X. Li, and J. Siegel. An empirically tuned 2D and 3D FFT library on CUDA GPU. In *Proceedings of the 24th ACM International Conference on Supercomputing - ICS '10*, page 305, Tsukuba, Japan, 2010. ACM Press.

[26] Z. Guz, E. Bolotin, I. Keidar, A. Kolodny, A. Mendelson, and U. C. Weiser. Many-core vs. many-thread machines: Stay away from the valley. *IEEE Computer Architecture Letters*, 8:25–28, 2009.

[27] J. Hein, H. Jagode, U. Sigrist, A. Simpson, and A. Trew. Parallel 3D-FFTs for multi-core nodes on a mesh communication network. In *Proceedings of the Cray User's Group (CUG) Meeting*, pages 1–15, Helsinki, Finland, 2008.

[28] H. Jagode, J. Hein, and A. Trew. Task placement of parallel multidimensional ffts on a mesh communication network. *University of Tennessee Knoxville, Technical Report No. ut-cs-08-613*, 2008.

[29] H. Jia-Wei and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing - STOC '81*, pages 326–333, New York, New York, USA, May 1981. ACM Press.

[30] P. Kogge and T. Dysart. Using the top500 to trace and project technology and architecture trends. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 28. ACM, 2011.

[31] P. Kogge et al. Exascale Computing Study: Technology challenges in acheiving exascale systems, Sept. 2008.

[32] S. Kumar, Y. Sabharwal, R. Garg, and P. Heidelberger. Optimization of All-to-All Communication on the Blue Gene/L Supercomputer. In *2008 37th International Conference on Parallel Processing*, pages 320–329. IEEE, Sept. 2008.

[33] H. T. Kung. Memory requirements for balanced computer architectures. In *Proceedings of the ACM Int'l. Symp. Computer Architecture (ISCA)*, Tokyo, Japan, 1986.

[34] G. H. Loh. 3D-Stacked Memory Architectures for Multi-core Processors. In *2008 International Symposium on Computer Architecture*, pages 453–464. IEEE, June 2008.

[35] J. McCalpin. Memory Bandwidth and Machine Balance in High Performance Computers. *IEEE Technical Committee on Computer Architecture (TCCA) Newsletter*, Dec. 1995.

[36] D. Pekurovsky and J. H. Goebbert. P3DFFT – highly scalable parallel 3d fast fourier transforms library. `http://www.sdsc.edu/us/resources/p3dfft`, November 2010.

[37] S. J. Pennycook, S. D. Hammond, S. A. Jarvis, and G. R. Mudalige. Performance analysis of a hybrid MPI/CUDA implementation of the NAS-LU benchmark. In *Proceedings of the International Workshop on Performance Modeling, Benchmarking and Simulation (PMBS)*, New Orleans, LA, USA, Nov. 2010.

[38] V. J. Reddi, B. C. Lee, T. Chilimbi, and K. Vaid. Web search using mobile cores: Quantifying and mitigating the price of efficiency. *ACM SIGARCH Computer Architecture News*, 38(3):215–314, June 2010.

[39] U. Sigrist. *Optimizing parallel 3D fast Fourier transformations for a cluster of IBM POWER5 SMP nodes.* PhD thesis, The University of Edinburgh, 2007.

[40] D. Takahashi. A Parallel 3-D FFT Algorithm on Clusters of Vector SMPs. In *Proceedings of Applied Parallel Computing: New Paradigms for HPC in Industry and Academia*, volume LNCS 1947, pages 316–323, 2001.

[41] J. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally, J. Meredith, J. Rogers, P. Roth, K. Spafford, et al. Keeneland: Bringing heterogeneous gpu computing to the computational science community. *IEEE Computing in Science and Engineering*, 13(5):90–95, 2011.

[42] H. Wang, S. Potluri, M. Luo, A. Singh, S. Sur, and D. Panda. MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters. *Computer Science-Research and Development*, pages 1–10.

[43] C. Young, J. Bank, R. Dror, J. Grossman, J. Salmon, and D. Shaw. A 32x32x32, spatially distributed 3D FFT in four microseconds on Anton. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 23. ACM, 2009.