

# Prospects for scalable 3D FFTs on heterogeneous exascale systems

Chris McClanahan\*, Kent Czechowski\*, Casey Battaglino†,  
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

{chris.mcclanahan,kentcz,cbattaglino3,kartik.iyer,pk.yeung,richie}@gatech.edu

## ABSTRACT

We consider the problem of implementing scalable three-dimensional fast Fourier transforms with an eye toward future exascale systems comprised of graphics co-processor (GPUs) or other similarly high-density compute units. We describe a new software implementation; derive and calibrate a suitable analytical performance model; and use this model to make predictions about potential outcomes at exascale, based on current and likely technology trends. We evaluate the scalability of our software and instantiate models on real systems, including 64 nodes (192 NVIDIA “Fermi” GPUs) of the Keeneland system at Oak Ridge National Laboratory. We use our analytical model to quantify the impact of both inter- and intra-node communication that impede further scalability. Among various observations, a key prediction is that although inter-node all-to-all communication is expected to be the bottleneck of distributed FFTs, it is actually intra-node communication that may play an even more critical role.

## 1. INTRODUCTION

The considerable interest in graphics co-processors (GPUs) for high-end computing systems raises numerous questions about performance, for both application developers and system architects alike. In essence, relative to current CPU-only clusters, GPUs imply clusters with fewer nodes having much higher per-node compute-densities than previously seen. However, this shift in compute density poses new challenges for overall scalability, both within the node and across the entire system.

In this paper, we ask what impact such a change will have on algorithm design and implementation, in the specific context of the three-dimensional fast Fourier transform (3D FFT). In nearly all modern implementations, the main communication step is an all-to-all exchange. As such, one would

reasonably expect network bandwidth to dominate all other performance factors, especially at exascale.

*Contributions.* Contrary to this intuition, we argue that it is actually the intra-node design that may play the more critical role, under business-as-usual assumptions. This claim is not just true today, where, unsurprisingly, relatively slow I/O bus communication (i.e., PCIe) can dominate performance. Rather, the surprise is that in the long-run, *PCIe does not matter* because current technology trends suggest that it is intra-node memory bandwidth actually scales more slowly than either I/O bus or network bandwidth.

To build this argument, this paper makes what we believe are three contributions to our current understanding of the role high-density compute nodes will have on future 3D FFT algorithms and implementations, summarized as follows.

1. **Software:** To better understand the impact GPU-enabled nodes will have on the 3D FFT, we first port the P3DFFT library [15, 40] to GPUs, and study its performance on two of the major United States-based GPU clusters.<sup>1</sup> P3DFFT uses the so-called pencil decomposition, making our GPU port the first *pencil-based* 3D FFT for a GPU cluster. Pending review of this paper, we will release this port as a set of open-source (GPL) patches to P3DFFT, which we refer to as DiGPUFFT (pronounced “dig-puffed,” for *Distributed GPU FFT*).
2. **Modeling:** Based on this implementation, we create an analytical performance model that accounts for computation and both inter- and intra- node communication. The inter-node terms can account for topology; the intra-node terms include memory bandwidth, cache, and I/O-bus (PCIe) effects, giving 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 (below). We instantiate and validate this model experimentally on existing systems.

<sup>1</sup>Namely, the Tesla C1060-based Lincoln system at National Center for Supercomputing Applications (NCSA) as well as the M2070 “Fermi”-based Keeneland system at Oak Ridge National Laboratory (ORNL).

3. **Predictions:** Using our model, we consider trends in architecture and FFT performance over the past 20-30 years and make a number of predictions about what we might expect approximately ten years hence, at exascale. One possible surprise is that even if the GPU were to remain a discrete device, it is actually *memory* bandwidth rather than I/O-bus bandwidth (today, PCIe) that is likely to be the intra-node limiter, barring certain memory technology changes as discussed below.

*Limitations.* Among our claimed contributions, we acknowledge several limitations.

First, our DiGPUFFT software does not employ novel algorithms per se. However, for GPU-based systems, it is a novel implementation, relative to the current state-of-the-art PKUFFT [6]. In particular, DiGPUFFT uses the much more scalable *pencil decomposition* rather than the slab decomposition of PKUFFT; and we evaluate DiGPUFFT on up to 192 NVIDIA “Fermi” GPUs on 64 nodes with QDR Infiniband of Keeneland, achieving roughly 700 Gflop/s on a large problem size, compared to 32 GPUs on 16 nodes and 1 GigaE at 120 Gflop/s for PKUFFT. More importantly, DiGPUFFT is the concrete basis for our performance model and predictions study.

Secondly, our model omits several factors that could play key roles in future systems. Chief among these are microscope memory and network contention effects, which we account for implicitly through parameter calibration. The net effect of this simplification is that our estimates are likely to be optimistic, as we show when we try to validate the model.

Lastly, regarding our predictions, we recall Niels Bohr’s famous quote that, “Prediction is very difficult, especially about the future.” Indeed, some of these predictions rely critically on business-as-usual trends that is subject to 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 main purpose in making any predictions at all is to influence the future rather than to obtain the “strictly correct answer.”

## 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 the 3D FFT algorithms and software at all scales of parallelism. Our paper most closely follows three recent studies.

The first study is the other major currently published distributed memory GPU 3D FFT code by Chen et al. [6], as mentioned in Section 1.

The second study is by Pennycook et al., who also consider inter- vs. intra-node communication issues in the context of the NAS-LU benchmark (parallel wavefront stencil), leveraging their earlier empirical modeling work [42]. Our model is by contrast more explicit about particular intra-node parameters, such as bandwidth, cache size, and I/O bus fac-

tors, and so our model adds improved algorithm-architecture understanding relative to this prior work.

The third study is Gahvari’s and Grop’s theoretical analysis of feasible latency and bandwidth regimes at exascale, using LogGP modeling and pencil/transpose-based FFTs as one benchmark [21]. 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 in making our projections.

Beyond these key studies, there is a vast literature on 3D FFTs [2–4, 7, 9–12, 17, 18, 22, 24, 26, 32, 44, 45, 47]. We call attention to just a few of these. For large problem sizes, the speed record is roughly 10-11 Tflop/s (1D) on the Cray XT5 and NEC SX-9 machines [1]. At our performance level ( $\approx 700$  Gflop/s), the closest report is for a 2005 Cray XT3 run, which used over 5000 nodes (and one processor per node) [1] compared to our 64 nodes and 3 GPUs per node. 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  $4 \mu\text{s}$  (614 Gflop/s) [47], which our code can only attain for relatively much larger problem sizes (see Section 4).

Among the other 3D FFT implementations, there are two broad classes that complement and would improve our work. The first class considers highly-optimized all-to-all implementations with a variety of sophisticated tricks like non-blocking asynchronous execution, overlap, and off-loading [12, 28, 29], which might improve our implementations by 10-30%.<sup>2</sup> The second class of implementations are single GPU FFTs, which are a building block for our code [23, 38, 39]. However, at present we believe from experiment and comparison to published results that the current NVIDIA implementation on which we rely compares well with these other approaches.

## 3. BACKGROUND ON THE 3D FFT

The vast majority of modern FFT implementations use some variation of the standard Cooley-Tukey algorithm. The classical parallel algorithms are the *binary exchange* and *transpose* algorithms [34]. The basic high-level trade-off between them is that, in 1D, the binary exchange method on  $p$  processors sends  $\mathcal{O}(\log p)$  messages of total volume  $\mathcal{O}\left(\frac{n}{p} \log p\right)$  words, compared to the transpose algorithm’s  $\mathcal{O}(p)$  messages of total volume  $\mathcal{O}\left(\frac{n}{p}\right)$ , ignoring overlap. (Recall from Section 2 that overlap may result in up to 10-30% improvements in practice.) Thus, we expect better performance from the transpose algorithm at large  $n$ , where we will be bound by network- or memory-bandwidth; and just the opposite for small  $n$ , where we expect to be latency bound.

Of these, nearly all modern parallel 3D FFT implementations use the transpose algorithm, for which there are two major variants: the so-called *slab* and *pencil* decompositions. We illustrate these variants in Figure 1.

<sup>2</sup>There are  $2\times$  demonstrations of improvement in the collective itself for very slow networks [13].

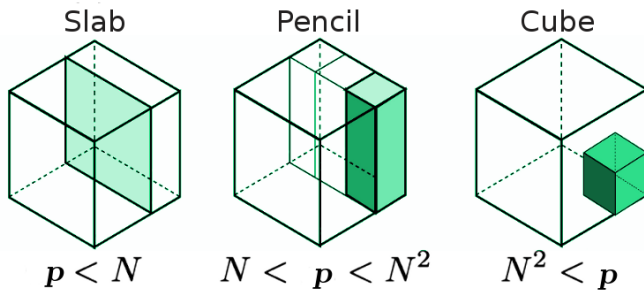


Figure 1: Parallel  $N^3$  3D FFT data distribution over  $p$  processors [31]

In the slab approach, the data is partitioned into 2D slabs along a single axis. For example, to compute a  $N*N*N$  FFT on  $p$  processors, each processor would be assigned a 2D slab of size  $N*N*(N/p)$ . Although this method helps in reducing communications costs, “...the scalability of the slab-based method is limited by the number of the data elements along a single dimension of the three-dimensional FFT” [15], as a  $128^3$  3D FFT scales to just 128 processors.

In the pencil approach, we partition the data into 1D pencils to overcome the scaling limitation inherent in FFT libraries based on the 1D (or slab) decomposition [41]. For example, to compute a  $N * N * N$  FFT on  $p_1 * p_2$  processors, each processor would be assigned a 1D pencil of size  $N * (N/p_1) * (N/p_2)$ . This approach increases scalability in the maximum number of processors capable of being used to  $N^2$ , for an  $N^3$  size FFT, compared to a maximum of  $N$  processor in the slab decomposition. In contrast to the slab approach, pencils enables scaling a  $128^3$  FFT, up to  $128^2 = 16,384$  processors [15].

#### 4. P3DFFT AND DIGPUFFT

The Parallel Three-Dimensional Fast Fourier Transform library, or P3DFFT, implements the distributed memory transpose algorithm using a pencil decomposition [41]. P3DFFT is freely available under a GPL license. A major use of P3DFFT is for a Direct Numerical Simulation (DNS) turbulence application for the 32,768 core Ranger cluster (at TACC).

P3DFFT version 2.4 serves as the basis for our DiGPUFFT code. On each node P3DFFT computes local 1D FFTs using third party FFT libraries, which by default is FFTW, though IBM’s ESSL and Intel’s MKL may serve as drop-in replacements. For DiGPUFFT, we developed a custom CUFFT wrapper for use within P3DFFT, making CUFFT an additional local FFT option. Pending the review of this paper, we will release DiGPUFFT as open-source software on Google Code.

##### 4.1 Performance Results

We performed our experiments with P3DFFT and DiGPUFFT on Keeneland, a National Science Foundation Track 2D Experimental System based on the HP SL390 server 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. Nodes are interconnected with single rail, QDR Infiniband. Unless otherwise specified, results were measured using the following software stack: Intel C/C++ Compiler version 11.1, OpenMPI 1.4.3, and NVIDIA CUDA 3.2.

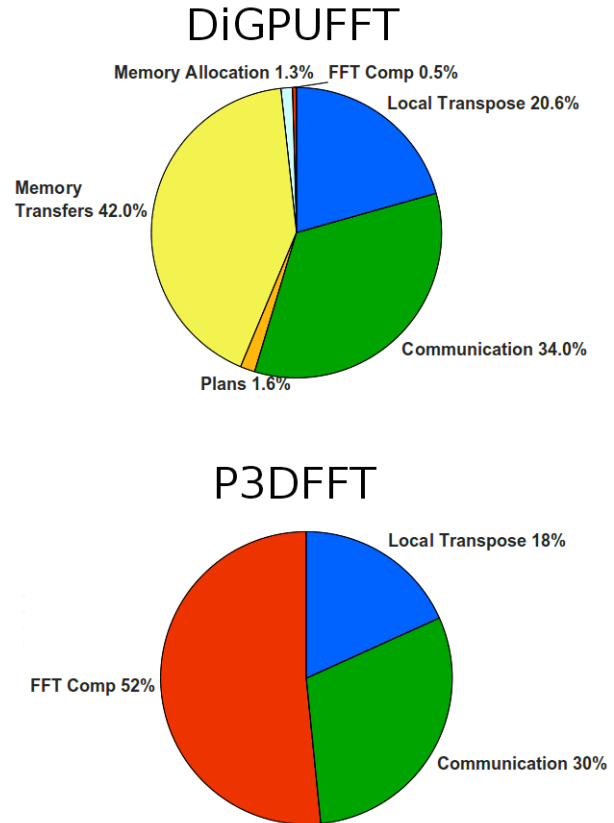
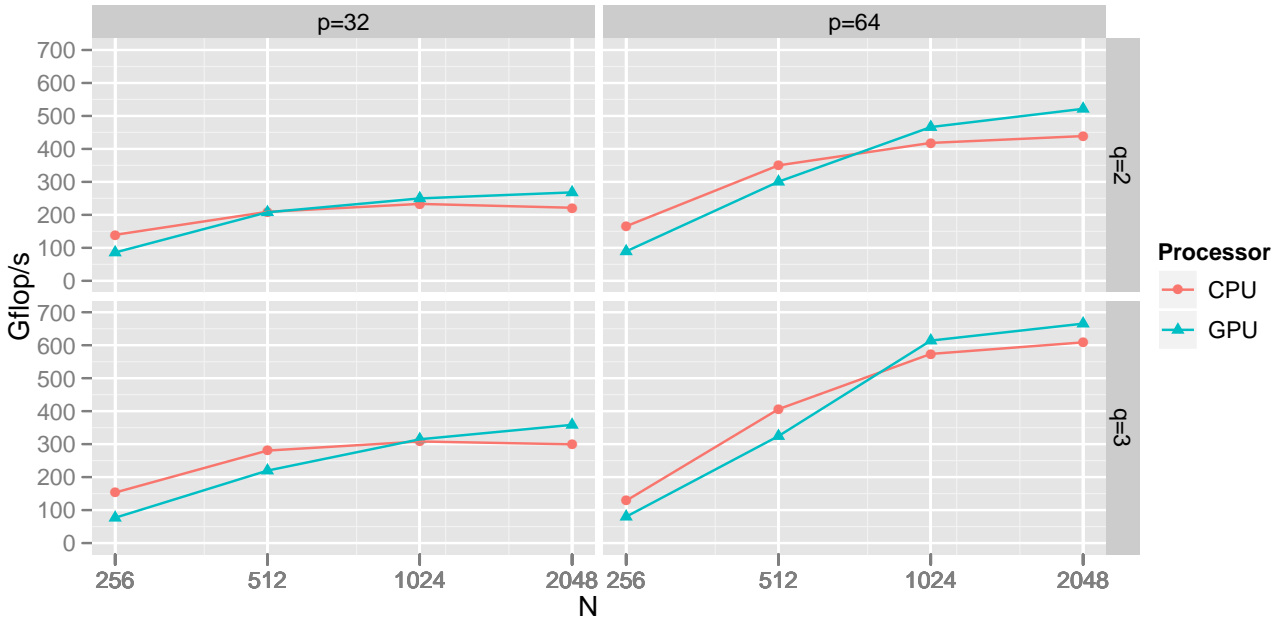


Figure 2: Accounting for the time spent during DiGPUFFT/P3DFFT. Results are from a 64 node run on the Keeneland cluster with a problem size of  $2048^3$ , and 3 MPI tasks per node

We summarize the performance of off-the-shelf P3DFFT and DiGPUFFT in Figure 3, which shows transform sizes of  $256^3$  to  $2048^3$  run on 32 and 64 nodes in Keeneland, at either 2 and 3 MPI tasks per node. For the P3DFFT/CPU runs, we configure P3DFFT to use FFTW in multithreaded mode. We make several observations:

- Doubling nodes ( $p$ ) from 32 to 64 roughly doubles performance, showing strong scaling across nodes.
- Increasing the number of GPUs per node from 2 to 3 delivers  $\approx 30\%$  performance improvement, showing strong scaling within a node.
- The win from GPU over CPU is modest and may be below expectations, showing just roughly 10% to 20% improvements.
- There is a performance cross-over point between the CPU vs. GPU implementations, which reflects the GPU memory transfer overhead.



**Figure 3: DiGPUFFT performance on an  $N \times N \times N$  FFT problem, using  $p$  nodes of Keeneland with  $q$  MPI tasks per node. The CPU curves are MPI + multithreaded FFTW; GPU curves use CUFFT on a total of  $p \cdot q$  GPUs.**

- The CPU’s performance levels off more quickly than the GPU’s performance, showing at least some modest scalability improvements due to the use of GPUs.

At smaller FFT sizes such as  $256^3$ , the CPU outperforms the GPU, due partly to the CPU still doing the local transposes, and the high cost of host-to-device memory transfers, as shown in Figure 2. As the FFT sizes grow, the GPU’s faster compute time quickly overcomes this additional overhead. We analyze Figure 2 in the following sections.

## 4.2 Memory Constraints

NVIDIA’s high-end GPUs provide up to 6 GB of memory, significantly less than typical host memory sizes. For large data-sets, memory space can become a scarce commodity. For FFT kernels, GPU memory must hold the frequency values as well as the  $\mathcal{O}(n)$  “twiddle factors” that are generated with the CUFFT planning procedures. Such plans mimic the equivalent construct in FFTW. A plan generated by CUFFT for an FFT of size  $n$  will use 168 MB of overhead plus an additional  $\mathcal{O}(n)$  bytes to store the twiddle factors. For DiGPUFFT, this memory constraint limits the number of plans that can be precomputed or overlapped because there is not enough memory to store them all for the duration of the FFT calculation.

As Figure 2 shows, 1.3% of the time along the critical path is spent computing the CUFFT plans. An additional 1.6% of the time is spent allocating and de-allocating memory on the GPU to accommodate the different pencil batch sizes used during the FFT computation. With enough memory to compute all of the plans and allocate all of the blocks ahead of time, we would expect to see a 3% speedup. More

GPU memory would also make it possible to run FFTs with larger pencil lengths.

## 4.3 PCIe Bottleneck

Figure 2 also shows that 52% of the time along the critical path is spent transferring frequency values from the host memory to GPU memory then transferring the computed results back to host memory before transferring them across the network.

Consider the 1D FFT using the transpose algorithm of size  $n$  with  $p$  GPUs. During the first phase,  $n/p$  words are transferred across the PCIe bus to the GPU memory, a local FFT is computed on the  $n/p$  values, and the resulting  $n/p$  values are transferred back across the PCIe bus to the host memory. Using the peak bandwidth of the PCIe bus ( $\beta_{PCIe}=8$  GB/s) and the peak GPU computational throughput for a local FFT ( $C_{fft}=380$  Gflop/s; see Table 1), we can calculate the effective computation time for the GPU,  $T_{cufft}$ :

$$T_{cufft}(n) = \frac{5n \log n}{C_{cufft}} + 2 \frac{n}{\beta_{PCIe}} \quad (1)$$

versus the time for the CPU version, which does not involve transfers across the PCIe bus:

$$T_{fftw}(n) = \frac{5n \log n}{C_{fftw}} \quad (2)$$

Comparing  $T_{cufft}(n)$  and  $T_{fftw}(n)$  for realistic values of  $n$ , ( $2^{10} \leq n \leq 2^{30}$ ), we observe that the effective speedup of the GPU is only  $2.1\times$  to  $3.8\times$ , significantly less than the  $14\times$  speedup suggested by Table 1. In fact, the GPU only reaches the  $C_{cufft}$  peak reported in Table 1 on a few special-case values of  $n$ . For most values of  $n$ , the GPU only achieves a

small fraction of  $C_{cufft}$  and no value of  $n$  greater than 4096 achieves more than 240 Gflop/s.

	Single Core of a CPU	One GPU
Model	6-core Intel X5660 2.8 GHz	NVIDIA Tesla M2070
FFT Software	FFTW 3.2.1	CUFFT 3.2
Hardware Peak	22.4 Gflop/s	1030 Gflop/s
Observed FFT Peak	4.7 Gflop/s	338 Gflop/s

Table 1: Local FFT Performance: CPU vs GPU

The model is more complicated when there are multiple GPUs on the node. Spafford et al. have shown that GPU $\rightleftharpoons$ CPU memory transfer bandwidth is impacted by the locality of the CPU and GPU [27]. For example, a memory transfer between the wrong GPU-CPU pair can be up to 31% slower than a transfer between properly paired GPU-CPU. Figure 4 shows the hardware layout of a node in the Keeneland cluster.

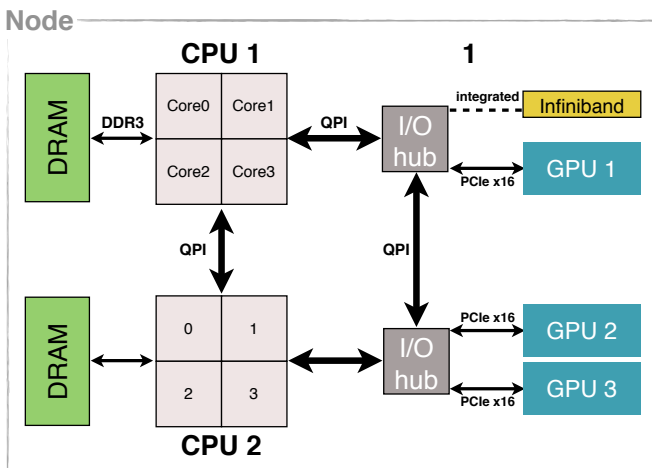


Figure 4: Architecture of a Keeneland node

We first verify the experimental results of Spafford et al. [27], and then extend them to show the measured bandwidth across various host and device combinations using a standard ping-pong bandwidth test. Table 2 reports the results.

The key observation from the Table 2 is that intra-node communication does not necessarily scale evenly as extra

	Bandwidth (GB/s)
CPU 1 $\rightleftharpoons$ GPU 1	6.0
GPU 2 $\rightleftharpoons$ GPU 3	4.7
CPU 1 $\rightleftharpoons$ GPU 2/3	4.6
GPU 1 $\rightleftharpoons$ GPU 2/3	4.1

Table 2: Intra-node NUMA costs in the Keeneland architecture

devices and processors are added. Spafford et al. introduced the notion of path length. We see that the bandwidth between CPU1 and GPU2 is bounded by the same PCIe bus bandwidth as GPU1 to CPU1 communication, and the longer path also impacts its bandwidth. Although, we discuss in Section 6, the good news is that the PCIe bandwidth bottleneck is likely to improve in the near future. In fact, early development in MVAICH-GPU, a MPI implementation that will allow direct communication between the GPU and network interface, has already shown a 45% performance improvement over the indirect `cudaMemcpy()` + `MPLSend` [46].

#### 4.4 Local Transpose

In addition to the local FFT, each node must also compute a local transpose to reshuffle the data. This operation can be a costly because it involves loading and storing nearly every value in the local data set. Indeed, our results show that this step can be as costly as the local FFT itself. Other FFT implementations have shown similar relative costs [17, 47].

Chen et al. describe a clever technique to embed the local transpose into GPU-to-host and host-to-GPU transfers, thereby hiding the cost of the local transpose [6]. This can lead to a significant speedup over a GPU implementation that perform the transpose in a separate step. Looking forward, we expect the newly developed “GPUDirect” features, which make it possible to send data directly from one GPU’s memory to a remote GPU’s memory, will eliminate the costly host-to-GPU and GPU-to-host steps. Unfortunately, at present our implementation does not yet take advantage of the embedded transpose, and due to GPU memory constraints performs all local transposes on the CPU.

### 5. MODELING COMMUNICATION COSTS

#### 5.1 Analytical Model

We consider the model of the 3D FFT using the pencil decomposition of the transpose method with  $p$  nodes and problem size  $N = n^3$ . We consider the communication costs incurred both within the node and between nodes.

##### 5.1.1 Intra-node Communication

In the pencil decomposition (Section 3), there are  $n^2$  1D FFTs of length  $n$  distributed evenly among the  $p$  nodes. Using the Cooley-Tukey recursive formulation of the DFT, there are approximately  $5n \log n$  floating point operations in a 1D FFT of length  $n$ . In total, the 1D FFTs must be computed 3 times, once for each dimension, for a total of  $\frac{3n^2}{p}$  1D FFTs per node. The time to compute these FFTs is the total number of floating point operations divided by the peak computation throughput  $C_0$ , in flops per unit time.

$$T_{flops} = 3 \times \frac{\left(\frac{n^2}{p}\right) 5n \log n}{C_0} \quad (3)$$

During the FFT, each data point must be loaded and stored from memory at least once. For a local 1-dimensional cache-oblivious FFT [20], the number of cache misses grows on the order of  $Sort(n) = \Theta(1 + (n/L)(1 + \log_Z n))$ , where  $Z$  and  $L$  are the cache and line sizes [19]. As computational throughput continues to outpace memory performance, we expect intra-node performance to adhere to this lower-bound.

Model	CPU Today (ms)	GPU Today (ms)	CPU 10 years (ms)	GPU in 10 years (ms)
$T_{flops}$	71	6	.1	.1
$T_{mem}$	224	13	19	1
$T_{comm}$	229	229	11	11

**Table 3: Example model times for a given machine**

The  $\frac{n^3}{p}$  data points on the node also need to be reshuffled before being exchanged with other nodes. This step is the *local transpose* step. The time for these memory access is the number of memory accesses divided by the memory bandwidth  $\beta_{mem}$ .

Thus, for some constant  $A$  and sufficiently large  $n$ , a 3D FFT will incur the following memory costs within a node:

$$T_{mem} = 3 \times \frac{n^2}{p} \cdot \frac{A(1 + (n/L)(1 + \log_Z n)) \cdot L}{\beta_{mem}} + 2 \times \frac{\frac{2n^3}{p}}{\beta_{mem}} \quad (4)$$

### 5.1.2 Inter-node Communication

During the 3D FFT, the node must exchange its  $\frac{n^3}{p}$  data points with other nodes on the network at least twice, typically with the `MPI_AllToAll` collective. Technically, only  $\frac{(p-1)(n^3)}{p^2}$  data points are exchanged per round, but we simplify it to  $\frac{n^3}{p}$  because  $\frac{p-1}{p}$  is nearly 1 for typical values of  $p$ . The communication time is therefore bounded by the total number of words exchanged across the network divided by the network bandwidth in/out of the node, denoted  $\beta_{link}$ . In the following subsections we will look more closely at how the network topology impacts the communication time. As a strict lower bound (i.e, on a fully connected network with full overlap):

$$T_{comm} \geq 2 \times \frac{\frac{2n^3}{p}}{\beta_{link}} \quad (5)$$

Comparing the relative times for the  $T_{flops}$ ,  $T_{mem}$ , and  $T_{comm}$  provides an easy way to explore the potential bottlenecks of a machine represented by the  $C_0$ ,  $\beta_{mem}$ , and  $\beta_{link}$  values. Table 3 shows the computed values for Keeneland using only CPUs or only GPUs. In the CPU case we see that time for the floating point calculations is relatively minor compared to the network times, as expected. But the more surprising result is that memory access time is significantly more than the flops and relatively close to the network time. The use of the three GPUs on each node boosts the memory and floating point throughput and has a profound impact on the relative value of  $T_{mem}$  in comparison to  $T_{link}$ . Unfortunately, as our DiGPUFFT analysis shows, intra-node communication bottlenecks prevent the GPU from reaching this potential. These times roughly correspond to the time distributions reported from the P3DFFT experiments.

If it were somehow possible to drive  $T_{mem}$  to zero, then the all-to-all personalized communication step becomes the limiter and optimizing this collective becomes critical to im-

proving performance [33]. For instance, maximizing communication overlap of this collective minimizes time lost due to latency. One state-of-the-art implementation achieves 95% overlap on the BlueGene/L [13].

In our model we assume that inter-node communication time ( $T_{comm}$ ) and intra-node communication time ( $T_{node} = \max(T_{flops}, T_{mem})$ ) are independent:  $T_{total} = T_{comm} + T_{node}$ , although in an ideal case they could be perfectly overlapped.

In the 3D FFT, we estimate communication time on a hypercube to be the cost of performing two all-to-all communication steps [34]:

$$T_{comm}^T = 2 \times \left[ (p-1)\alpha_{link} + \frac{n(p-1)}{p^2 \cdot \beta_{link}} \right]$$

where  $\alpha_{link}$  is inter-node latency. We use  $T_{comm}^T$  to denote the transpose algorithm, as distinct from the binary-exchange algorithm modeled below.

For a torus topology, Eleftheriou et al. give lower-bounds on the all-to-all communication time with the expression

$$T \geq \frac{V_{received} N_{hops}}{N_{links} \beta \cdot f}$$

Where  $N_{hops}$  is the average number of hops and  $N_{links}$  is the number of links connected to any node. This expression is constant regardless of the dimension of the torus [17].

### 5.1.3 Algorithm Selection

One notable FFT method that does not involve an all-to-all communication is the Binary Exchange algorithm. This algorithm requires more bandwidth than the transpose method by a factor of  $\log p$  but requires  $\log p$  communication steps with neighboring nodes, compared with the single global communication step required for the transpose method. This can decrease communication time in high-latency networks, or in the case of strong-scaling when the volume of data is not enough to take advantage of all of a machine's bandwidth. In the 3D case, we must carry this out in three phases:

$$T_{comm}^B = 3 \times \left[ (\log p)\alpha + \frac{n \log p}{p \cdot \beta} \right]$$

Figure 5.1.3 contrasts the transpose and binary-exchange variants. For smaller values of  $n$ , the lower latency costs could make binary-exchange the right choice. However, in most cases this would be an inefficient use of a large number of nodes, and it would be smarter to scale to a smaller network. However, in some cases the FFT is part of a larger code and it is necessary to run a smaller problem across a large network, for instance a recent turbulence simulation running on 8000 nodes of Jaguar [15].

### 5.1.4 Bandwidth limits

Any problem able to utilize a system efficiently is expected to be communication-bound, either intra-node or inter-node.

In 1999, Edelman et al. [16] projected that the rapid rate of increase in floating point performance relative to communication bandwidth improvements would lead to prohibitively

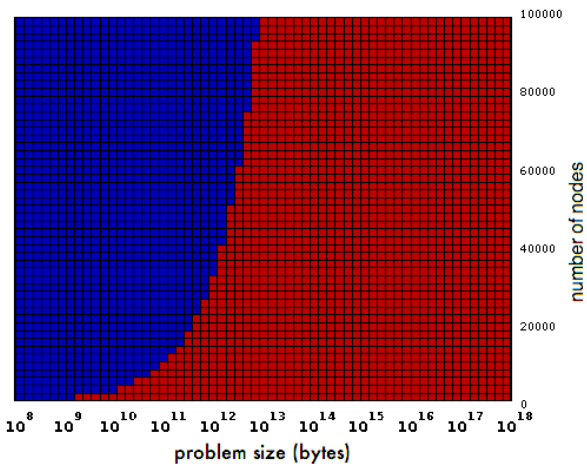


Figure 5: Binary-Exchange (blue) vs. Transpose (red) algorithm for typical values of  $\alpha, \beta$ . Colors indicate the algorithm with lower communication costs.

large communication costs in the future. As a solution, they proposed an FFT approximation algorithm for distributed memory clusters that would reduce the total communication cost. This specific approach has not seen practical implementation, though there have been other attempts to overcome communication bandwidth costs by compressing data before sending it across the network [30, 36]. The compression rate is highly dependent on the problem domain and the amount of computation available to spend compressing it, but researchers have shown that large sets of double-precision floating point values have the potential to be compressed down to a quarter the size with only modest computation costs [43]. Using a lossy or lossless compression to reduce message sizes would have the effect of artificially boosting network bandwidth linearly. As systems become even more communication-bound, we essentially have free cycles carry this out.

## 5.2 Model Validation

The network models presented above provide a lower bound on communication time. In this section, we check them against real machines. We timed the communication primitives on 4,096 nodes of Hopper, a 1.288 Peta-flop Cray XE6 housed at the the National Energy Research Scientific Computing Center. Hopper’s 6,392 nodes form a 3D torus with a Gemini interconnect.

Results are shown in Figure 6. As expected, the transpose method outperforms the binary-Exchange when the problem size is large. Recall that this is due largely to the fact that the binary-exchange method sends  $\log p$  times as much data as the transpose method. It is also important to note that the binary-exchange is ideal for a hypercube topology because it only exchanges data with neighboring nodes, but on a 3D torus, such as Hopper, this no longer holds.

Equation (4) describes intra-node communication. We validate this behavior by benchmarking FFTW3.2.2 across two Intel Xeon E5530 processors, as illustrated in Figure 7

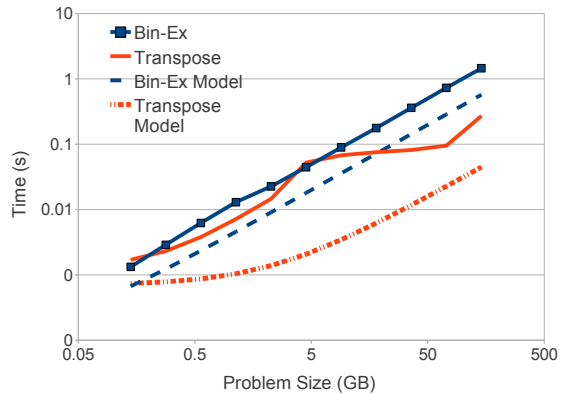


Figure 6: Communication costs: Binary-Exchange vs Transpose on 4,096 nodes of Hopper

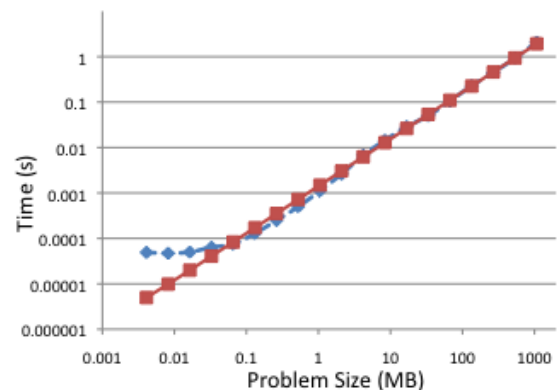


Figure 7: Intra-node 1D FFTW performance on power-of-two problem sizes. The red line is the memory cost expected by Equation (4), and the blue line is measured performance.

## 6. PROJECTING FORWARD

### 6.1 Swim Lanes

Recent discussions surrounding the direction of supercomputing has separated into two competing strategies (commonly referred to as the two swim lanes): Many-Cores machines (MC) and Many-Threads machines (MT)

Debates about future exascale architectures frequently contrast two candidate strategies, or “swim lanes.” Some refer to these strategies as the many-core (embedded CPU-like) vs. many-thread (GPU-like) approaches [14, 25].

The MT strategy tries to hide latency by overlapping the latency with concurrency, rather than reducing the latency outright. This strategy can be seen in the Tiahne-1A GPU Cluster. Processor dies are monopolized by relatively simple floating point units with very little die-area devoted to cache or processor optimizations (e.g. branch prediction or instruction reordering). Unfortunately, MT processors depend on an enormous amount of parallelism from the ap-



Parameter	Keeneland values	doubling time (in years)	10-year increase factor
Cores: $p_{\text{cpu}}$	12	1.87	40.7×
$p_{\text{gpu}}$	448		
Peak: $p_{\text{cpu}} \cdot C_{\text{cpu}}$	268 Gflop/s	1.7	59.0×
$p_{\text{gpu}} \cdot C_{\text{gpu}}$	1 Tflop/s		
Memory bandwidth: $\beta_{\text{cpu}}$	25.6 GB/s	3.0	9.7×
$\beta_{\text{gpu}}$	144 GB/s		
Fast memory: $Z_{\text{cpu}}$	12 MB	2.0	32.0×
$Z_{\text{gpu}}$	1MB		
I/O device: $\beta_{\text{I/O}}$	8 GB/s	2.39	18.1×
Network bandwidth, $\beta_{\text{link}}$	10 GB/s	2.25	21.8×

**Table 4: Using the hardware trends we can make predictions about relative performance of future hardware.**

plication and implement a wide SIMD vector width because the memory bandwidth necessary to fetch instruction for thousands of processors every cycle can be prohibitive. The MT strategy results in very high floating point density, and therefore have fewer nodes than their MC counterparts.

Alternatively, the MC strategy takes aggressive measures to reduce latency. Large portions of the processor die are devoted to caches with the aim of reducing the number of out-of-die accesses. These machines are exemplified by the direction of the Blue Gene line of clusters.

While the concepts of the two swim lanes are still rather nebulous and evolving on a regular basis, its not too soon to begin thinking about how the differences between MC and MT machines impacts particular algorithms. In the case of the FFT, the models presented in this paper can be used to compare and contrast the two strategies.

## 6.2 Predictions

Armed with the performance model of Section 5, a natural exercise is to consider how performance and scaling could change in light of current technology trends. Our projections are based primarily on Table 4, which shows current values on Keeneland for various machine parameters, as well as the time (in years) for a particular parameter to double and the factor by which current values will increase in ten years. Note that for several parameters we separate CPU vs. GPU performance, though since the two are fundamentally based largely on similar technologies (e.g., silicon and manufacturing processes), we hypothesize that the rates of growth will be identical. (Note to reviewers: Appendix A explains how we derived these values.)

**Prediction:** Under business-as-usual assumptions (Table 4), a 3D FFT will achieve 1.23 Pflop/s on a GPU-like exascale machine in 2020. This value is 0.1% of peak, compared to today’s fraction of peak, which is about 0.5%. Interestingly, the communication time due to memory bandwidth just starts to exceed the communication time due to network bandwidth at this time.

To obtain this estimate, we extrapolated the various system parameters as suggested by Table 4, used them to determine

the form (e.g., number of nodes) required to get a system running at 1 Eflop/s, selected a problem size according to the methodology of Gahvari and Gropp [21], and then evaluated our performance model to estimate execution time.

**Prediction:** Intra-node memory bandwidth will dominate future FFT performance, but not for what might be the expected reasons. The trends of Table 4 suggest that increases in network bandwidth are, perhaps surprisingly, actually outstripping increases main memory bandwidth. Thus, barring memory technology game-changers, network bandwidth will eventually catch up to and surpass memory bandwidth.

It is not entirely clear to us why this observation would be true. One conjecture 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 [5].

The primary mechanism proposed to improve the rate of memory bandwidth scaling relative to compute scaling is stacked memory [37]. Applying our model and known I/O complexity estimates for the FFT, we can establish that a node’s computation and memory transfers will be balanced when  $T_{\text{mem}} \leq T_{\text{comp}}$  [8, 35], which implies that

$$\frac{p \cdot C_0}{\beta} \leq \mathcal{O} \left( \log \frac{Z}{p} \right) \quad (6)$$

Theoretically, stacked memory makes it possible to keep the left-hand side constant over time. Although  $p$  grows faster than  $Z$ , it enters into this inequality through the log and so will not decrease too quickly.

**Prediction:** We predict that the “CPU-like swim lane” will perform better in the future. Based on our model, the system-wide performance of an FFT in 2020 will be 4.87 PFlop/s.

## 7. CONCLUSIONS

For us, the interesting finding of this paper is that I/O-bus (PCIe) and network bandwidth are, in fact, not 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. This suggests that much more architectural emphasis on intra-node issues will have the biggest pay-off in the long-run.

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 solve the problem. The most interesting solution for FFT-like computations, which are like sorting, is most likely stacked memory if it can indeed deliver proportional scaling of memory bandwidth to core counts.

Looking forward, we believe our basic modeling methodology and its level of detail will provide similar kinds of insights for other computations. We think this style of analysis could be especially useful in the context of algorithm-architecture co-design, a notion we have outlined for intra-



node designs elsewhere [8].

## 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] 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.
- [5] 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.
- [6] Y. Chen, X. Cui, and H. Mei. Large-scale FFT on GPU clusters. ICS'10, 2010.
- [7] 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.
- [8] K. Czechowski, C. Battaglini, C. McClanahan, A. Chandramowlishwaran, and R. Vuduc. Balance principles for algorithm-architecture co-design. In Proc. USENIX Wkshp. Hot Topics in Parallelism (HotPar), Berkeley, CA, USA, May 2011. (*accepted*).
- [9] 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] J. Doi and Y. Negishi. Overlapping methods of all-to-all communication and FFT algorithms for torus-connected massively parallel supercomputers. Supercomputing, pages 1–9, 2010.
- [14] J. Dongarra et al. The international exascale software project roadmap. IJHPCA, 25(1):3–60, 2011.
- [15] D. Donzis, P. Yeung, and D. Pekurovsky. Turbulence simulations on  $o(10^4)$  processors. 2008.
- [16] A. Edelman, P. McCorquodale, and S. Toledo. The future fast Fourier transform? SIAM Journal on Scientific Computing, 20(3), 1999.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] M. Frigo, Steven, and G. Johnson. The design and implementation of FFTW3. In Proceedings of the IEEE, pages 216–231, 2005.
- [21] H. Gahvari and W. Gropp. An introductory exascale feasibility study for ffts and multigrid. IEEE, 2010.
- [22] 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.
- [23] N. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli. High performance discrete Fourier transforms on graphics processors. In 2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis, number November, pages 1–12, Austin, TX, USA, Nov. 2008. IEEE.
- [24] 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.
- [25] 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.
- [26] 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.

- [27] K. Jeffrey and S. Vetter. Quantifying NUMA and Contention Effects in Multi-GPU Systems. 2011.
- [28] L. Kale, S. Kumar, and K. Varadarajan. A framework for collective personalized communication. In Proceedings International Parallel and Distributed Processing Symposium, volume 00, page 9. IEEE Comput. Soc, 2003.
- [29] K. Kandalla, H. Subramoni, K. Tomko, D. Pekurovsky, N. Dandapanthula, S. Sur, and D. K. D. K. Panda. Improving Parallel 3D FFT Performance using Hardware Offloaded Collective Communication on Modern InfiniBand Clusters, 2010.
- [30] J. Ke, M. Burtscher, and E. Speight. Runtime compression of MPI messages to improve the performance and scalability of parallel applications. In Proceedings of the 2004 ACM/IEEE conference on Supercomputing, page 59. IEEE Computer Society, 2004.
- [31] J. Kim. Blue waters : Sustained petascale computing. PDF, <https://wiki.ncsa.illinois.edu/download/attachments/17630761/INRIA-NCSA-WS4-dtakahashi.pdf>.
- [32] S. Kumar, Y. Sabharwal, R. Garg, and P. Heidelberg. 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] S. Kumar, Y. Sabharwal, R. Garg, and P. Heidelberg. Optimization of all-to-all communication on the Blue Gene/L supercomputer. In Proceedings of the 2008 37th International Conference on Parallel Processing, ICPP '08, pages 320–329, Washington, DC, USA, 2008. IEEE Computer Society.
- [34] V. Kumar, A. Grama, A. Gupta, and G. Karypis. Introduction to parallel computing: design and analysis of algorithms. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.
- [35] H. T. Kung. Memory requirements for balanced computer architectures. In Proceedings of the ACM Int'l. Symp. Computer Architecture (ISCA), Tokyo, Japan, 1986.
- [36] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. Visualization and Computer Graphics, IEEE Transactions on, 12(5):1245–1250, 2006.
- [37] G. H. Loh. 3D-Stacked Memory Architectures for Multi-core Processors. In 2008 International Symposium on Computer Architecture, pages 453–464. IEEE, June 2008.
- [38] A. Nukada and S. Matsuoka. Auto-tuning 3-D FFT library for CUDA GPUs. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis - SC '09, number 30, page 1, New York, New York, USA, 2009. ACM Press.
- [39] A. Nukada, Y. Ogata, T. Endo, and S. Matsuoka. Bandwidth Intensive 3-D FFT kernel for GPUs using CUDA. In Proceedings of the ACM/IEEE Conf. Supercomputing (SC), volume 11pages.
- [40] D. Pekurovsky. P3DFFT introduction. <http://code.google.com/p/p3dfft/>, November 2010.
- [41] D. Pekurovsky and J. H. Goebbert. P3DFFT – highly scalable parallel 3d fast fourier transforms library. <http://www.sdsc.edu/us/resources/p3dfft>, November 2010.
- [42] 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.
- [43] P. Ratanaworabhan, J. Ke, and M. Burtscher. Fast lossless compression of scientific floating-point data. 2006.
- [44] U. Sigrist. Optimizing parallel 3D fast Fourier transformations for PhD thesis, The University of Edinburgh, 2007.
- [45] 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.
- [46] 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.
- [47] 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.

Year	GB/s
1991	105.497
1992	22.4365
1993	2.3737
1994	12.3839
1995	168.193
1996	359.841
1997	437.358
1998	12.7826
1999	607.492
2000	288.345
2001	101.6956
2002	213.0243
2003	1008.594
2004	437.01
2005	42.632
2006	4385.585
2007	227.059
2008	805.8046
2009	445.869
2010	4384.461
2011	5859.367

**Table 5: STREAM bandwidth.** We show the best value for any of the STREAM benchmarks reported in each year, as of the time of submission of this paper.

## APPENDIX

### A. TREND DATA: SOURCES

We include this appendix as optional reading for reviewers interested in how we arrived at the trends and projections of Section 6. In particular, our doubling-time estimates are based on the sources listed below.

*STREAM memory bandwidth.* We extrapolate a trend using data collected at the STREAM website,<sup>3</sup> where we consider the best reported value in each year since 1991. “Best” means we consider all reported values for any platform, including data from the “tuned” category. By this methodology, our estimate should provide a reasonable estimate of best-case sustainable memory bandwidth. The specific values we used appear in Table 5 and visualized in Figure 8.

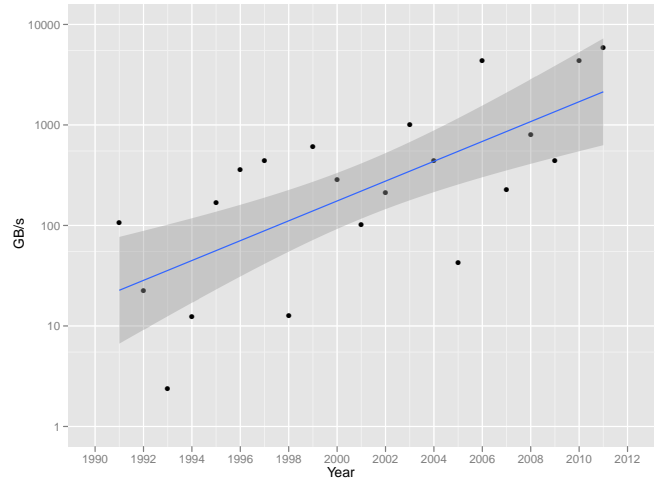
*Network bandwidth.* We begin with data collected by Dally, which covers the period 1986–2005.<sup>4</sup> We then independently collected the same data for systems that made the Top 500 list<sup>5</sup> since 2005, as listed in Table 6.

*Intra-node PC I/O Bus Bandwidth.* Since the introduction the IBM PC’s Industry Standard Architecture (ISA) bus line, there have been several generations of I/O bus standards. We show the trends for these busses in Table 7, which

<sup>3</sup><http://streambench.org>

<sup>4</sup>[http://2007.nocsymposium.org/keynote1/dally\\_nocs07.ppt](http://2007.nocsymposium.org/keynote1/dally_nocs07.ppt)

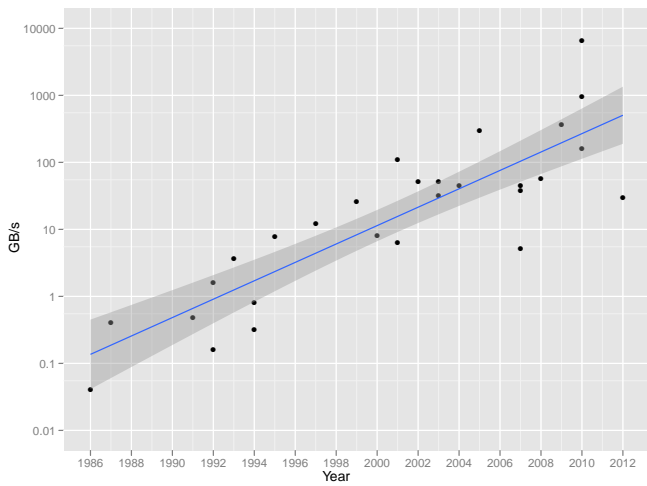
<sup>5</sup><http://top500.org>



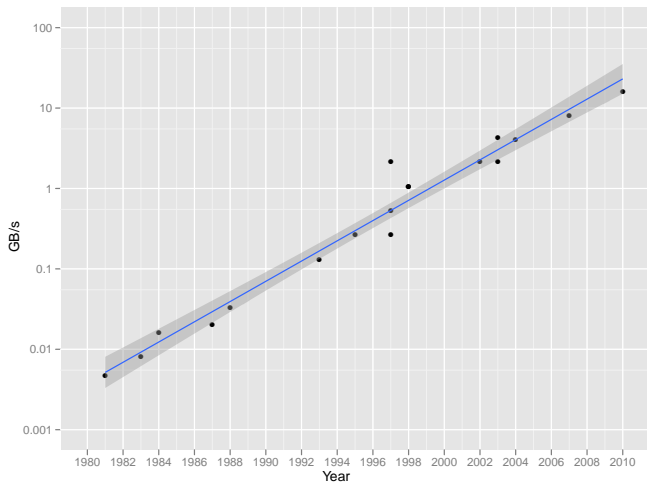
**Figure 8: STREAM bandwidth.** For methodology, see Table 5.

Year	Bandwidth (Gbit/s)	System
1986	0.32	Torus Routing Chip
1987	3.2	Intel iPSC/2
1991	3.84	J-Machine
1992	1.28	CM-5
1992	12.8	Intel Paragon XP
1993	28.8	Cray T3D
1994	6.4	IBM Vulcan
1994	2.56	MIT Alewife
1995	63.0	Cray T3E
1997	96.0	SGI Origin 2000
1999	204.8	AlphaServer GS320
2000	64.0	IBM SP Switch2
2001	51.2	Quadrics QsNet
2001	875.0	Velio 3003
2002	409.6	Cray X1
2003	409.6	SGI Altix 3000
2003	256.0	IBM HPS
2004	364.8	Cray SeaStar (XT3)
2005	2400.0	YARC
2007	40.8	IBM BlueGene/P (Torus network)
2007	303.6	D.E. Shaw Anton
2007	364.8	Cray SeaStar2 (XT4)
2008	460.8	Cray SeaStar2+ (XT5)
2009	2880.0	Voltaire GridDirector 4036
2010	1280.0	Cray Gemini (XE6)
2010	7680.0	Tianhe-1A
2010	51800.0	Voltaire GridDirector 4700
2012	240.0	IBM BlueGene/Q (5-D torus)

**Table 6: Network router and link bandwidth.**



**Figure 9: Network bandwidth.** For methodology, see Table 6.



**Figure 10: Intra-node PC I/O bus bandwidth.** For methodology, see Table 7.

includes data since 1981 and taken from both Wikipedia and industry committee press releases.

Year	Gbps.Type	
1981	0.03816	ISA-PC-8 (4.77 MHz)
1983	0.064	ISA-XT-8 (8 MHz)
1984	0.128	ISA-AT-16 (8 MHz)
1987	0.16	MCA-16 (10 MHz)
1988	0.26656	EISA-32 (8.33 MHz)
1993	1.056	PCI-32 (33 MHz)
1995	2.112	PCI-32 (66 MHz)
1997	2.128	AGP-1.0
1997	4.264	AGP-1.0-2x
1997	17.064	AGP-32
1998	8.528	AGP-2.0
1998	8.512	PCI-X-64 (133 MHz)
2002	17.064	AGP-3.0
2003	17.024	PCI-X-2.0-64 (266 MHz)
2003	34.112	PCI-X-2.0-64 (533 MHz)
2004	32.0	PCIe-1.0a (x16)
2007	64.0	PCIe-2.0 (x16)
2010	128.0	PCIe-3.0 (x16)

**Table 7: Intra-node PC I/O bus bandwidth.**