FFT on XMT: Case Study of a Bandwidth-Intensive Regular Algorithm on a Highly-Parallel Many Core

James Edwards and Uzi Vishkin
University of Maryland Institute for Advanced Computer Studies (UMIACS)
University of Maryland
College Park, Maryland, USA
Email: {jedward5, vishkin}@umd.edu

Abstract—**FFT has been a classic computation engine for numerous applications. The bandwidth-intensive nature of FFT capped its performance on off-the-shelf parallel machines that are bandwidth-limited, and forced application researchers into seeking easier-to-speedup alternatives to FFT, even when inferior to FFT. But, what if effective support of FFT is feasible? Using FFT as an example, we examine the impact that adoption of some enabling technologies, including silicon photonics, would have on the performance of a many-core architecture. The results show that a single-chip many-core processor could potentially outperform a large high-performance computing cluster.**

Index Terms—**data movement; Fast Fourier Transform (FFT); many-core; PRAM**

I. Introduction

The parallel computing community has increasingly shifted its attention to communication avoidance as a way to address the end of Dennard scaling and the attendant difficulty in scaling down power consumption: see for example the National Academies report [1], work by Jim Demmel's group [2] on communication avoidance upper and lower bounds and many of the recent books in the computer architecture series by publisher Morgan and Claypool such as [3]. However, there are limits to the performance improvements that can be attained by focusing on reducing data movement. The strength of current parallel architectures lies in solving problems, such as dense-matrix multiplication, that can be solved by algorithms that are regular and require limited communication. For other algorithms, which are irregular or require high bandwidth, these platforms have been able to demonstrate limited speedups. Furthermore, the challenges of communication avoidance have arguably harmed programmers' productivity [4].

Still, the default mode for parallel programming research is reliance on off-the-shelf hardware. But what if alternative machines, or hardware features, are feasible, and can offer significant advantages? Clearly, such out-of-the-box hardware and the enabling technologies it may require are unlikely to ever be developed before their advantages are sufficiently understood. In contrast to work that seeks to avoid data movement, the current work examines the problem from an alternate angle: assuming that is it possible to reduce the energy cost of data movement, is it possible to obtain strong speedups on problems for which such speedups have proven elusive?

This question has been partially answered in the affirmative by prior work on the Explicit Multi-Threading (XMT[1]) general-purpose architecture [5], which aims to improve single-task completion time and ease-of-programming for parallel applications by supporting Parallel Random Access Model (PRAM) programming [6], [7]. Such work, discussed in Section III (in part, by way of reference to [8]), has focused largely on speedups for highly irregular parallel algorithms. Here, we begin to examine another class of algorithms that also appear too challenging for current platforms, namely, those that are regular but communication intensive.

Specifically, we examine one such algorithm, the fast Fourier transform (FFT). The FFT is an important numerical algorithm used in fields such as signal processing and scientific computing. What sets the FFT apart from other regular numerical algorithms is its high communication needs; given $O(S)$ local memory, it requires $O(n \log n/ \log S)$ I/O operations [9], which suggests that caches are of limited use in reducing the bandwidth required by the FFT. Indeed, prior work using existing platforms obtained modest speedups relative to the hardware invested; see Section I-A for some speedup examples, and for a comparison of prior speedups and hardware invested in them with the results (speedups and assumed hardware) of the current paper, see Section VI-A and Table VI.

Companion work on XMT [10] investigates the use of enabling technologies including 3D VLSI and microfluidic cooling to increase communication bandwidth on chip to shared cache, concluding that these technologies indeed enable XMT to scale up to 8x larger than would be possible without them. It also briefly considers the potential of photonics to extend this improvement off chip to greatly increase bandwidth to DRAM. The companion work uses XMT as a vehicle for performing a quantitative feasibility analysis of the enabling technologies in terms of temperature and power.

The intellectual merit of the work presented here is to complement the foregoing feasibility analysis with a quantitative analysis of the corresponding performance benefit, again using XMT as a vehicle. The two combined lay the groundwork for future analysis of the enabling technologies.

[1]XMT at the University of Maryland, not to be confused with the code name Cray XMT used during 2007-2011

In particular, the purpose of this work is to resolve the chicken-and-egg problem posed by enabling technologies: development of enabling technologies will not advance without evidence of their benefit, while such evidence apparently cannot be obtained until these technologies have already been developed. In order to resolve this impasse, we obtain preliminary results using a simulator (Section III-A), which does not require actual hardware to already exist. We recognize that the validity of these results is limited and will only reach the level of those for existing systems once we are able to obtain results on actual hardware.

While it is expected that increased bandwidth enabled by such technologies would lead to improved performance the (high) rate of improvement shows great promise. Of particular interest is the potential benefit of silicon photonics. Development of photonic technologies advanced enough to enable the largest systems considered herein would require non-trivial engineering effort. Although photonics is a topic of current interest, even the most recent progress has been modest in scale (e.g., [11]). In order to motivate more ambitious effort, we demonstrate that photonics could enable a single-chip many-core processor to outperform a much larger high-performance computing (HPC) cluster of nodes interconnected via traditional optics. This is especially true when the application, such as FFT, greatly underutilizes the peak computational capacity of the HPC system due to limited inter-node bandwidth. Here, XMT is a natural fit as the high off-chip bandwidth is coupled with matching on-chip bandwidth, permitting significantly higher utilization of available computational resources.

Our analysis consists of two parts. First, we measure the speedup of FFT on XMT relative to existing platforms. Specifically, we compare against FFTW [12], a highly-optimized implementation of the FFT, running on a single core of a modern Intel processor and also on multiple cores. In addition, we compare against a tuned FFT implementation running on Edison, a large HPC cluster. Second, we use the Roofline [13] model to evaluate how close our FFT implementation comes to achieving the peak performance possible on selected configurations of XMT and how performance may be further improved.

A. Comparison to prior work on the FFT

GPGPU Researchers at Microsoft [14] demonstrated performance of up to 300 GFLOPS on the NVIDIA GTX 280, with speedups of 2-4X over NVIDIA's cuFFT and 8-40X over Intel's MKL. The best result for a 2D FFT was around 120 GFLOPS, achieved with an input size of 1024 $\times$ 1024. No results are reported for 3D FFT.

Using a hybrid GPU-CPU algorithm, Chen and Li [15] achieved up to 43 GFLOPS for a 2D FFT and up to 27 GFLOPS for a 3D FFT on an NVIDIA Tesla C2075.

MPI Recent work [16] considers large 3D FFT on two high-end Cray systems using up to 32,768 cores. For an input of size 1024^3, the best result was 13,603 GFLOPS using 32,768 cores. Weak scaling results ranged from 159 GFLOPS for an input of size 512^3 to 17,611 GFLOPS for an input of size 4096 $\times$ 4096 $\times$ 2048.

Similar work on large MPI clusters [17] shows that a 3D FFT on an input of size 1024^3 can be computed in as little as 49 milliseconds (i.e., 3287 GFLOPS) using 16384 cores of an IBM-BlueGene/Q cluster.

Prior work on XMT Prior work on the FFT on XMT [18] did not consider 3D FFT and was limited to fixed-point arithmetic. Also, the prior work focused exclusively on FFT as an application rather than as a benchmark for evaluating the benefit of augmenting a computer architecture with enabling technologies.

II. Background

A. XMT Architecture

The Explicit Multi-Threading (XMT) general-purpose architecture [5] is a many-core architecture which aims to improve single-task completion time and ease-of-programming for parallel applications by supporting Parallel Random Access Model (PRAM) programming [6], [7]. For some advantages of XMT, see Section III.

The XMT processor includes a master thread control unit (MTCU); processing clusters, each comprising several light-weight thread-control units (TCUs); a high-bandwidth low-latency interconnection network; memory modules (MM), each comprising on-chip cache and off-chip memory; prefix-sum (PS) unit(s); and global registers. The shared-memory-modules block (bottom left of Fig. 1) suppresses the sharing of a memory controller by several MMs. The processor alternates between serial mode (in which only the MTCU is active) and parallel mode. The MTCU has a standard private data cache (used in serial mode) and a standard instruction cache. The TCUs, which lack a write data cache, share the MMs with the MTCU.

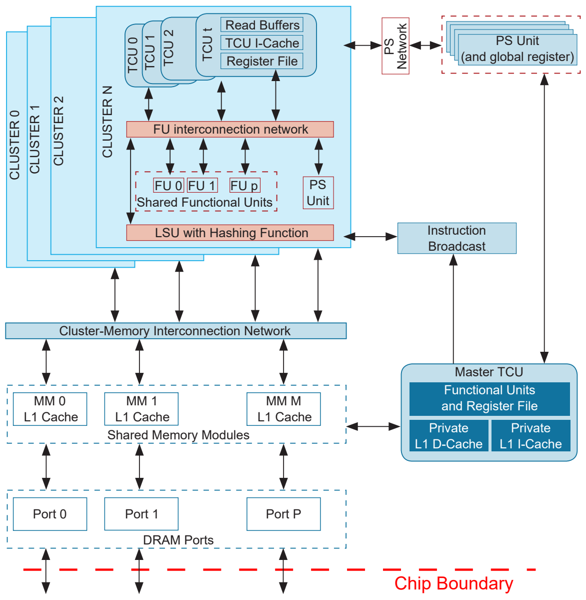

Fig. 1. Block Diagram of the XMT Architecture

The overall XMT design is guided by a general design ideal we call no-busy-wait finite-state-machines, or NBW FSM, meaning the FSMs, including processors, memories, functional units, and interconnection networks comprising the parallel machine, never cause one another to busy-wait. It is ideal because no parallel machine can operate that way. Nontrivial parallel processing demands the exchange of results among FSMs. The NBW FSM ideal represents our aspiration to minimize busy-waits among the various FSMs comprising a machine.

We cite the example of how the MTCU orchestrates the TCUs to demonstrate the NBW FSM ideal. The MTCU is an advanced serial microprocessor that also executes XMT instructions (such as spawn and join). Typical program execution flow can also be extended through nesting of sspawn commands. The MTCU uses the following XMT extension to the standard von Neumann apparatus of the program counters and stored program. Upon encountering a spawn command the MTCU broadcasts the instructions in the parallel section starting with that spawn command and ending with a join command on a bus connecting to all TCU clusters. The largest ID number of a thread the current spawn command must execute (Y) is also broadcast to all TCUs. The largest ID (index) of the executing threads is stored in a global register X. In parallel mode, a TCU executes one thread at a time. Executing a thread to completion (upon reaching a join command), the TCU does a prefix-sum using the PS unit to increment global register X. In response, the TCU gets the ID of the thread it could execute next; if the ID is $\leq$Y, the TCU executes a thread with this ID. Otherwise, the TCU reports to the MTCU that it finished executing. When all TCUs report they have finished, the MTCU continues in serial mode. The broadcast operation is essential to the XMT ability to start all TCUs at once in the same time it takes to start one TCU. The PS unit allows allocation of new threads to the TCUs that just became available within the same time as allocating one thread to one TCU. This dynamic allocation provides run-time load-balancing of threads coming from an XMTC program.

We are now ready to connect with the NBW FSM ideal. From the moment the MTCU starts executing a spawn command until each TCU terminates the threads allocated to it, no TCU can cause any other TCU to busy-wait for it. An unavoidable busy-wait ultimately occurs when a TCU terminates and begins waiting for the next spawn command.

TCUs, with their own local registers, are simple in-order pipelines, including fetch, decode, execute/memory-access, and write-back stages. A cluster includes functional units shared by several TCUs and one load/store port to the interconnection network shared by all its TCUs.

The global memory address space is evenly partitioned into the MMs through a form of hashing. The XMT design eliminates the cache-coherence problem, a challenge in terms of bandwidth and scalability. In principle, there are no local caches at the TCUs. Within each MM, the order of operations to the same memory location is preserved.

Quite a few performance enhancements have been incorporated into the XMT hardware, including compiler and run-time scheduling methods for nested parallelism and prefetching methods.

B. NoC (Network on Chip)

The high-throughput interconnection network required for the XMT architecture presents an implementation challenge. A unique data path can be provided for each pair of clusters and cache modules, such that there is no blocking in the network, using a mesh of trees (MoT) network. However, the number of switches required is proportional the product of the number of clusters and the number of cache modules, which translates to a large silicon area. For example, an XMT architecture in 22 nm technology with 8k TCUs requires silicon area of 190 mm^2 just for an MoT NoC. The area required for an MoT NoC of an XMT architecture with 16k TCUs is 760 mm^2, and would not fit on a single silicon layer. In order to reduce network area, a hybrid MoT and butterfly network can be used, where the inner levels of the "pure" MoT network are replaced with butterfly levels [19].

III. Motivation for using the XMT framework in this paper

Our choice to use XMT in this paper is motivated by several factors, described below.

A. Ease of experimentation

A practical reason for using XMT is the availability of XMTSim, a cycle-accurate simulator of the XMT architecture. XMTSim allows setting various architectural parameters such number of clusters, number of cache modules, and number of DRAM ports, which determines bandwidth to DRAM. This allows us to model the various configurations given in Section V. XMTSim and the XMTC compiler are described in [20] and have already been the basis for several publications including [21]. The most recent validation of the cycle-accuracy of the simulator is [22], which shows that the simulator cycle counts match those of the FPGA except in a minority of cases, where the discrepancy may be up to 33%, due in part to interconnect and DRAM technology limitations in the FPGA prototype that would not exist in an ASIC product. For the FFT, the difference due to these limitations is 5%.

B. Past XMT Speedups

For placing this debate in historical context, recall that claims that the main reason that parallel machines provide limited speedups is that the bandwidth between processors and memories is so limited are not new, as formally demonstrated in [23], [24].

PRAM is the main theory of parallel algorithms. A "proof-of-performance" with respect to PRAM algorithms demonstrated speedups between 1 and 2 orders of magnitude (up to 129X) on the most advanced parallel algorithms in the literature relative to the best known results on any machine (e.g., on GPUs) for any algorithms for the same problem. See Table I. Other published speedups include 20.4X on a

64-TCU XMT versus 4X on a 16-core AMD (using the same silicon area) for FFT [18] and 100X on a gate-level simulation benchmark suite [25].

C. Ease of programming

For brevity, we refer interested readers to Section 5 of [8] for an extensive discussion of results demonstrating ease of programming on the XMT platform.

IV. FFT ALGORITHM

A fast Fourier transform (FFT) is an efficient way to compute the discrete Fourier transform (DFT) of a vector of complex numbers. Given an N-point vector $\vec{x}$ as input, a naive implementation of the DFT amounts to multiplication of $\vec{x}$ by an $N \times N$ matrix containing Nth roots of unity. Formally, the DFT $\vec{X}$ of $\vec{x}$ is defined for $0 \leq k < N$ as

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N} \tag{1}$$

One way to derive an FFT from the basic DFT is to split the above summation into separate summations over even and odd indices:

$$X_k = \sum_{n=0}^{N/2-1} x_{2n} \cdot e^{-i2\pi k(2n)/N} + \sum_{n=0}^{N/2-1} x_{2n+1} \cdot e^{-i2\pi k(2n+1)/N} \tag{2}$$

If we define $\vec{x}^{\text{even}}$ and $\vec{x}^{\text{odd}}$ to be the even- and odd-indexed elements of $\vec{x}$, respectively, the above can be rewritten as

$$X_k = \sum_{n=0}^{N/2-1} x_n^{\text{even}} \cdot e^{-i2\pi kn/(N/2)} + e^{-i2\pi k/N} \sum_{n=0}^{N/2-1} x_n^{\text{odd}} \cdot e^{-i2\pi kn/(N/2)} \tag{3}$$

Each summation in Eq. (3) corresponds to Eq. (1) for a DFT of size $N/2$. Therefore, the DFT of $\vec{x}$ can be rewritten in terms of the DFTs of $\vec{x}^{\text{even}}$ and $\vec{x}^{\text{odd}}$. If we define $\omega_N = e^{i2\pi/N}$ to be the primitive Nth root of unity, the above can be rewritten as

$$X_k = X_k^{\text{even}} + \omega_N^{-k} X_k^{\text{odd}} \tag{4}$$

where terms of the form ω_N^{-k} are known as *twiddle factors*. See Fig. 2, noting that $\omega_N^{-(k+N/2)} = -\omega_N^{-k}$. Thus, an N-point DFT can be computed by separately computing two $N/2$-point DFTs and performing an additional $O(N)$ operations in $O(1)$ depth, as all the X_k can be computed in parallel. This decomposition can be applied recursively, leading to an $O(N \log N)$ work, $O(\log N)$ depth algorithm to compute the DFT; this divide-and-conquer algorithm is known as a radix-2 decimation-in-time Cooley-Tukey FFT.

Higher radixes Equation (1) can be split into any number $r \geq 2$ of interleaved summations provided r is a factor of N. The number r is known as the radix of the FFT.

Multidimensional FFT The FFT of a two-dimensional (2D) array is computed by separately computing the FFT of each row of the array and subsequently computing the FFT of each column of the array. Similarly, the FFT of a three-dimensional (3D) array is computed by taking the FFT along the first dimension, then the second, and finally the third.

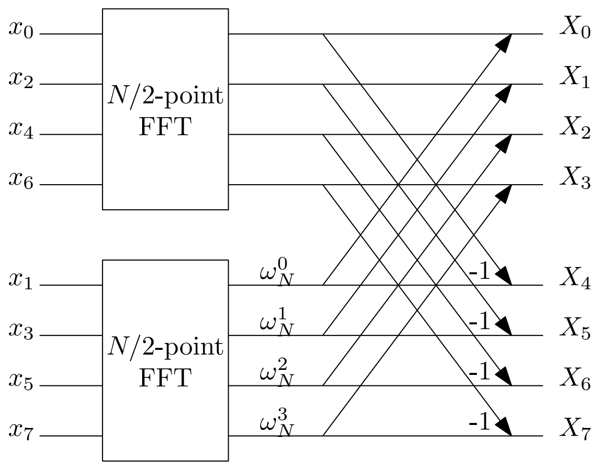

Fig. 2. Even/odd decomposition used by the Cooley-Tukey FFT

A. Implementation

We use a radix-8 Cooley-Tukey FFT. Here, we explain the design decisions we made in our FFT implementation.

Granularity of parallelism A multidimensional FFT can be parallelized using a coarse-grained approach or a fine-grained one. In a coarse-grained approach, one or more rows of the input are assigned to a thread, and the thread applies a serial FFT algorithm to its assigned row(s). In a fine-grained approach, multiple threads work on the FFT of a single row. Because the overhead for spawning threads on XMT is low, we choose a fine-grained approach to maximize the amount of available parallelism.

Depth-first versus breadth-first The recursion in the definition of the FFT may be resolved in one of two ways. If the recursive calls are made sequentially (the first completes before the second begins), then this results in a depth-first traversal of the recursion tree. On the other hand, making both calls in parallel results in a breadth-first traversal of the recursion tree. The choice of recursion order implies a tradeoff between locality and parallelism.

Using depth-first recursion, the size of the subproblem at the ith level of recursion is $N/2^i$. This means that the amount of cache used by the algorithm is less at deeper levels of the recursion. In fact, a cache-oblivious FFT algorithm uses this approach [29]. The drawback is that the degree of parallelism available decreases as well.

In contrast, the parallelism in the breadth-first approach can be flattened, resulting in an iterative implementation. This allows all subproblems at each level of the recursion to be solved simultaneously. The advantage is that the maximum amount of parallelism is always available; the disadvantage is that the working set is always as large as the entire problem. For inputs that are not too large, including those of a size examined herein, breadth-first is advantageous as it provides as

TABLE I
XMT SPEEDUPS

	Algorithm	XMT	GPU/CPU	Factor
	Graph Biconnectivity [8]	33X	4X, but only on random graphs	$\gg$8
	Graph Triconnectivity [26]	129X	Only serial result	129
	Max Flow [27]	108X	2.5X	43
Burrows Wheeler Transform - bzip2	Compression [28]	25X	X/2.5 on GPU	70
	Decompression [28]	13X	1.1X	11

much parallelism as possible for XMT to exploit, and XMT is designed to provide high bandwidth. For larger problem sizes, it may be advantageous to start with depth-first and switch to breadth-first when the subproblem becomes small enough.

Choice of Radix We implement a radix-r FFT by assigning r elements apiece to N/r threads. Each thread reads its r inputs from memory, solves the small FFT problem of size r for those inputs, and writes the result back to memory. The advantage of choosing a larger r is that fewer accesses to shared memory are required: $N \log_r N$ reads and the same number of writes. On the other hand, larger r also results in reduced parallelism. Another consideration is that larger values of r require more local storage for threads to store intermediate values. On XMT, each thread has access to 32 floating-point registers, which is enough to store 16 single-precision complex numbers. The largest practical r on XMT is 8, as some registers are required to store twiddle factors and intermediate values of computations. Choosing $r = 8$ provides sufficient parallelism for all but very small inputs: for an input size of 256^3, 2 million threads are available.

Twiddle Factors In principle, the twiddle factors can be computed on demand by computing the sine and cosine of the corresponding angle. However, these computations are relatively expensive. Since the twiddle factors depend only on the size of the input and not the input itself, the twiddle factors can be precomputed and stored in a lookup table. For 2D and higher-dimensional FFTs, the savings in computation can be significant, as all of the rows use the same set of twiddle factors. The downside to having all threads share the same lookup table is that it requires as many concurrent reads to each memory location as there are rows. Since accesses to the same memory location on XMT are queued, this results in a bottleneck. To alleviate this, we store multiple copies of the lookup table and spread accesses by threads uniformly across the copies. We choose the number of copies to be just enough so that one cache line in each cache module contains a portion of the lookup table. Using more copies would not provide any benefit as requests to the same cache module are queued; using fewer copies would mean that not all of the cache modules are utilized.

Decimation-in-time versus -frequency The formulation of the FFT presented here performs the recursive calls before doing the work of the current level of recursion, analogous to the way merging proceeds in merge sort; this formulation is referred to as decimation-in-time. It is possible to rearrange the computation so that the recursion occurs after the computation, analogous to quicksort; this version of the FFT is referred to as decimation-in-frequency.

A notable side effect of this choice is the way in which the roots of unity are used. Using decimation-in-time, roots of unity become increasingly fine-grained, starting with 2nd roots of unity for the smallest subproblems, followed by the 4th roots, 8th roots, and so on. This is reversed for decimation-in-frequency, which starts by using the Nth roots, followed by the $N/2$th roots for the next smaller subproblem, then the $N/4$th, and so on.

We chose to use decimation-in-frequency for our implementation because it more naturally fits the replication scheme we use for twiddle factors. In the first iteration, there are N Nth roots of unity, each of which will be accessed once. In the second iteration, there will be N/r N/rth roots of unity, which are a subset of the Nth roots of unity, each of which will be accessed r times. The remaining Nth roots of unity will be unused for the remaining iterations, so we replace them with replicas of the next lowest N/rth root of unity. We perform a similar procedure after each iteration, replacing unused roots of unity with replicas of roots that are still being used.

B. *Ease of programming*

Finally, we note that the tuning described above required only a modest effort beyond that required for a serial implementation of FFT. In particular, only the handling of twiddle factors required special handling in parallel, and the solution was a simple application of the logarithmic-time PRAM broadcast algorithm. In contrast, the MPI implementation of Song and Hollingsworth [16] requires domain decomposition of the data and further requires breaking the communication steps into multiple phases to allow pipelining.

V. EXPERIMENTAL CONFIGURATIONS

A goal of this paper is to examine the level of enabling technology needed to build various sizes of parallel systems and determine the opportunities that such systems provide to applications. To that end, we choose some configurations of XMT that represent what can be achieved with a given level of technology and explain what the barrier is to reaching the following level. For most configurations below, we consider a 2 cm by 2 cm chip (4 cm^2) using 22 nm technology, though the largest ones assume 14 nm technology. These configurations are summarized in Table II, and the required silicon area is given in in Table III.

TABLE II
XMT ARCHITECTURE CONFIGURATIONS

	4k	8k	64k	128k x2	128k x4
TCUs	4096	8192	65536	131072	131072
Clusters	128	256	2048	4096	4096
Memory Modules	128	256	2048	4096	4096
NoC MoT Levels	14	16	8	6	6
NoC Butterfly Levels	0	0	7	9	9
MMs per DRAM Ctrl.	8	8	8	4	1
FPUs per Cluster	1	1	1	2	4
TCUs per Cluster			32		
ALUs per Cluster			32		
MDUs per Cluster			1		
LSUs per Cluster			1		

TABLE III
XMT PHYSICAL CONFIGURATIONS

	4k	8k	64k	128k x2	128k x4
Technology Node (nm)	22	22	22	14	14
Silicon (Si) Layers	1	2	8	9	9
Si Area per Layer (mm^2)	227	276	380	365	393
Total Si Area (mm^2)	227	551	3046	3284	3540

A. Baseline: 4096 TCUs ("4k")

The smallest configuration we consider consists of 4096 TCUs. This is the largest system we can fit in a single silicon layer using 22 nm technology. This configuration is strictly smaller than the one in the next section and therefore does not require any enabling technologies.

B. 3D VLSI: 8192 TCUs ("8k")

To overcome the area limitation of the baseline configuration, we can split the XMT chip across multiple layers using 3D VLSI. Companion work [10] shows that an 8192-TCU configuration of XMT is feasible using air cooling alone, but not a larger one.

Another issue that arises at this point is off-chip bandwidth. The 32 DRAM channels of this configuration require a total of 6.76 Tb/s of off-chip bandwidth. Using a standard parallel memory interface such as DDR3, this would require about 4000 pins on the XMT processor package. This may already be infeasible, as even the NVIDIA Tesla K40 GPU (with 561 mm^2 of silicon area) only has 2397 pins, and this problem becomes more acute for larger XMT configurations that require more off-chip bandwidth. A high-speed serial interface would allow consolidating a DRAM channel into a few pins. For example, using the 32.75 Gb/s GTY transceivers on the Xilinx UltraScale+ line of FPGAs, a DRAM channel can be reduced to 7 pins. A configuration with 32 DRAM channels would then require just 224 pins.

C. Microfluidic cooling: 65536 TCUs ("64k")

A significant issue with 3D VLSI is that the middle layers of the stack are thermally insulated from the outside of the chip, and therefore cooling those layers is difficult. One possible solution for cooling the middle layers is microfluidic cooling (MFC), which uses a liquid (such as water) pumped through tiny channels between layers to remove heat. Companion work [10] shows that MFC is sufficient to cool even a 65536-TCU configuration of XMT. At this point, the number of layers in the 3D stack becomes a limiting factor. Off-chip bandwidth also becomes a limiting factor, as even with high-speed serial transceivers, the 256 DRAM channels of this configuration would require a total of 1792 pins.

D. Photonics and 14 nm node: 131072 TCUs ("128k x2")

For larger configurations of XMT, we need to look ahead to smaller technology nodes. For scaling from 22 nm to 14 nm, Intel claims a scaling factor of 0.54 for logic area and similar scaling for power consumption [30]. If we keep the area of the network-on-chip fixed, this allows us to double the number of clusters and memory modules with some area to spare. With the remaining area, we can add more FPUs. Because we double the off-chip bandwidth per memory module (see below), we choose to also double the number of FPUs per cluster to balance computation capability with communication.

In order to provide sufficient off-chip bandwidth for this configuration, we need to replace the copper interconnect with a more advanced one, such as an optical interconnect driven by silicon photonics. A significant issue with this solution is heat. Faster photonic transceivers tend to be less energy efficient than slower ones. For example, by combining eight 10-Gb/s channels in a single transceiver using wave division multiplexing, it is possible to achieve an efficiency of 600 fJ/bit and an I/O density of 700 Gbps/mm^2 [31]. For a 4 cm^2 chip, this solution provides 280 Tb/s of off-chip bandwidth using 168 W, which is enough to double the ratio of DRAM controllers to memory modules. More recent work achieves higher rates per channel but at the cost of an order of magnitude more power; two approaches using 30 Gb/s transceivers without multiplexing require approximately 3 pJ/bit [32] and 8 pJ/bit [33].

If the photonic transceivers are air cooled, then this limits their power dissipation and thus the bandwidth that can be achieved. In 2004, forced air cooling was predicted to achieve little more than 100 W/cm^2 [34, p. 4] to 150 W/cm^2 [35], and this projection has since remained steady [36]. This means that for a 4 cm^2 chip, air cooling can remove no more than 600 W of heat. In this case, the 10-Gb/s channels provide more bandwidth within the power budget than the 30-Gb/s ones.

Another limit at this point is the number of through silicon vias (TSVs) that connect to the network-on-chip (NoC). A practical limit to the number of TSVs on a single layer may be one hundred thousand [37], as beyond this point manufacturing cost quickly increases and total TSV footprint becomes a significant percentage of silicon area. The width of a NoC port is 50 bits; at 3.3 GHz, the required bandwidth is 165 Gb/s per port. Each TSV can operate at 40 Gb/s [38], [39], so five TSVs are required per port. A 131072-TCU configuration with 4096 clusters and 4096 cache modules will require 20480 TSVs for each of the following: from the NoC to processors, from processors to the NoC, from NoC to memory modules, and memory modules to NoC. This is a total of 81920 TSVs, which allows eighteen thousand TSVs for other purposes, namely power delivery. Assuming a TSV pitch of 12 μm [40], one hundred thousand TSVs will require 14.4 mm^2 of silicon area.

E. MFC-cooled photonics: more off-chip bandwidth ("128k x4")

Although silicon area limits the size of the XMT chip, there is still room for growth. Namely, the amount of off-chip bandwidth could be increased by applying microfluidic cooling to the photonic transceivers as well as the rest of the chip. This would allow using smaller, faster photonic transceivers, which would provide sufficient bandwidth to allow each memory module to have its own DRAM controller rather than sharing bandwidth with other memory modules. We also increase the number of FPUs to four per cluster; beyond this number, we observe diminishing returns.

Another possible application of MFC-cooled photonics is to split the XMT floorplan across multiple chips at the interface between clusters (and/or memory modules) and the network-on-chip. This would allow for reducing the height of the 3D VLSI stack on each chip without reducing the system size. With sufficient off-chip bandwidth, it would even be possible to split the network-on-chip across multiple chips. It is up to future technology development to indicate which approach works better.

VI. Results

We use XMTSim to obtain cycle counts for computing a single-precision, complex 3D FFT with an input of size $512 \times 512 \times 512$. We assume that the clock speed of XMT is the same as that of the Intel processor used as the reference for our speedup figures, namely 3.3 GHz. To allow comparison with other work on the FFT (e.g., [16]), we report FLOPS based on the standard rule of $5N \log_2 N$ floating-point operations for an FFT of N elements. An exception to this is Section VI-B, as the Roofline model defines FLOPS to be the actual number of floating-point operations (as reported by XMTSim) per second.

A. Comparison to FFTW

Serial FFTW We evaluate the performance of our implementation of FFT on XMT for the configurations given in Table II by comparing it to an existing highly-optimized implementation of FFT, namely FFTW version 3.3.4. The baseline for our speedups is serial FFTW running on one core of an 8-core Intel Xeon E5-2690 with 20 MB of cache. Performance in GFLOPS is in Table IV, and speedup results are in Table V.

TABLE IV
FFT Performance on XMT

Configuration	4k	8k	64k	128k x2	128k x4
GFLOPS	239	500	3667	12570	18972

Parallel FFTW The E5-2690 uses 416 mm^2 of silicon area in 32 nm technology. If we assume an ideal scaling to 22 nm, then the E5-2690 would use about 197 mm^2 in 22 nm technology. This implies that the 4k configuration of XMT would use about 1.15 times as much silicon as an E5-2690. We ran parallel FFTW on a system consisting of two E5-2690 processors, which supports up to 32 threads (16 cores with hyper-threading). Notably, the 4k configuration achieves a 2.8X speedup relative to this system while using only 58% of its silicon area.

TABLE V
Speedups relative to FFTW

Configuration	4k	8k	64k	128k x2	128k x4
Speedup vs. serial	31X	66X	482X	1652X	2494X
Speedup vs. 32 threads	2.8X	5.8X	43X	147X	222X

B. Evaluation using Roofline model

Speedup results provide useful information, but limited insight. In particular, they do not establish that the problem cannot be solved more quickly, even on the same platform. Because the FFT is regular, its performance can be analyzed by comparison with the peak performance that the platform is capable of.

The Roofline model [13] describes a platform in terms of two parameters: peak computation rate and peak off-chip bandwidth. Peak computation rate is often (but not necessarily) measured in terms of floating-point operations per second (FLOPS), while bandwidth is measured in bytes per second. These two parameters are plotted on a graph whose y-axis is FLOPS and whose x-axis is computational intensity, the ratio of computation to data movement (measured in FLOPs/byte). Algorithms with low computational intensity are data bound; such algorithms fall under the sloped portion of the graph. Algorithms with high computation intensity are compute bound; these fall under the horizontal portion of the graph. Based on a constant-factor analysis of the number of operations performed by the FFT and its I/O complexity, an upper bound for the computational intensity of the FFT is $\log S$ FLOPS/word [41], where S is the size of the last-level cache in words; for single-precision floating-point numbers, this is $0.25 \log S$ FLOPS/byte.

Our multidimensional FFT implementation consists of two phases that are executed once per dimension. First, the FFT of each row is computed. Second, the axes of the array are rotated[2] so that the next time the FFT is applied to the rows of the array, it will actually compute the FFT of what was originally the columns of the array. In our implementation, the rotation is combined with the last iteration of the computation to reduce the number of synchronization points and round trips to memory.

In Fig. 3, we show how the observed performance of the overall FFT computation and its two phases compares to the theoretical Roofline model of the tested configurations of XMT. The rotation phases are communication intensive

[2]In the special case of a 2D array, rotation is equivalent to a matrix transpose.

and thus fall to the left of the non-rotation phases, which are more computation intensive. The overall performance of the algorithm is equal to the weighted average of the two phases with respect to the time each cycle takes, so the overall performance falls on the line connecting the two phases. The overall performance is closer to the non-rotation phase since the non-rotation phase takes the majority of the time.

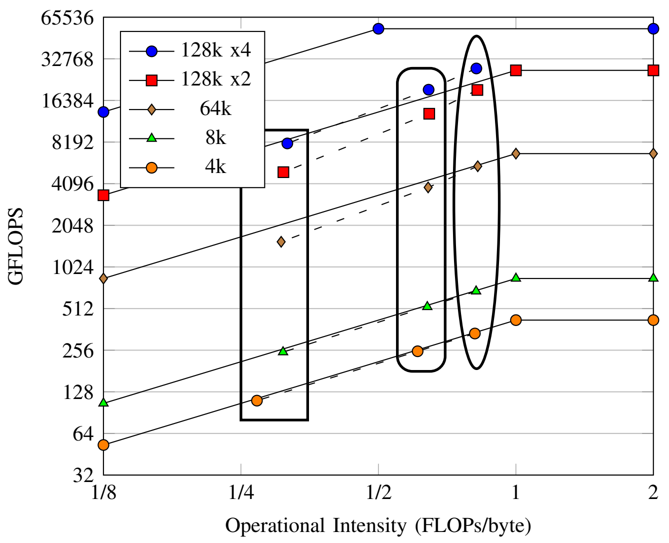

Fig. 3. Roofline model of each XMT configuration (solid line with markers) with empirical results for 3D FFT (markers on dashed line). On each dashed line, the marker on the left (inside rectangle) corresponds to iterations where rotation is performed, the marker on the right (inside ellipse) corresponds to iterations where no rotation is performed, and the marker in the middle (inside rounded rectangle) is for the overall FFT algorithm.

We make the following observations about the results:

(a) In the 4k and 8k configurations, both phases are essentially on the sloped line, indicating that they operate very close to the peak off-chip bandwidth.

(b) In the 64k configuration, the rotation step is beginning to fall below the sloped line. Since virtual parallelism is not lacking, this must be due to the decreased number of mesh-of-trees levels in the interconnection network (ICN), a result of the constraint on interconnection network area. This effect is more pronounced in the 128k x2 configuration, which has even fewer mesh-of-trees levels.

(c) The 128k x4 configuration provides only a 51% improvement over the 128k x2 configuration. As in (b), this is because the ICN is the bottleneck, and increasing the bandwidth to DRAM beyond that of the 128k x2 configuration does little to reduce congestion in the ICN.

Future technology scaling should allow for a more dense network-on-chip, which would alleviate the bottleneck and allow for an even larger configuration of XMT.

C. Comparison to Edison

Edison is a Cray XC30 machine consisting of numerous 12-core Intel Xeon E5-2695v2 processors interconnected using a Cray Aries network with a Dragonfly topology. Edison is an enormous machine while even the largest configuration of XMT we consider here is of a much more modest size, as shown in Table VI. For example, in order to facilitate comparison of the silicon area required by the two systems, following the row that compares total actual silicon areas and the VLSI process used, we present areas normalized to 22 nm technology.

TABLE VI
COMPARISON OF EDISON MACHINE (CRAY XC30) TO XMT

	Edison	XMT (128k x4)
# processing elements	124,608 cores	131,072 TCUs
# processor groups	5,192 nodes	4,096 clusters
Total cache memory	311,520 MB	128 MB
# chips	10,384 CPU + 1,298 router	1
Total silicon area (process)	56,177 cm^2 (22 nm) + 4,072 cm^2 (40 nm)	35.4 cm^2 (14 nm)
Normalized silicon area (22 nm)	57,409 cm^2	66 cm^2
Peak power consumption	2,500 KW	7.0 KW
Peak teraFLOPS	2,390	54
TeraFLOPS for FFT (size)	13.6 (1024^3)	19.0 (512^3)
% of peak FLOPS	0.57%	35%

The 128k x4 XMT system achieves a 1.4X higher speedup than Edison even though the latter requires 870 times the silicon area and 375 times the power of the XMT system. To put the power consumption of the XMT chip into perspective, microfluidic cooling can remove nearly 1 KW/cm^2 of heat per layer; see [42] and [43] for examples of single layer microfluidic cooling prototypes that have removed 790 W/cm^2 and 681 W/cm^2 respectively.

VII. CONCLUSION

We have shown the potential for significant speedups relative to off-the-shelf platforms on the FFT, an important mathematical algorithm. In contrast, without co-design of algorithms and architectures, strong speedups have been elusive. This suggests that it is indeed worth investing further effort into development of a cohort of enabling technologies including silicon photonics for affording higher bandwidth.

REFERENCES

[1] L. I. Millett, S. H. Fuller *et al.*, *The Future of Computing Performance: Game Over or Next Level?* National Academies Press, 2011.

[2] J. Demmel, "Communication-avoiding algorithms for linear algebra and beyond," in *IEEE 27th International Symposium on Parallel Distributed Processing (IPDPS)*, May 2013, pp. 585–585.

[3] M. Sjlander, M. Martonosi, and S. Kaxiras, "Power-efficient computer architectures: Recent advances," *Synthesis Lectures on Computer Architecture*, vol. 9, no. 3, pp. 1–96, 2014.

[4] U. Vishkin, "Is multicore hardware for general-purpose parallel processing broken?" *Commun. ACM*, vol. 57, no. 4, pp. 35–39, Apr. 2014.

[5] X. Wen and U. Vishkin, "FPGA-based prototype of a PRAM-on-chip processor," in *Proceedings of the 5th conference on Computing frontiers*, 2008, pp. 55–66.

[6] J. JáJá, *An introduction to parallel algorithms.* Addison-Wesley Reading, 1992, vol. 17.

[7] J. Keller, C. Kessler, and J. Träff, *Practical PRAM programming*. Wiley-Interscience, J. Wiley & Sons, Inc., 2001.

[8] J. A. Edwards and U. Vishkin, "Better speedups using simpler parallel programming for graph connectivity and biconnectivity," in *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores - PMAM '12*. New York, New York, USA: ACM Press, 2012, pp. 103–114.

[9] J.-W. Hong and H. T. Kung, "I/O complexity: The red-blue pebble game," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, 1981, pp. 326–333.

[10] S. O'Brien, U. Vishkin, J. Edwards, E. Waks, and B. Yang, "Can cooling technology save many-core parallel programming from its programming woes?" in *Compiler, Architecture and Tools Conference (CATC)*, Intel Development Center, Haifa, Israel, November 23, 2015, or http://drum.lib.umd.edu/handle/1903/17153.

[11] C. Sun *et al.*, "Single-chip microprocessor that communicates directly using light," *Nature*, vol. 528, no. 7583, pp. 534–538, 2015.

[12] M. Frigo and S. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, Feb. 2005.

[13] S. W. Williams, A. Waterman, and D. A. Patterson, "Roofline: An insightful visual performance model for floating-point programs and multicore architectures," University of California at Berkeley, Tech. Rep., 2008. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-134.html

[14] N. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli, "High performance discrete Fourier transforms on graphics processors," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, November 2008.

[15] S. Chen and X. Li, "A hybrid GPU/CPU FFT library for large FFT problems," in *32nd International Performance Computing and Communications Conference (IPCCC)*. IEEE, Dec 2013, pp. 1–10.

[16] S. Song and J. K. Hollingsworth, "Scaling Parallel 3-D FFT with Non-Blocking MPI Collectives," in *5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, 2014, pp. 1–8.

[17] V. Nikl and J. Jaros, "Parallelisation of the 3D fast Fourier transform using the hybrid OpenMP/MPI decomposition," in *9th International Doctoral Workshop, MEMICS 2014, Revised Selected Papers*, 2014, pp. 100–112.

[18] A. B. Saybasili, A. Tzannes, B. R. Brooks, and U. Vishkin, "Highly parallel multi-dimensional fast Fourier transform on fine- and coarse-grained many-core approaches," in *Proc. Parallel and Distributed Computing and Systems*, 2009.

[19] A. O. Balkan, G. Qu, and U. Vishkin, "An area-efficient high-throughput hybrid interconnection network for single-chip parallel processing," in *Proceedings of the 45th annual Design Automation Conference*, 2008, pp. 435–440.

[20] F. Keceli, A. Tzannes, G. Caragea, R. Barua, and U. Vishkin, "Toolchain for programming, simulating and studying the XMT many-core architecture," in *Proc. IPDPSW*, 2011, pp. 1282–1291.

[21] G. C. Caragea, F. Keceli, A. Tzannes, and U. Vishkin, "General-purpose vs. GPU: Comparison of many-cores on irregular workloads," in *HotPar '10: Proceedings of the 2nd Workshop on Hot Topics in Parallelism*. USENIX, Jun. 2010.

[22] F. Keceli, "Power and performance studies of the explicit multi-threading (XMT) architecture," Ph.D. dissertation, University of Maryland, 2011, chapter 4. http://hdl.handle.net/1903/11926.

[23] Y. Mansour, N. Nisan, and U. Vishkin, "Trade-offs between communication throughput and parallel time," in *Proc. 26th Annual ACM Symp. on Theory of Computing*, 1994, pp. 372–381.

[24] U. Vishkin and A. Wigderson, "Trade-offs between depth and width in parallel computation," *SIAM J. Comput.*, vol. 14, no. 2, pp. 303–314, 1985.

[25] P. Gu and U. Vishkin, "Case study of gate-level logic simulation on an extremely fine-grained chip multiprocessor," *Journal of Embedded Computing*, vol. 2, no. 2, pp. 181–190, 2006.

[26] J. A. Edwards and U. Vishkin, "Brief announcement: Speedups for parallel graph triconnectivity," in *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architecture*, 2012, pp. 190–192.

[27] G. C. Caragea and U. Vishkin, "Brief Announcement: Better Speedups for Parallel Max-flow," in *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2011, pp. 131–134.

[28] J. A. Edwards and U. Vishkin, "Empirical speedup study of truly parallel data compression," University of Maryland, Tech. Rep., 2013. [Online]. Available: http://hdl.handle.net/1903/13890

[29] M. Frigo, C. Leiserson, H. Prokop, and S. Ramachandran, "Cache-oblivious algorithms," in *40th Annual Symposium on Foundations of Computer Science*, 1999.

[30] R. Borkar, M. Bohr, and S. Jourdan, "Advancing Moore's Law on 2014," Intel, Aug. 2014. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/presentation/advancing-moores-law-in-2014-presentation.pdf

[31] X. Zheng, F. Liu, J. Lexau, D. Patil, G. Li, Y. Luo, H. D. Thacker, I. Shubin, J. Yao, K. Raj, R. Ho, J. E. Cunningham, and A. V. Krishnamoorthy, "Ultralow power 80 Gb/s arrayed CMOS silicon photonic transceivers for WDM optical links," *Journal of Lightwave Technology*, vol. 30, no. 4, pp. 641–650, 2012.

[32] N. Dupuis, B. G. Lee, J. E. Proesel, A. Rylyakov, R. Rimolo-Donadio, C. W. Baks, A. Ardey, C. L. Schow, A. Ramaswamy, J. E. Roth, R. S. Guzzon, B. Koch, D. K. Sparacin, and G. A. Fish, "30-Gb/s optical link combining heterogeneously integrated III-V/Si photonics with 32-nm CMOS circuits," *Journal of Lightwave Technology*, vol. 33, no. 3, pp. 657–662, 2015.

[33] J. Joo, K.-S. Jang, S. H. Kim, I. G. Kim, J. H. Oh, S. A. Kim, G.-S. Jeong, Y. Kim, J.-E. Park, S. Kim, H. Chi, D.-K. Jeong, and G. Kim, "Silicon photonic receiver and transmitter operating up to 36 Gb/s for λ˜1550 nm," *Optics Express*, vol. 23, no. 9, p. 12232, 2015.

[34] L. Zhang, K. E. Goodson, and T. W. Kenny, *Silicon Microchannel Heat Sinks: Theories and Phenomena*. Springer Berlin Heidelberg, 2004.

[35] G. Upadhya, J. Hom, K. Goodson, and M. Munch, "Electro-kinetic microchannel cooling system for servers," in *The Ninth Intersociety Conference on Thermal and Thermomechanical Phenomena In Electronic Systems*, 2004, pp. 367–371.

[36] M. R. Stan, S. Gurumurthi, R. J. Ribando, and K. Skadron, "Interaction of scaling trends in processor architecture and cooling," *2010 26th Annual IEEE Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*, pp. 198–204, 2010.

[37] D. Velenis, M. Stucchi, E. J. Marinissen, B. Swinnen, and E. Beyne, "Impact of 3D design choices on manufacturing cost," in *IEEE International Conference on 3D System Integration (3DIC)*, 2009.

[38] X. Sun, G. Van der Plas, M. Detalle, and E. Beyne, "Analysis of 3D interconnect performance: Effect of the Si substrate resistivity," in *IEEE International Conference on 3D Systems Integration (3DIC)*, 2014.

[39] R. Weerasekera, M. Grange, D. Pamunuwa, and H. Tenhunen, "On signalling over through-silicon via (TSV) interconnects in 3-D integrated circuits," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2010.

[40] M. A. Rabie, P. C. S., R. Ranjan, M. I. Natarajan, S. F. Yap, D. Smith, S. Thangaraju, R. Alapati, and F. Benistant, "Novel stress-free keep out zone process development for via middle TSV in 20nm planar CMOS technology," in *IEEE International Interconnect Technology Conference*, 2014, pp. 203–206.

[41] V. Elango, N. Sedaghati, F. Rastello, L.-N. Pouchet, J. Ramanujam, R. Teodorescu, and P. Sadayappan, "On using the Roofline model with lower bounds on data movement," *ACM Transactions on Architecture and Code Optimization*, vol. 11, no. 4, pp. 1–23, 2015.

[42] D. B. Tuckerman and R. Pease, "High-performance heat sinking for VLSI," *Electron Device Letters, IEEE*, vol. 2, no. 5, pp. 126–129, 1981.

[43] T. Brunschwiler, B. Michel, H. Rothuizen, U. Kloter, B. Wunderle, H. Oppermann, and H. Reichl, "Forced convective interlayer cooling in vertically integrated packages," in *IEEE 11th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITHERM)*, 2008, pp. 1114–1125.