Power-Performance Comparison of Single-Task Driven Many-Cores

Fuat Keceli *, Tali Moreshet †, and Uzi Vishkin *
* Department of Electrical and Computer Engineering, University of Maryland, keceli,vishkin@umd.edu
† Department of Engineering, Swarthmore College, tali@swarthmore.edu

***Abstract*—Many-cores, processors with 100s of cores, are becoming increasingly popular in general-purpose computing, yet power is a limiting factor in their performance. In this paper, we compare the power and performance of two design points in the many-core processor domain. The XMT[1] general-purpose processor provides significant runtime advantage on irregular parallel programs (e.g., graph algorithms). This was previously demonstrated and tied to its architecture choices and ease-of-programming. In contrast, current commercial GPUs excel at regular parallel programs that require high processing capability. In this work, we set the power envelope as a constraint and evaluate an envisioned 1024-core XMT processor against an NVIDIA GTX280 GPU considering various scenarios for estimating the power of the XMT chip. Even under worst-case assumptions and scenarios, simulations show that the XMT processor sustains its advantage over the GPU on irregular parallel programs, while not falling significantly behind on regular programs. The total energy spent per benchmark fits a similar pattern. Given that the two architectures target different types of parallelism, a future system can potentially utilize an XMT chip and a GPU chip in complementary roles.**

***Index Terms*—XMT, GPU, parallelism, power and performance comparison, many-core, PRAM**

I. Introduction

The power wall forced the transition from single to multi-core microprocessors in general purpose computing [5]. Nevertheless, power and thermal feasibility remains a first class design constraint for today's computing systems, whereas the introduction of parallelism to the mainstream general-purpose domain brings another long elusive problem to focus: ease of parallel programming. Consequently, the success of a many-core system depends on its competitiveness on both power dissipation and programmability.

The eXplicit Multi-Threading (XMT) architecture is a general-purpose many-core platform for fine grained parallel programs that scales to hundreds, or even thousands of lightweight cores. XMT aims at: *(i)* improving single task execution time through parallelism, and at the same time providing competitive performance on serial code (backwards compatibility), and *(ii)* making the task for the parallel programmer as easy as possible [37]. The current XMT platform consists of a proof-of-concept 64-core FPGA and ASIC prototypes [41], [42] and a highly configurable cycle-accurate simulator (XMTSim) [20]. A 1024 core XMT chip (XMT1024) was estimated to use the same silicon area as an implementation of the NVIDIA Tesla architecture, the GTX280 GPU [7]. It was also reported that XMT1024 outperforms GTX280 for programs with irregular memory access and/or parallelism patterns that commonly exist in the general-purpose domain [7]. The purpose of this work is to show that the speedups remain significant when a power envelope, obtained from GTX280, is an added constraint. For many-cores, the power envelope is a suitable metric [4] that is closely related to feasibility and cooling costs of individual chips or large systems consisting of many processors.

Early estimates in a simulation based architecture study are always prone to errors due to possible deviations in the parameters used in the power model. Therefore we consider various scenarios that represent potential errors in the model. We show that for the best case scenario XMT1024 overperforms GTX280 by an average of 8.8x on irregular benchmarks and 6.4x overall. Speedups are only reduced by an average of 20% for the average-case scenario and approximately halved for the worst-case. We also compare the energy consumption per benchmark on both chips, which follows the same trend as the speedups.

Irregular parallel programs are among the hardest problems in parallel computing today (see $Benchmarks$ in Sec. IV). These programs defy optimizations, such as programming for locality, that are common in typical many-cores, and require significant programming effort to obtain minor performance improvements. Previous work, however, documents a strong advantage of XMT on ease-of-programming [13], [30], [36], [40] and effectiveness for achieving speedups, even on such workloads [7], [8]. Establishing that power constraints are not likely to dampen the performance of XMT leads to the conclusion that XMT is an attractive option for future computer systems, especially when coupled with a GPU chip for more GPU-specific computations. Even as a standalone general purpose processor, XMT still performs well on all types of parallel programs and provides the valuable asset of ease-of-programming.

II. Many-Core Platforms: XMT and Tesla

This section provides an overview of the XMT and the Tesla architectures. Table I highlights some of the key issues that affect the design of both architectures and lists the main differences between them. These differences motivate the power and performance study that we will present in the subsequent sections. Not reflected in the table are design decisions that are shared by the two designs, such as using lightweight cores and not including coherent per-core caches. In this paper, we extend the comparison between XMT and

[1]This refers to the XMT architecture developed at the University of Maryland and not the Cray XMT system.

	Tesla	XMT
Memory Latency Hiding and Reduction	·Heavy multithreading (requires large register files and state aware scheduler). ·Limited local shared scratchpad memory. ·No coherent private caches at SM or SP.	·Large globally shared cache. ·No coherent private TCU or cluster caches. ·Software prefetching.
Memory and Cache Bandwidth	·Memory access patterns need to be coordinated by the user for efficiency (request coalescing). ·Scratchpad memories prone to bank conflicts. ·High bandwidth interconnection network.	·Caches relax the need for user-coordinated DRAM access. ·Address hashing for avoiding memory module hotspots. ·Mesh-of-trees interconnect able to handle irregular communication efficiently.
Functional Unit (FU) Allocation	·Dedicated FUs for SPs and SFUs. ·Less arbitration logic required. ·Higher theoretical peak performance.	·Heavyweight FUs (FPU/MDU) are shared through arbitrators. ·Lightweight FUs (ALU and branch unit) are allocated per TCU (ALUs do not include multiply/divide functionality).
Control Flow and Synchronization	·Single instruction cache and issue per SM for saving resources. Warps execute in lock-step (penalizes diverging branches). ·Efficient local synchronization and communication within blocks. Global communication is expensive. ·Switching between serial and parallel modes (i.e. passing control from CPU to GPU) requires off-chip communication.	·One instruction cache and program counter per TCU enables independent progression of threads. ·Coordination of threads can be performed in constant time via prefix-sum. Other forms of thread communication are done over the shared cache. ·Dynamic hardware support for fast switch between serial and parallel modes and load balance of virtual threads.

TABLE I: Implementation differences between XMT and Tesla. FPU and MDU stand for floating-point and multiply/divide units respectively.

Tesla architectures presented in [7]. Extending our study to Tesla's successor, the Fermi architecture [28] is reserved for future work.

The XMT Architecture. The primary goal of the eXplicit Multi-Threading (XMT) general-purpose computer architecture [27], [39] has been improving single-task performance through parallelism. XMT was designed from the ground up to capitalize on the huge on-chip resources becoming available in order to support the formidable body of knowledge, known as Parallel Random Access Model (PRAM) algorithmics [18], [21], and the latent, though not widespread, familiarity with it. Ease-of-programing is one of the main objectives of XMT: considerable amount of evidence was developed on *ease of teaching* [36], [40] and improved *development time* with XMT relative to alternative parallel approaches including MPI [13], OpenMP [30] and CUDA (experiences in [7]).

The XMT architecture, depicted in Fig. 1.a, includes an array of lightweight cores, Thread Control Units (TCUs), and a serial core with its own cache (Master TCU). The architecture includes several clusters of TCUs connected by a high-throughput interconnection network, for example using a mesh-of-trees (MOT) topology [2]; an instruction and data broadcast mechanism; a global register file; and a prefix-sum unit (similar to Fetch-and-Add [11], provides constant, low overhead inter-thread coordination). The first level of cache is shared and partitioned into mutually-exclusive cache modules sharing several off-chip DRAM memory channels. The TCU Load-Store unit applies hashing on each address to avoid memory hotspots. Cache modules handle concurrent requests, which are buffered and reordered to achieve better DRAM bandwidth utilization. Within a cluster, a compiler-managed read-only (constant) cache is used to store constant values across all threads. TCUs include lightweight ALUs, but the more expensive Multiply/Divide (MDU) and Floating Point Units (FPU) are shared by all TCUs in a cluster. TCUs also feature prefetch buffers which are utilized via a compiler optimization to hide memory latencies.

XMT is programmed in XMTC, a simple extension of the C language which contains succession of serial and parallel code sections. The code of a parallel section is expressed in the SPMD (single program, multiple data) style, specifying an arbitrary number of virtual threads sharing the same code. An algorithm designed in the XMT model usually permits each thread to progress at its own speed from its initiating spawn to the end of the section, without ever having to busy-wait for other threads.

The Tesla Architecture. Fig. 1.b depicts an overview of the NVIDIA Tesla architecture. It consists of an array of Streaming Multiprocessors (SMs), connected through an interconnection network to a number of memory controllers and off-chip DRAM modules. Each SM contains a shared register file, shared memory, constant and instruction caches, special function units and several Streaming Processors (SPs) with integer and floating point ALU pipelines. SFUs are 4-wide vector units that can handle complex floating point operations. More information on the GPU architecture and programming can be found in [23], [29].

Compared Processors. In our experiments we physically measure runtime and power of an NVIDIA GTX280 and simulate a 1024-TCU XMT processor, XMT1024 (Sec. IV). Table II lists the specifications of these processors. GTX280 is used as a baseline in evaluating the envisioned XMT chip. Previous work estimated the XMT1024 silicon area to be equivalent to that of GTX280 [7], and the table outlines the hardware resources that were considered in that derivation. The DRAM latency and bandwidth of XMT1024 were set to approximately match the specifications of the GPU. The clock frequency of the XMT chip is set so that the power envelope of the base case described in Sec. III does not exceed that of the GPU. In Sec. V, we will consider different cases which will require the XMT clock frequencies to be modified accordingly.

Fundamental differences. In XMT, complex functional units are sacrificed for a higher number of TCUs with dedicated program counters. Unlike GTX280, a TCU can execute a thread at its own speed and take different paths on conditional statements without affecting the performance of other TCUs. A serial on-chip processor enables fast switching between serial and parallel modes. Moreover, the interconnection network is specifically designed to boost the bandwidth for irregular memory traffic, at the cost of reduced peak bandwidth. These

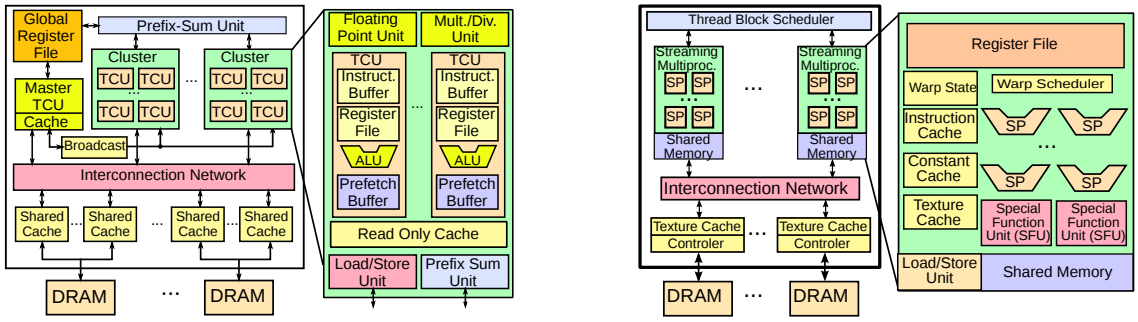

Fig. 1: (a) The XMT architecture. (b) The Tesla architecture.

	GTX280	XMT1024
Principal Computational Resources		
Cores	240 SP, 60 SFU	1024 TCU
Integer Units	240 ALU+MDU	1024 ALU, 64 MDU
Float. Point Units	240 FPU, 60 SFU	64 FPU
On-chip Memory		
Registers	1920KB	128KB
Prefetch Buffers	–	32KB
Regular caches	480KB	4104KB
Constant cache	240KB	128KB
Texture cache	480KB	–
Other Parameters		
Pipeline clk. freq.	1.3 GHz	1.3 GHz
Interconnect clk. freq.	650 MHz	1.3 GHz
Bandwidth to DRAM	141.7 GB/sec (peak theoretical)	
Fab. technology	65 nm	
Silicon area	576 mm^2	

TABLE II: Hardware specifications of GTX280 and the simulated XMT configuration. Values that span both columns are common to GTX280 and XMT1024.

design choices are geared towards better support for the XMT's distinctive programming model and improving the runtime of irregular parallel programs.

III. XMTSim: The Cycle-Accurate XMT Simulator

We simulated the XMTC programs on XMTSim, the cycle-accurate simulator of the XMT architecture [20]. XMTSim is modeled after the FPGA/ASIC prototypes of XMT and it can be customized to realistically simulate any configuration, beyond the resource limitations of the prototype. At this time, off chip buses and DRAM modules are modeled as simple latency components in the simulator. The XMT compiler and simulation environment are publicly available [20].

Cycle-Accuracy and Reliability of Power Estimation. The cycle-accuracy of XMTSim is verified against the FPGA prototype with a maximum error of 15%. This error is not necessarily an indication of inaccurate modeling, as in parallel processors the execution time can show variation between runs due to complex interactions between memory and thread progression.

We used validated tools to determine the model parameters, and in order to avoid excessively optimistic power estimates we added a worst case error margin whenever possible. Furthermore, Sec. V demonstrates the sensitivity of our results to potential errors in the initial power estimation. Note that validation of the power model through the FPGA prototype is not possible since an FPGA, by nature, does not constitute a reliable medium to run such a study. As a positive indication, the measurements presented in Sec. IV show a strong correlation between our simulator and GTX280 in power dissipation trends among the benchmarks.

The Power Model. The power model of XMTSim is based on the model proposed in [17]. According to this model, the simulation provides the access rate for each component (C_i), which is a value between 0 and 1. The power of a component is a linear function of the access rate with a constant offset.

$$\begin{aligned} Power(C_i) = & AccessRate(C_i) \cdot \\ & MaxActPower(C_i) + \\ & Const(C_i) \end{aligned} \quad (1)$$

C is the set of microarchitectural components for which we estimate the power in a simulated chip. $Const(C_i)$ is the power of a component which is spent regardless of its activity (static power). $MaxActPower(C_i)$ is the upper bound on power that depends on the activity. These two parameters are constants and are statically determined per component. XMTSim utilizes internal counters that monitor the activity of each architectural component and produce its $AccessRate(C_i)$.

The $Const(C_i)$ parameter includes the static power of a component and may also include a portion of the dynamic power depending on factors such as efficiency of the techniques that manage the dynamic power (i.e., clock gating) applied to the circuits. In our model we use the same assumption as the Wattch power simulator [6] and regard 10% of the maximum dynamic power as constant.

Table III lists the cumulative values of the $Const(C_i)$ and $MaxActPower(C_i)$ parameters for all groups of simulated components. In most cases, parameter values were obtained from McPAT 0.9 [22] and Cacti 6.5 [26], [43]. The DRAM parameters were based on the information in [31].

The designs of the shared caches and the ICN are unique to XMT and their power model parameters cannot be reliably estimated using the above tools[2]. Instead, they are estimated from our 200MHz 90nm ASIC prototype, using the Synopsys PrimeTime tool [35], and the implementation in [1] for the ICN, as indicated in Table III. The power values obtained from Synopsys were scaled following the relationship between

[2]Shared caches of XMT contain a significant amount of logic circuitry for serving multiple outstanding requests simultaneously. Cacti estimates for the shared caches were found to be much lower than our estimates.

Component	MaxActPower	Const	Source
Computing Clusters			
TCU Pipeline	51.2W	13.3W	McPAT
ALU	122.9W	20.5W	McPAT
MDU	21.1W	6.4W	McPAT, ASIC
FPU	29W	3.2W	McPAT, ASIC
Register File	30.8W	∼0W	McPAT
Instr. Cache	15.4W	1W	Cacti
Read-only Cache	2.7W	300mW	Cacti
Pref. Buffer	18.4W	2W	McPAT
Memory System			
Interconnect	28.5W	7.2W	ASIC [1]
Mem. Contr./DRAM	44.8W	104mW	McPAT, [31]
Shared Cache	58.9W	19.2W	ASIC

TABLE III: Power model parameters for XMT1024.

power and clock frequency/voltage.

$$P_{dyn} \sim f \cdot V^2 \qquad P_{stat} \sim V^2 \qquad V = 0.22 \cdot f + 0.86 \quad (2)$$

P_{dyn} is dynamic power and P_{stat} is static power. The voltage is reduced at lower clock frequencies in order to save further power. We use the published data on the Intel Pentium M765 and AMD Athlon 4000+ processors [24], to determine the minimum feasible voltage for a given clock frequency in GHz. When feature size scales down, power ideally scales in proportion to the square of the feature size [5]. From 90nm for the ASIC prototype to 65nm for envisioned XMT1024, we use a scaling factor of $(65nm/90nm)^2 \approx 0.5$, but this assumption is subject to further analysis in Section V.

Next, we provide details on the power model of major microarchitectural components in XMTSim.

Computing Clusters. The power dissipation of an XMT cluster is calculated as the sum of the individual elements indicated in Table III. The access rate of a TCU pipeline is calculated according to the number of instructions that are fetched and executed, which is a simple but sufficiently accurate approximation. For the integer and floating point units (including arbitration), access rates are the ratio of their throughputs to the maximum throughput. The remainder of the units are all memory array structures and their access rates are computed according to the number of reads and writes they serve.

Memory Controllers, DRAM and Global Shared Caches. The access rate of these components are calculated as the ratio of the requests served to the maximum number of requests that can be served over the sampling period (i.e. one request per cycle).

Global Operations and Serial Processor. We omit the power spent on global operations, since the total gate counts of the circuits that perform these operations were found to be insignificant with respect to the other components, and these operations make up a negligible portion of execution time. For example, prefix-sum operations and global register file accesses make up less than 1.5% of the total number of instructions among all the benchmarks. We also omit the power of the XMT serial processor, which is only active during serial sections of the XMTC code and when parallel TCUs are inactive. None of our benchmarks contain significant portions of serial code: the number of serial instructions, in all cases, is less than 0.005% of the total number of instructions executed.

Interconnection Network. The power of the Mesh-of-Trees (MoT) ICN includes the total cost of communication from all TCUs to the shared caches and back. The access rate for the ICN is equal to its throughput ratio, which is the ratio of the packets transferred between TCUs and shared caches to the maximum number of packets that can be transferred over the sampling period.

The power cost of the ICN can be broken into various parts [19], which fit into the framework of Eq. 1 in the following way. The power spent by the reading and writing of registers, and the charge/discharge of capacitive loads due to wires and repeater inputs can be modeled as proportional to the activity (i.e. number of transferred packets). All packets travel the same number of buffers in the MoT-ICN and the wire distance they travel can be approximated as a constant which is the average of all possible paths. The power of the arbiters is modeled as a worst-case constant, as it is not feasible to model it accurately in a fast simulator.

The power of the interconnection network (ICN) is a central theme in Sec. V, which will explore various scenarios considering possible estimation errors in the power model.

The Thermal Model. For thermal estimation, we incorporated HotSpot, an accurate and fast thermal solver [33]. The heatsink convection thermal resistance was set to $0.15K/W$ in order to follow the power-temperature trend observed for GTX280 in Sec. IV, and the ambient temperature was set to 45C. XMTSim accounts for the effect of temperature on leakage power via a linear approximation [34].

IV. GPU Measurements and Simulation Results

Power envelope is the main constraint in our study and in this section we aim to show that the performance advantage of XMT1024 previously demonstrated in [7] holds under this constraint. First, we provide a list of our benchmarks, followed by the results from the GTX280 measurement setup and the XMT simulation environment. We report the benchmark execution times, power dissipation values and average temperatures on both platforms. We also compare the execution time and energy consumption.

As mentioned earlier, the power envelope of many-core chips is more closely related to their thermal feasibility than is the case with large serial processors. Many-cores can be organized in thermally efficient floorplans such as the one in [15]. As a result, activity is less likely to be focused on a particular area of the chip for power intensive workloads, and temperature is more likely to be uniformly distributed (unlike serial processors, in which hotspots are common). In some cases, as observed in [14], the temperature of the memory controllers might surpass the temperature of the rest of the chip. However, this typically happens for low power benchmarks such as *Bfs*, which are heavy on memory operations but not on computation.

We thermally simulated an XMT floorplan, in which caches and clusters are organized in a checkerboard pattern and the ICN is routed through dedicated strips. For the most power-hungry benchmarks, maximum variation between adjacent

Name	Description	Dataset	Type	GTX280			XMT1024		
				Time	Power	Tempr.	Time	Power	Tempr.
Bfs	Breadth-First Search on graphs [9]	1M nodes, 6M edges	Irregular	16.3ms	155W	74C	1.34ms	161W	70C
Bprop	Back Propagation machine learning algorithm [9]	64K nodes	Irregular	15ms	97W	61C	2.26ms	98W	65C
Conv	Image convolution kernel with separable filter [29]	1024x512	Regular	0.18ms	180W	78C	0.69ms	179W	81C
Msort	Merge-sort algorithm [12], [32]	1M keys	Irregular	33.3ms	120W	70C	2.77ms	144W	71C
Nw	Needleman-Wunsch sequence alignment [9]	2x2048 sequences	Irregular	13.4ms	116W	67C	1.46ms	136W	71C
Reduct	Parallel reduction (sum) [29]	16M elts.	Regular	0.1ms	156W	74C	0.52ms	165W	75C
Spmv	Sparse matrix - vector multiplication. [3]	36Kx36K, 4M non-zero	Irregular	0.9ms	200W	80C	0.23ms	189W	78C

TABLE IV: Benchmarks and results of experiments.

blocks was only $1C$. However, from the midpoint of the chip towards the edges, temperature can gradually drop by up to $4C$. This is due to the greater lateral dissipation of heat at the edges and does not change the relationship between power and temperature. These results as well as the linear relationship between the power and temperature of the GTX280 chip [14] strengthens the argument of using power envelope as relevant metric.

Benchmarks. Our benchmarks are listed in Table IV. The selection of benchmarks is guided by the fact that a "general-purpose" architecture should provide good performance on both regular and irregular applications. Regularity, in this context, implies regularity of memory accesses and a consistently high amount of parallelism. Regular programs are the forte of the GPU concept as they can efficiently take advantage of the raw computing capacity of the GPU. In contrast, XMT excels in more general-purpose type irregular programs, which are outside the realm of the GPUs. An example irregular program is Bfs [38]. In Bfs, every newly discovered edge is traversed in a new parallel thread. The amount of parallelism and the memory access patterns are not predictable, in contrast to regular DSP-type applications, since the graph structure is not known in advance.

We selected benchmarks whose GPU results are published, and CUDA source code was made available by authors. This ensures that we are using the most optimized code for the CUDA implementation, whigh is highly tuned for GPUs. All benchmarks use single-precision floating point arithmetic only, to allow for a fair comparison with Tesla.

GPU Measurements. A P4460 Power Meter [16] is orchestrated to measure the total power of the computer system that we use in our experiments. The system is configured with a dual-core AMD Opteron 2218 CPU, an NVIDIA GeForce 280 GPU card and RedHat Linux 5.6 OS. The temperatures of the CPU cores and the GPU are sampled every second via the commands `sensors` and `nvclock -T`. The clock frequencies of the CPU cores and the GPU are monitored via the commands `cat /proc/cpuinfo` and `nvclock -s`.

Our preliminary experiments show that the effect of the CPU performance is negligible on the runtime of our benchmarks. Therefore, for reducing its effect on overall power, the clock frequency of the CPU was set to its minimum value, $1GHz$. While the GPU card does not provide an interface for manually configuring its clock, we observed that during the execution of the benchmarks, core and memory controller frequencies remain at the maximum values.

The GTX280 column of Table IV lists the data collected from the execution of the benchmarks on the GPU. The power of a benchmark is computed by subtracting the idle power of the system without the GPU card ($98W$) from the measured total power. Each benchmark is modified to execute in a loop long enough to let the system reach a stable temperature, and execution time is reported per iteration. The initialization phases, during which the input data is read and moved to the GPU memory, are omitted from the measurements [3]. We also measured the idle power of the GPU card at various temperatures to better understand its dependency on the operating temperature. Results range from 76W at 61C to 85W at 80C. We use this data to calibrate the leakage/temperature model of XMTSim mentioned in Sec. III. The CPU core temperatures deviate at most $2°C$ from the initial temperature, which is not expected to affect the leakage power of the CPU significantly.

XMT Simulations and Comparison with GTX280. The simulation results for XMT1024 are given in the XMT1024 column of Table IV. We need to ensure that the two most power intensive benchmarks on XMT1024, $Spmv$ and $Conv$, do not surpass the maximum power on GTX280, which is 200W for $Spmv$. Under these restrictions, we determined that the XMT1024 chip can be clocked at the same frequency as GTX280, 1.3 GHz.

Fig. 2 presents the speedups of the benchmarks on XMT1024 relative to GTX280. Fig. 3 then shows the ratio of benchmark energy on GTX280 to those on XMT1024.

As expected, XMT1024 performance exceeds GTX280 on irregular benchmarks (8.8x speedup) while GTX280 performs better for the regular benchmarks (0.24x slowdown). The average is 6.34x among all benchmarks. The trends of the speedups match those demonstrated in [7] and the differences in the exact values are caused by improved simulation models and the newer version of the CUDA compiler used in our experiments.

The two chips show similar power trends among the benchmarks. On XMT1024, the average power of irregular benchmarks, 138W, is lower than the average power of the two regular benchmarks, which is 168W. A similar trend can be observed for GTX280. We see that XMT does not require a higher power envelope in order to boost the performance

[3]In XMT, the Master TCU and the TCUs share the same memory space, and no explicit data move operations are required. However, this advantage of XMT over Tesla is not reflected in our experiments.

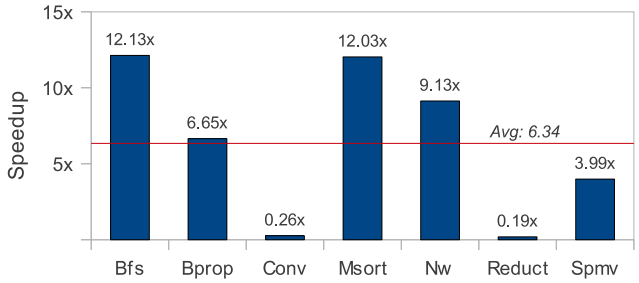

Fig. 2: Speedups of XMT1024 with respect to GTX280. A value less than 1 denotes slowdown.

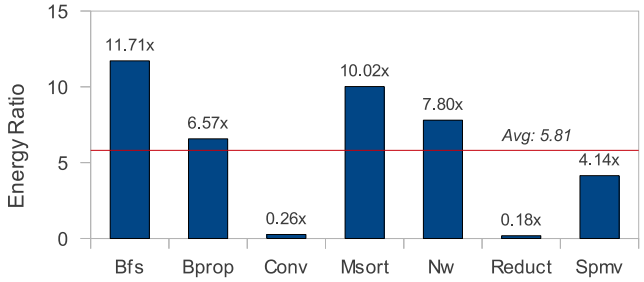

Fig. 3: Ratio of benchmark energy on GTX280 to energy on XMT1024.

of the irregular benchmarks, for which efficient handling of irregular communication and control flow is more important. Acceleration of regular benchmarks on GTX280, however, reflects directly on its power envelope. Predictable flow of regular programs allows the GPU to extract more arithmetic operations, given sufficient peak ICN bandwidth and peak computation capacity. It can be noted that $Spmv$ is an exception to the rule-of-thumb that irregular programs spend lower power. In the case of $Spmv$, irregularity is caused by the complexity of the memory addressing, whereas the high density of floating point operations elevates the power.

The energy comparison yields results similar in trend to the speedup results (ratio of 8.1 for irregular and 0.22 for regular benchmarks; 5.81 for all benchmarks). Energy is the product of power and execution time, and the relation of power dissipation among the benchmarks is alike between the two chips, as can be seen in Table IV. Therefore, the energy is roughly proportional to the speedups.

V. Sensitivity of Results to Power Model Errors

Early estimates in a simulation based architecture study are always prone to errors due to possible deviations in the parameters used in the power model. In this section we will challenge some of the assumptions of Sec. III that led to the results in Sec. IV: The accuracy of the parameter values obtained from the McPAT and Cacti tools, the ideal scaling factor of 0.5 used to scale the power of the shared caches from 90nm to 65nm, and the overall interconnection network power.

These modifications will increase the estimated power dissipation, which will in turn cause the maximum power observed for the benchmarks to surpass GTX280. To satisfy the power envelope constraint in our experiments, we reduce the clock frequency and voltage of the XMT computing clusters and the

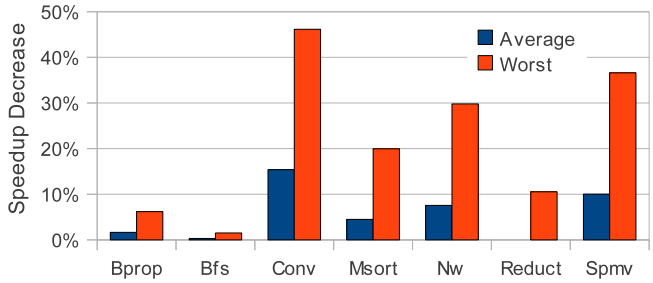

Fig. 4: Decrease in XMT vs. GPU speedups with average case and worst case assumptions for power model parameters.

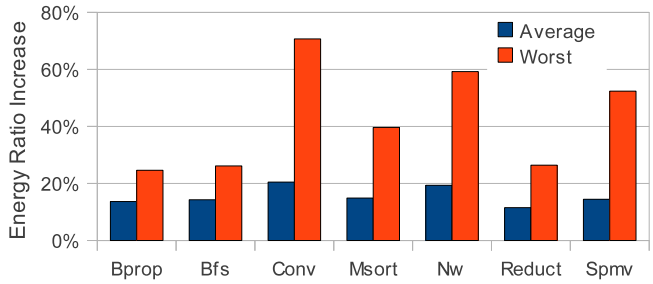

Fig. 5: Increase in benchmark energy ratios on XMT with average case and worst case assumptions for power model parameters.

ICN according to Eq. 2[4]. As a result, XMT runtimes increase and the speedups shown in Fig. 2 degrade (GTX280 runtimes do not change).

Cluster, Cache and Memory Controller. In their validation study, the authors of McPAT observed that the total power dissipation of the modeled processors may exceed the predictions by up to 29%. Therefore, we added 29% to the value of all the parameter obtained from McPAT to account for the worst case error. As a worst case assumption we also set the technology scaling factor to 1, which results in no scaling. In addition to the worst case assumptions, we explored an average case, for which we used 15% prediction correction for McPAT and Cacti and a technology scaling factor of 0.75. We measure the speedup and energy ratio changes with respect to the best case values given in Figs. 2 and 3.

Fig. 4 shows that the average speedups decrease by 5.6% and 21.5% for the average and the worst cases, respectively. The energy ratios of the benchmarks increase by average of 15.5% for the average and 42.7% for the worst case (Fig. 5). For each case, we ran an exhaustive search for cluster and ICN frequencies between 650MHz and 1.3GHz and chose the frequencies that give the maximum average speedup while staying under the power envelope of 200W. For the average case the cluster and ICN frequencies were 1.1GHz and 1.3GHz, respectively. For the worst case, they were 650MHz and 1.3GHz. The optimization tends to lower the cluster frequencies and keep the ICN frequency as high as possible since the irregular benchmarks are more sensitive to the rate of data flow rather than computation. Also, with the current parameters, ICN power contributes to the total power less than the cluster power and the effect of reducing the ICN frequency on power is relatively lower.

[4]We assume that clusters and ICN are in separate voltage and frequency domains as in GTX280.

Conv and *Spmv* are the two most power intensive benchmarks and they are also the most affected by lowering the cluster frequency. Under best-case assumptions, *Conv* is very balanced in the ratio of computation to memory operations, and *Spmv* has a relatively high number of FP and integer operations that it cannot overlap with memory operations. Reducing the cluster frequency slows down the computation phases in both *Conv* and *Spmv*. *Msort* and *Nw* are programmed with a high number of parallel sections (i.e., high synchronization) and they are also affected by the lower cluster frequency, as it slows down the synchronization process. *Bfs*, *Bprop* and *Reduct* are not sensitive to the cluster frequency since they spend most of the time in memory operations. The trend of the energy increase data in Fig. 5 is similar to the data in Fig. 4, however unlike Fig. 4, all benchmarks are affected. This is expected as the average and worst case assumptions essentially increase the overall power dissipation.

Interconnection Network. The ICN model parameters used in Sec. III may be inaccurate due to a number of factors. First, the implementation on which we based our model [1] was placed and routed for a smaller chip area than we anticipate for XMT1024, and therefore might underestimate the power required to drive longer wires. Second, as previously mentioned, the ideal technology scaling factor we used in estimating the parameters might not be realistic. To accommodate for these inaccuracies, we run a study to show the sensitivity of the results we previously presented to the possible errors in ICN power estimation.

We assume that the errors might be manifested in the form of two parameters that we will explore: P_{max} - ICN power at maximum activity and clock frequency, and α - the activity-power correlation at the maximum clock frequency, described as follows:

$$\begin{aligned} P_{max} &= MaxActPower(ICN) + Const(ICN) \\ \alpha &= MaxActPower(ICN)/Const(ICN) \end{aligned} \quad (3)$$

$MaxActPower$ and $Const$ are the parameters in Eq. 1. The motivation for exploring α as a parameter arises from the fact that ICN is the only major distributed component in the XMT chip. Efficient management of power (including dynamic power) in interconnection networks is an open research question [25], which affects the activity-power correlation implied by α. For example, if clock-gating is not implemented very efficiently the dynamic power may contain a large constant part. Other distributed components are the prefix-sum network and the parallel instruction broadcast, which do not contribute to power significantly. The remainder of the components in the chip are *off-the-shelf* parts, for which optimal designs exist.

The values of P_{max} and α in Figs. 6 and 7 (which will be explained next) are given for the maximum clock frequency of 1.3GHz. However, ICN frequency may be reduced as a part of the optimization process, which will change the effective values of these parameters. For example, assume that $MaxActPower(ICN) + Const(ICN)$ is set to 125W. The power envelope required with this parameter values is more than the 200W, which is our constraint. An exhaustive search looking for the maximum speedup point within the power envelope is performed, and the ICN and cluster clock frequencies are then both set to 650MHz. Since the ICN clock and voltage are both lowered, the sum of $MaxActPower(ICN) + Const(ICN)$ decreases to 78W.

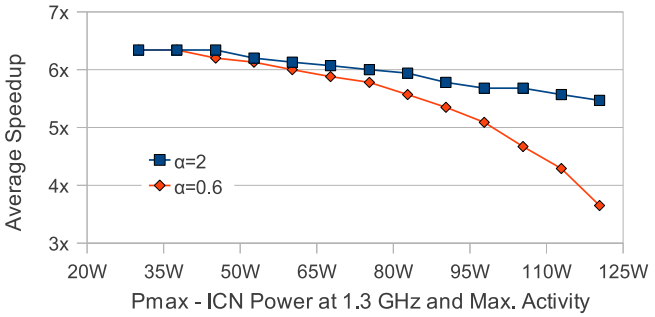

Fig. 6: Degradation in the average speedup with different ICN power scenarios. P_{max} and activity-power correlation (α) are regarded as variables. Their values are with respect to 1.3GHz clock frequency.

Fig. 6, shows the average speedups for different scenarios of the ICN power model. In order to compress the large amount of data, we only show the average of the speedups of all benchmarks. Values of α, 2 and 0.6, are determined such that 10% (which was the initial value from Sec. III) and 50% of the maximum dynamic power is regarded as a constant, respectively. As in the previous section, we ran an exhaustive search for cluster and ICN frequencies ranging from 650MHz to 1.3GHz for finding the suitable design point. The change in speedups for the $\alpha = 2$ series of the plot is relatively low, whereas the decline for the $\alpha = 0.6$ series is faster. A lower value of α will cause the ICN to spend more power regardless of its activity, which will increase the overall power dissipation. As a result, the solutions found have lower clock frequencies.

Putting it together. In Fig. 7, we combine various scenarios from the previous two sections, namely the variations in P_{max} of ICN with the best, average and worst case scenarios for the rest of the chip. The α parameter for ICN is set to the worst case value of 0.6. For the data points that do not exist in the plot, no solution exists within our search space. It should be noted that even for the worst scenarios the average speedup of XMT, is greater than 3x.

We also included the power breakdown of the XMT chip for two representative cases. Fig. 8(a) is the best case assumptions with minimum ICN power and $\alpha = 2$, and Fig. 8(a) is the average case assumptions with P_{max} of 82W and $\alpha = 0.6$ for ICN. The average speedup for the first case was 6.34 and for the second case it was approximately 4.4. As can be noted, the ratio of ICN power to cluster power is higher in the second case since ICN power is increased more than the cluster power as a part of the assumptions.

VI. Conclusion

This paper demonstrates the runtime advantage of XMT under power constraints by comparing it to a state-of-the-art GPU as the baseline. Consequently, it strengthens the claim

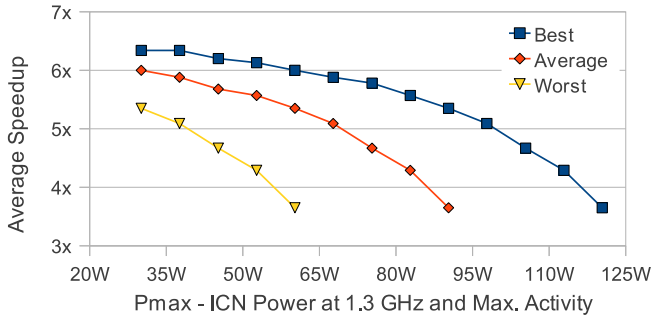

Fig. 7: Degradation in the average speedup with different chip power scenarios. P_{max} for ICN and best, average and worst case for the rest of the chip are regarded as variables. ICN activity-power correlation (α) is set to 0.6. P_{max} and α are with respect to 1.3GHz clock frequency.

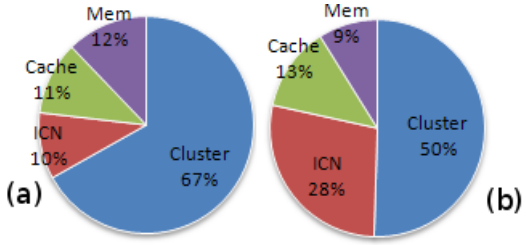

Fig. 8: Power breakdown for two cases: (a) $\alpha = 2$, $P_{max} = 33W$, best case for the rest of the chip, and (b) $\alpha = 0.6$, $P_{max} = 82W$, average case for the rest of the chip.

that XMT is competitive with GPUs, as a general-purpose many-core architecture, on performance and programmer's productivity when using comparable resources. Caragea et al. [7] compared a simulated 1024-core XMT chip with a silicon-area equivalent GPU, but left open the issue of power. We complete the comparison by establishing that the XMT chip does not require a higher power envelope than the GPU. Namely, this verifies that a general-purpose desktop XMT computer is as feasible as a current high-end GPU. Performance [7] already demonstrated a variety of workloads for which a simulated 1024-core XMT chip comes ahead; it also showed that XMT does not fall behind significantly on workloads for which GPU is particularly stronger. Ease-of-programming As its programming is PRAM-like, XMT has a big advantage, documented in [13], [30], [36], [40], on ease-of-programming and effectiveness on achieving speedups. XMT also offers backwards compatibility on serial code and rewards even small amount of parallelism with speedups over uni-processing.

In view of the results of the current paper, XMT is well-positioned for the mainstream general-purpose platform of the future, especially when coupled with a GPU (see, e.g., [10]) for GPU-specific tasks.

Acknowledgment

Partial support by NSF grants 0325393, CCF-0811504, 0834373 and 0926237 is gratefully acknowledged.

References

[1] A. O. Balkan, "Mesh-of-trees interconnection network for an explicitly multi-threaded parallel computer architecture," Ph.D. dissertation, University of Maryland, 2008.
[2] A. O. Balkan, M. N. Horak *et al.*, "Layout-accurate design and implementation of a high-throughput interconnection network for single-chip parallel processing," in *Proc. Hot Interconnects*, 2007.
[3] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Proc. SC*, 2009.
[4] S. Borkar, "Thousand core chips: a technology perspective," in *Proc. DAC*, 2007.
[5] S. Borkar and A. A. Chien, "The future of microprocessors," *CACM*, vol. 54, pp. 67–77, 2011.
[6] D. Brooks, V. Tiwari *et al.*, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proc. ISCA*, 2000.
[7] G. Caragea, F. Keceli *et al.*, "General-purpose vs. GPU: Comparison of many-cores on irregular workloads," in *Proc. HotPar*, 2010.
[8] G. C. Caragea, B. Saybasili *et al.*, "Performance potential of an easy-to-program pram-on-chip prototype versus state-of-the-art processor," in *Proc. SPAA*, 2009.
[9] S. Che, M. Boyer *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IISWC*, 2009.
[10] T. M. DuBois, B. Lee *et al.*, "XMT-GPU: A PRAM architecture for graphics computation," in *Proc. ICPP*, 2008.
[11] A. Gottlieb, R. Grishman *et al.*, "The NYU ultracomputer – designing a MIMD, shared-memory parallel machine," *IEEE Trans. on Comp.*, vol. C-32, pp. 175–189, 1983.
[12] J. Hoberock and N. Bell, "Thrust: A parallel template library," 2009, version 1.1. [Online]. Available: http://www.meganewtons.com/
[13] L. Hochstein, V. R. Basili *et al.*, "A pilot study to compare programming effort for two parallel programming models," *J. Sys. and Softw.*, vol. 81, no. 11, pp. 1920 – 1930, 2008.
[14] S. Hong and H. Kim, "An integrated GPU power and performance model," in *Proc. ISCA*, 2010.
[15] W. Huang, M. R. Stan *et al.*, "Many-core design from a thermal perspective." in *Proc. DAC*, 2008.
[16] P. International, "P4460 Electricity Usage Monitor," www.p3international.com/products/p4460.html.
[17] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proc. Symp. on Micro.*, 2003.
[18] J. JáJá, *An introduction to parallel algorithms*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1992.
[19] A. B. Kahng, B. Li *et al.*, "Orion 2.0: A power-area simulator for interconnection networks," *IEEE Trans. on VLSI Sys.*, 2011, to appear.
[20] F. Keceli, A. Tzannes *et al.*, "Toolchain for programming, simulating and studying the XMT many-core architecture," in *Proc. HIPS*, 2011, in conj. with IPDPS.
[21] J. Keller, C. Kessler *et al.*, *Practical PRAM Programming*. New York, NY, USA: John Wiley & Sons, Inc., 2001.
[22] S. Li, J. H. Ahn *et al.*, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. Symp. on Micro.*, 2009.
[23] E. Lindholm, J. Nickolls *et al.*, "NVIDIA Tesla: A unified graphics and computing architecture," *IEEE Micro*, vol. 28, pp. 39–55, 2008.
[24] Y. Liu, H. Liang *et al.*, "Scheduling for energy efficiency and fault tolerance in hard real-time systems," in *Proc. DATE*, 2010.
[25] R. Marculescu, U. Ogras *et al.*, "Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives," *IEEE Trans. on CAD of ICs and Systems*, vol. 28, pp. 3 – 21, 2009.
[26] N. Muralimanohar, R. Balasubramonian *et al.*, "CACTI 6.0: A Tool to Model Large Caches," HP Laboratories, Tech. Rep., 2005.
[27] D. Naishlos, J. Nuzman *et al.*, "Towards a first vertical prototyping of an extremely fine-grained parallel programming approach," in *Proc. SPAA*, 2001.
[28] J. Nickolls and W. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56 –69, March 2010.
[29] NVIDIA, "CUDA SDK 3.2," www.nvidia.com/cuda, 2010.
[30] D. Padua and U. Vishkin, "Joint UIUC/UMD parallel algorithms/programming course," in *Proc. EduPar*, 2011, in conj. with IPDPS.
[31] Samsung, "Samsung Green GDDR5," www.samsung.com/global/business/semiconductor/Greenmemory/Downloads/Documents/downloads/green_gddr5.pdf.
[32] N. Satish, M. Harris *et al.*, "Designing efficient sorting algorithms for manycore GPUs," in *Proc. IPDPS*, 2009.
[33] K. Skadron, M. R. Stan *et al.*, "Temperature-aware microarchitecture," in *Proc. ISCA*, 2003.
[34] H. Su, F. Liu *et al.*, "Full chip leakage-estimation considering power supply and temperature variations," in *Proc. ISLPED*, 2003.
[35] Synopsys, "PrimeTime," www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx.
[36] S. Torbert, U. Vishkin *et al.*, "Is teaching parallel algorithmic thinking to high-school student possible? one teachers experience," in *Proc. SIGCSE*, 2010.
[37] U. Vishkin, "Using simple abstraction to guide the reinvention of computing for parallelism," *CACM*, vol. 54, no. 1, pp. 75–85, Jan. 2011.
[38] U. Vishkin, G. C. Caragea *et al.*, *Handbook of Parallel Computing: Models, Algorithms and Applications*. CRC Press, 2007, ch. Models for Advancing PRAM and Other Algorithms into Parallel Programs for a PRAM-On-Chip Platform.
[39] U. Vishkin, S. Dascal *et al.*, "Explicit multi-threading (XMT) bridging models for instruction parallelism," in *Proc. SPAA*, 1998.
[40] U. Vishkin, R. Tzur *et al.*, "Programming for high schools," www.umiacs.umd.edu/~vishkin/XMT/CS4HS_PATfinal.ppt, July 2009, Keynote, The CS4HS Workshop.
[41] X. Wen and U. Vishkin, "PRAM-on-chip: first commitment to silicon," in *Proc. SPAA*, 2007.
[42] X. Wen and U. Vishkin, "FPGA-based prototype of a PRAM on-chip processor," in *Proc. Comp. Front.*, 2008.
[43] S. Wilton and N. Jouppi, "CACTI: an enhanced cache access and cycle time model," *J. Solid-State Circ.*, vol. 31, no. 5, pp. 677–688, May 1996.