Thermal Management of a Many-Core Processor under Fine-Grained Parallelism

Fuat Keceli[1], Tali Moreshet[2], and Uzi Vishkin[1]

[1] University of Maryland, College Park, MD, USA
[2] Swarthmore College, Swarthmore, PA, USA

Abstract. In this paper, we present the work in progress that studies the run-time impact of various DTM techniques on a proposed 1024-core XMT chip. XMT aims to improve single task performance using fine-grained parallelism. Via simulations, we show that relative to a general global scheme, speedups of up to 46% with a dedicated interconnection controller and 22% with distributed control of computing clusters are possible. Our findings lead to several high level insights that can impact the design of a broader family of shared memory many-core systems.

1 Introduction

Thermal feasibility has become a first-class architectural design constraint reflected in specifications of modern processors. It is typical to incorporate dynamic thermal management (DTM) in order to improve the power envelope without having to adopt more expensive cooling solutions. Going forward, it is important to advance the understanding of DTM techniques for efficiently supporting the architecture trend of increase in core count.

We base our work on the eXplicit Multi-Threading (XMT) architecture. XMT is a general-purpose many-core platform for fine-grained parallel programs, with significant evidence on ease-of-programming (e.g., [24]) and competitive performance (e.g., [3]). The reasons for choosing XMT as our platform are: *(a)* As noted in [25], ease-of-programming is crucial for the success of parallel computers and XMT constitutes a competitive and realistic direction in this respect, *(b)* The XMT simulator allows for evaluation of DTM techniques and such an infrastructure is not publicly available for other current many-core platforms.

In this paper, we evaluate the potential benefits of several DTM techniques on XMT. We focus on a system which executes one parallel task at a time. The relevance and the novelty of our work can be better understood by answering the following two questions.

Why Is Single Task Fine-Grained Parallelism Important? On a general-purpose many-core system the number of concurrent tasks is unlikely to often reach the number of cores (i.e., thousands). Parallelizing the most time consuming tasks is a sensible way for both improved performance and taking advantage

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 249–259, 2012.

of the plurality of cores. The main obstacle then is the difficulty of programming for single-task parallelism. Scalable fine-grained parallelism is natural for easy-to-program approaches such as XMT.

What Is New in Many-Core DTM? DTM on current multi-cores mainly capitalizes on the fact that cores show different activity levels under multi-tasked workloads [5]. In a single-tasked many-core, the source of imbalance is likely to lie in the structures that did not exist in the former architectures such as the large scale on-chip interconnection network (ICN) and distributed shared caches.

Contribution. This paper introduces XMTSim+dtm, a new, DTM-enabled cycle-accurate simulator based on XMTSim [16]. Using XMTSim+dtm we measure the performance improvements introduced by several DTM techniques for a 1024-core XMT chip. We compare techniques that are tailored for a many-core architecture against a global DTM (GDVFS), which is not specific to many-cores. Following are the highlights of the insights we provide: (a) Workloads with scattered irregular memory accesses benefit more from a dedicated ICN controller (up to 46% runtime improvement over GDVFS). (b) In XMT, cores are arranged in clusters. Distributed DTM decisions at the clusters provide up to 22% improvement over GDVFS for high-computation parallel programs, yet overall performance may not justify the implementation overhead.

Our work is relevant for architectures that consider similar design choices as XMT (for example the Plural system [7]) which promote the ability to handle both regular and irregular parallel general-purpose applications competitively (see Section 3.1 for a definition of regular and irregular). These design choices include an integrated serial processor, no caches that are local to parallel cores, and a parallel core design that provides for a true SPMD implementation. We aim to establish high-level guidelines for designers of such systems.

A comprehensive body of previous work is dedicated to dynamic power and thermal management techniques for multi-core processors [13]. However, in most cases (e.g., [6, 12, 21]), authors assume a pool of uncorrelated serial benchmarks as their workload, and capitalize on the variance in the execution profiles of these benchmarks. The study by Ma, et al. [22] is notable, as they simulate a set of parallel benchmarks, however, it focuses on power rather than thermal management and considers up to only 128 cores. To our knowledge, our work is among the first to evaluate DTM techniques on a many-core processor for single task parallelism.

2 Experimental Platform

2.1 The XMT Architecture

The primary goal of the eXplicit Multi-Threading (XMT) general-purpose computer architecture [28] 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], and

the latent, though not widespread, familiarity with it. Driven by the repeated programming difficulties of parallel machines, ease-of-programming was a leading design objective of XMT. Indeed, considerable amount of evidence was developed on *ease of teaching* and improved *development time* with XMT (e.g., [24])[1].

The XMT architecture 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 to mutually-exclusive shared cache modules by a high-throughput interconnection network [1]. XMT does not feature writable private caches. Moreover, since the cache modules are mutually exclusive, no cache coherence is required. TCUs include lightweight ALUs, but the more heavy-weight units are shared by all TCUs in a cluster. 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. Further details on the XMT architecture can be found in [28].

2.2 Simulation Environment

The simulation environment used in this work is XMTSim+dtm, an extension of publicly available XMTSim [16]. XMTSim is the cycle-accurate simulator of the XMT computer architecture, built to model the on-chip components that constitute the ASIC and FPGA prototypes of XMT. XMTSim has been validated against the FPGA prototype and cycle counts are shown to be accurate within a margin of 15% [17]. The parameters of the power model used in XMTSim is based on other validated tools: McPat [20] and Cacti [29]. The details of this estimation are the subject of a companion paper [15]. More details on the simulator than presented hereafter can be found in [14, 16, 17].

The high level specifications of the 1024-TCU XMT chip that we simulated in our experiments are given in Table 1a. The values in the table were compiled to match the chip area of an advanced GPU, NVIDIA GTX280. Details of this analysis are given in [3].

Power Estimation. XMTSim estimates power based on the activity that a benchmark induces on the clusters, ICN and the shared caches. Effect of die temperatures is included in calculation of the leakage power. The power model used in XMTSim is similar to the model proposed in [11] and explained in detail in [14, 17]. The model reflects the assumptions that a components implement clock gating and voltage gating. Table 1a lists the maximum powers of the components in the simulated XMT chip.

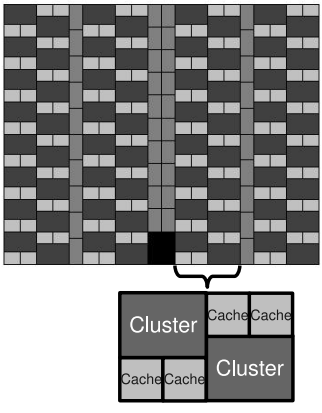
The Temperature Model. XMTSim uses HotSpot [27] to estimate the temperatures of the clusters and the maximum temperature of the area dedicated to the ICN routing. We set the thermal limit at 65C, whenever required, and

[1] A complete list of references can be found at
`http://www.umiacs.umd.edu/users/vishkin/XMT/index.shtml#publications`

Table 1. (a) The specifications of the 1024-TCU XMT. Given in parentheses are the maximum cumulative power for each group of components. (b) Checkerboard floorplan for a 1024 TCU XMT. Vertical strips with light gray color are reserved for interconnection network routing and the black colored rectangle is the Master TCU.

Processing Clusters (240.7W)
·1024 In-order 5–stage TCUs with ALUs, 2-bit br. prediction, 16 prefetch buffers
·64 Mult./Div. and 64 Float. Point units
·2K read-only cache per cluster
Interconnection Network (45.3W)
·64-to-128 Mesh-of-Trees [1]
Shared Parallel Cache (80.4W)
·128 modules x 32K. 4MB total
·2-way associative
Other Specifications
·Max. clock freq.: Clusters – 1.3 GHz, ICN – 2.6 GHz
·Max. DRAM bwidth.: 141.7 GB/sec

(a)



(b)

simulated our benchmarks with a heatsink convection resistance of 0.05W/K, observed in advanced cooling solutions [10]. Fig. 1b denotes the simulated XMT floorplan. It is motivated by previous work that shows it is thermally more efficient to place the clusters and shared caches in a checkerboard pattern rather than keeping them separately [10].

Simulating DTM. XMTSim samples the power and computes temperature at regular intervals. At each interval, DTM algorithm calculates a new set of frequencies for the clusters and the ICN. In order to avoid very long simulation times, we use steady-state temperature computations. Steady-state is an approximation to transient solutions for very long intervals with steady inputs. We observed that the behaviors of the kernels we simulated do not change significantly with larger data sets, except that the phases of consistent activity stretch in time. Therefore, we interpret the steady-state results obtained from simulating relatively short kernels with narrow sampling intervals as indicators of potential results from longer kernels. The sampling intervals, ranging from 5K to 200K clock cycles, are determined empirically to filter the noise in the activity patterns.

3 Performance under Thermal Constraints

In this section, we discuss the performance of the XMT chip under thermal constraints and evaluate a set of dynamic thermal management techniques that can potentially improve the performance. Our main objective is obtaining the

Table 2. Benchmark properties. Results are clock cycles, average power and temperature, consecutively

Name	Results	Description	Characteristic and Dataset
BFS-I	1.825M, 161W, 60C	Breadth-first search on graphs.	Hi-P. 1M nodes, 6M edges. Low cluster and moderate ICN activity. Irregular.
BFS-II	135.2M, 118W, 56C		Low-P. 200K nodes, 1.2M edges. Very low activity. Irregular.
Bprop	3.990M, 118W, 56C	Back Propagation machine learning alg.	Hi-P. 64K nodes. Low activity. Irregular.
Conv	0.885M, 215W, 66C	Image convolution with separable filter.	Hi-P. 1024x512. Highest activity. Regular.
FFT	4.905M, 194W, 63C	1-D fixed point Fast-Fourier transform.	Hi-P. 1M points. Moderate activity. Regular.
Mmult	10.6M, 180W, 63C	Multiplication of two integer matrices.	Low-P. 512x512 elts. Moderate cluster and high ICN activity. Regular.
Msort	3.625M, 161W, 60C	Merge-sort algorithm.	Hi-P. 1M keys. Variable moderate to low activity. Irregular.
NW	1.725M, 166W, 61C	Needleman-Wunsch sequence alignment.	Low-P. 2x2048 seqs. Variable moderate to low activity. Irregular.
Reduct	0.67M, 185W, 63C	Parallel reduction (sum).	Hi-P. 16M elts. Moderate cluster and high ICN activity. Regular.
Spmv	0.31M, 205W, 64C	Sparse matrix - vector multiplication.	Hi-P. 36Kx36K, 4M non-zero. Moderate cluster and high ICN activity. Irregular.

shortest execution time for a benchmark without exceeding a predetermined temperature limit. Note that energy efficiency is not within the scope of this objective.

3.1 Benchmarks

We consider it essential for a general-purpose architecture to perform well on a range of applications. Therefore we include both regular benchmarks, such as graphics processing, and irregular benchmarks, such as graph algorithms. In a typical regular benchmark, memory access addresses are predictable and there is no variability in control flow. In an irregular benchmark, memory access addresses and the control flow (if it is data dependent) are less predictable. Since it is our purpose to make the results relevant to other many-core platforms, we select benchmarks that commonly appear in the public domain [2, 4, 8, 23, 26]. Where applicable, benchmarks use single precision floating point format, except FFT, which uses fixed point arithmetics (i.e., utilizing the integer functional units instead of the FP units).

Table 2 provides a summary of the benchmarks that we use in our experiments. Execution times (in clock cycles), average power and temperature values are obtained from simulating benchmarks with no thermal constraint. Power and temperature estimations assume a global clock frequency of 1.3 GHz and $R_c = 0.05W/K$. Each benchmark is characterized in terms of its degree of parallelism, regularity and activity. More detail on these will follow next.

The degree of parallelism for a benchmark is defined to be low (*low-P*) if the number of TCUs executing threads is significantly smaller than the total number of TCUs when averaged over the execution time of the benchmark. Otherwise the benchmark is categorized as highly parallel (*hi-P*). According to Table 2, three of our benchmarks, *BFS-II*, *Mmult* and *NW*, are identified as low-P. In *Mmult*, the size of the multiplied matrices is 512×512 and each thread handles one row, therefore only half of the 1024 TCUs are utilized. *BFS-II* shows a random distribution of threads between 1 and 11 in each one of its 300K parallel sections. *NW* shows varying amount of parallelism between the iterations of a large number of synchronization steps (i.e., parallel sections in XMTC).

As mentioned above, regularity of a benchmark is affected by the predictability of memory access patterns and the control flow. For instance, *BFS*, *Msort* and *Spmv* are irregular due to their memory access patterns, whereas *BFS* also shows data dependent control flow. *FFT* is another benchmark with irregular memory access patterns, however it is classified as regular since the uniform distribution of work among TCUs was the dominant factor in this case. Some of the other factors that play a role in the regularity of a benchmark are the amount and variability of parallelism (e.g., *NW*), and memory bottlenecks (e.g., *Bprop*). *Bprop* is a complex irregular program with heavy memory queuing.

The activity profile of a benchmark plays an important role in the behavior of the system under thermal constraints, as we demonstrate in Section 3.3. It can be observed in Table 2 that there is a direct correlation between the regularity and activity/power value of a benchmark. For example, *Conv*, *Reduct* and *Mmult* are typical regular benchmarks with steady and high activity and power values.

In addition to regularity, cluster and ICN activities are determined by a number of additional factors, such as the computation to memory operation ratio of the threads and amount of parallelism (even if it is constant). An example is fixed-point *FFT*, which has lower activity than the other regular benchmarks, partly because it performs integer operations rather than floating point, spending less time in computation. Another example, *Mmult*, despite being very regular, is not as active as *Conv* due to the fact that it is a low-P benchmark.

3.2 Dynamic Thermal Management

Dynamic thermal management is the general term for various algorithms used to increase, or more efficiently utilize, the power envelope without exceeding a limit temperature at any location on the chip, as observed by thermal sensors. DTM alters the operation of the system during runtime via tools such as dynamic voltage and frequency scaling (DVFS), clock gating and voltage gating. These tools are commonly implemented in current processors [9, 13, 19].

In our experiments, we evaluated the following DTM techniques that are motivated by previous work on single and multi-cores [13]. We adapted these techniques to our many-core platform. The DTM decisions for each technique are determined during runtime and based on proportional-integral (PI) controllers which take temperatures as input and output the clock frequency and voltage level [27]. The clusters and ICN are assumed to feature one or more temperature

sensors that report their maximum temperatures. The exact sensor configuration is beyond the scope of this work. In the distributed DVFS algorithms, a controller assigned to a certain component in the chip (for example, the ICN or a cluster) responds to the temperature of that component. Further details can be found in [14].

Global DVFS (GDVFS): All clusters, caches and the ICN are connected to one central controller which converges to the maximum frequency/voltage values possible without exceeding limit temperature. Global DVFS is the simplest DTM technique in terms of physical implementation, therefore, any other technique should perform better in order to justify its added design complexity.

Coarse-Grain Distributed DVFS (CG-DDVFS): The ICN is assigned a separate controller while the rest of the chip remains connected to the global controller as in GDVFS. Some many-cores, such as GTX280, already have separate clock domains for the computation elements and the interconnection network, leading us to conclude that the implementation cost of this technique is acceptable.

Fine-Grain Distributed DVFS (FG-DDVFS): Each cluster is connected to an isolated voltage/frequency domain and assigned a separate controller. The shared caches are connected to a common domain and their frequency is set to the average frequency of the clusters. The ICN is assigned a dedicated controller as in CG-DDVFS. The implementation of distributed DVFS may be prohibitively expensive on large systems due to the high number of voltage/frequency domains.

3.3 Analysis of DTM Results

A chip with no dynamic thermal management is designed to work at the highest possible clock frequency with which it can tolerate the thermal stress of the worst case (i.e., the most active, most power intensive) benchmark. Consequently, applications that are not as thermally demanding are penalized, since they are subject to the same clock frequency. DTM techniques provide the highest improvement on the execution time of such benchmarks compared to the *no-DTM* case. Consistent with this statement, throughout this analysis it can be observed that benchmarks identified as low activity in Section 3.1 show the highest speedups with DTM techniques.

In evaluating the DTM techniques introduced in Section 3.2, we set an XMT chip with *no-DTM* as the baseline and express the performance of a DTM technique in terms of speedups over the baseline. We assume that the *no-DTM* system is optimized to run at the fastest clock frequency that is thermally feasible for *Conv*, which is the most thermally active benchmark according to Section 3.1. We determined the baseline clock frequency as 900MHz. The thermal limit was set at 65C, as indicated in Section 2.2.

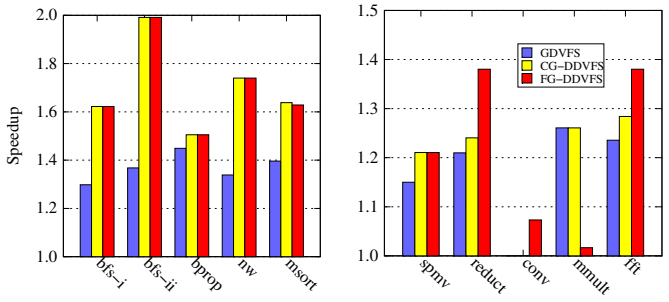


Fig. 1. Benchmark speedups for the DTM algorithms. The benchmarks are grouped into low (left) and high (right) cluster activity. Note that the two groups have different y-axis ranges.

In Fig. 1, we present the benchmark speedups when simulated with the examined DTM techniques. If thermal management is present, the clock frequencies of the clusters and caches can be dynamically scaled up to 1.3GHz. We also assume that ICN frequency can be raised to a maximum of 2.6GHz. A higher interconnect speed is possible due to the simplicity of the pipeline stages in the Mesh-of-Trees ICN (as in [9]). The speedup of a benchmark is calculated using the following formula: $S = Exec_{base}/Exec_{dtm}$, where $Exec_{base}$ and $Exec_{dtm}$ are the execution times on the baseline, *no-DTM* system and with thermal management, respectively.

As a general trend, the benchmarks that benefit the most from the DTM techniques are benchmarks with low cluster activity factors, namely *BFS-I*, *BFS-II*, *Bprop*, *NW* and *Msort* (note the scale difference between the y axes of the two rows of Fig. 1). The highest speedups are observed for *BFS-II* since it has very low overall activity and runs at the maximum clock speed without even nearing the limit temperature. Moreover, CG-DDVFS and FG-DDVFS performs up to 46% better than GDVFS for *BFS-II*, which is mainly bound by memory latency, hence benefits greatly from the independently increased ICN frequency. On the other extreme, *Conv* has the highest cluster activity and was used as the worst case in determining the feasible baseline clock frequency, and therefore shows the least improvement in most experiments.

In the remainder of this section we elaborate on the performance of CG-DVFS and FG-DVFS separately.

CG-DDVFS. Our experiments show that the CG-DDVFS algorithm presents a very reasonable trade-off between hardware complexity and performance. However, performance of certain benchmarks can suffer from the CG-DDVFS algorithm without the modification explained next.

We observed that the performance of the originally proposed CG-DDVFS algorithm is adversely affected by the fact that the ICN and cluster temperatures are correlated. The conduction of the heat generated by the ICN activity adds to the temperature of the clusters, requiring a slowdown in the cluster clock

frequency. Moreover, when the ICN frequency is increased, the cores spend less time idling on memory operations and the cluster power escalates. As a consequence, CG-DDVS is most effective on benchmarks with low cluster activity, and may hurt the performance of a computation bound benchmark by causing the cluster frequency to drop excessively while trying to increase the ICN frequency for an insignificant gain. Therefore, we modified the algorithm to fall back to GDVFS when the ICN activity is lower than the cluster activity (this criterion was determined empirically). Fig. 1 reflects the results of this modification.

The CG-DDVFS technique provides better performance than GDVFS on *BFS-I* (25% faster), *BFS-II* (46% faster), *NW* (30% faster) and *Msort* (17% faster), which have irregular memory accesses and low computation to memory operation ratios. For these benchmarks, the performance bottleneck is the amount of time that TCUs wait on memory operations. CG-DDVFS shortens the wait time by increasing the ICN clock frequency. Irregularity of memory accesses implies that the ICN is not saturated, and it is not fully utilizing its power envelope. Therefore, ICN has sufficient thermal slack that can be picked up by increasing its clock frequency. *Bprop* does not benefit from CG-DDVFS significantly, due to the high degree of queuing on the shared memory modules with this benchmark. We also observed that CG-DDVFS can cause a performance degradation of up to 12% with respect to GDVFS for the remainder of the benchmarks if the GDVFS fall-back is not implemented.
Insight: For a system with a central ICN component such as XMT, workloads that are characterized by scattered irregular memory accesses usually benefit more from dedicated ICN thermal monitoring and control. Conversely, CG-DDVFS can hurt the performance and should be disabled for regular parallel programs which usually have higher computation to memory operation ratios. Performance of CG-DDVFS improves for more advanced cooling solutions.

FG-DDVFS. For the single-tasked system such as the one we examine in this work, the activity does not vary significantly among the clusters when averaged over a sufficiently long time window. The said time window is shorter than the duration required for a significant change in temperature to occur. Therefore, the only benefit that FG-DDVFS can provide is due to the temperature gradient between the middle of the die, where it is harder to remove the heat, and the edges. FG-DDVFS tries to pick up the thermal slack at the edge clusters by increasing their clock frequency. However, the performance of FG-DDVFS suffers from the interaction between the temperatures of the center and edge clusters: as the temperature of the edge clusters rises, so does the temperature of the center clusters, and the controllers in the middle will respond by converging at lower clock frequencies, diminishing the performance gain.

FG-DDVFS exhibits speedups similar to CG-DDVFS on the low activity benchmarks. This is due to the dedicated ICN clock controller introduced in CG-DDVFS. The added value of the dedicated cluster controllers of FG-DDVFS is observed on *Spmv*, *Reduct*, *Conv* and *FFT*, which are the high activity benchmarks. For these benchmarks, runtimes with FG-DDVFS are faster than GDVFS

by 14%, 7% and 11%, respectively. *Mmult* is the only benchmark that shows a slowdown over GDVFS due to its low-P characteristic.
Insight: Individual temperature monitoring and control for computing clusters may be worthwhile even in a single-tasking system with fairly uniform workload distribution. The gains are noteworthy for regular parallel programs with high amounts of computation. Conversely, the overall performance of FG-DDVFS on the low activity benchmarks may not justify its added cost for some systems. It should also be noted that FG-DDVFS has advantage over CG-DDVFS only for cases where the overall speedups are lower than average.

4 Conclusion

In this paper, we outline ongoing work in which we tailor various DTM techniques that exist in uniprocessors and multi-cores to a many-core architecture. We evaluate these techniques on the easy-to-program XMT many-core architecture according to their implementation/design complexity and improvement in single task execution time. We observed that in a many-core processor with fine-grained parallel workloads, the dominant source of thermal imbalance is often between the cores and the interconnection network. According to preliminary results, a DTM technique that exploits this imbalance by individually managing the interconnect can perform up to 46% better than the global DTM for irregular parallel benchmarks. This paper provides several other high-level insights on the effect of individually managing the interconnect and the computing clusters. Our analysis is a step forward from the previous work on multi-core DTM which exclusively focused on systems with a small number of relatively complex serial processors and uneven distribution of the load among these cores. Future work will extend the results presented here to a multi-tasked environment, where the XMT chip will be able to simultaneously execute multiple fine-grained parallel tasks. This extension poses an interesting optimization problem where the management algorithm will also determine the number of threads to be run from each task and how they map on the TCUs.

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

References

1. Balkan, A.O., Horak, M.N., Qu, G., Vishkin, U.: Layout-accurate design and implementation of a high-throughput interconnection network for single-chip parallel processing. In: Proc. Hot Interconnects (2007)
2. Bell, N., Garland, M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors. In: Proc. SC (2009)
3. Caragea, G., Keceli, F., Tzannes, A., Vishkin, U.: General-purpose vs. GPU: Comparison of many-cores on irregular workloads. In: Proc. HotPar (2010)
4. Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Lee, S.H., Skadron, K.: Rodinia: A benchmark suite for heterogeneous computing. In: Proc. IISWC (2009)
5. Donald, J., Martonosi, M.: Techniques for multicore thermal management: Classification and new exploration. In: Proc. ISCA (2006)

6. Ge, Y., Malani, P., Qiu, Q.: Distributed task migration for thermal management in many-core systems. In: Proc. DAC (2010)
7. Ginosar, R.: The plural architecture (2011), www.plurality.com, also see course on Parallel Computing, Electrical Engineering, Technion, http://webee.technion.ac.il/courses/048874
8. Hoberock, J., Bell, N.: Thrust: A parallel template library version 1.1 (2009), http://www.meganewtons.com/
9. Howard, J., Dighe, S., et al.: A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS. In: Proc. ISSCC (2010)
10. Huang, W., Stan, M.R., Sankaranarayanan, K., Ribando, R.J., Skadron, K.: Many-core design from a thermal perspective. In: Proc. DAC (2008)
11. Isci, C., Martonosi, M.: Runtime power monitoring in high-end processors: Methodology and empirical data. In: Proc. MICRO (2003)
12. Kadin, M., Reda, S., Uht, A.: Central vs. distributed dynamic thermal management for multi-core processors: which one is better? In: Proceedings of the Great Lakes Symposium on VLSI (2009)
13. Kaxiras, S., Martonosi, M.: Computer Architecture Techniques for Power Efficiency. Morgan and Claypool Publishers (2008)
14. Keceli, F.: Power and Performance Studies of the Explicit Multi-Threading (XMT) Architecture. Ph.D. thesis, University of Maryland (2011)
15. Keceli, F., Moreshet, T., Vishkin, U.: Power-performance comparison of single-task driven many-cores, submitted for publication
16. Keceli, F., Tzannes, A., Caragea, G., Vishkin, U., Barua, R.: Toolchain for programming, simulating and studying the XMT many-core architecture. In: Proc. HIPS (2011), in conj. with IPDPS
17. Keceli, F., Vishkin, U.: XMTSim: Cycle-accurate Simulator of the XMT Many-Core Architecture. Tech. Rep. UMIACS-TR-2011-02, Univ. of Maryland (2011)
18. Keller, J., Kessler, C., Traeff, J.L.: Practical PRAM Programming. John Wiley & Sons, Inc., New York (2001)
19. Kumar, R., Hinton, G.: A family of 45nm IA processors. In: Proc. ISSCC (2009)
20. Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M., Jouppi, N.P.: McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In: Proc. MICRO (2009)
21. Liu, S., Zhang, J., Wu, Q., Qiu, Q.: Thermal-aware job allocation and scheduling for three dimensional chip multiprocessor. In: Proceedings of the International Symposium on Quality Electronic Design (2010)
22. Ma, K., Li, X., Chen, M., Wang, X.: Scalable power control for many-core architectures running multi-threaded applications. In: Proc. ISCA (2011)
23. NVIDIA: CUDA SDK 2.3 (2009), www.nvidia.com/cuda
24. Padua, D., Vishkin, U.: Joint UIUC/UMD parallel algorithms/ programming course. In: Proc. EduPar (2011), in conj. with IPDPS
25. Patterson, D.: The trouble with multicore: Chipmakers are busy designing microprocessors that most programmers can't handle. IEEE Spectrum (July 2010)
26. Satish, N., Harris, M., Garland, M.: Designing efficient sorting algorithms for manycore GPUs. In: Proc. IPDPS (2009)
27. Skadron, K., Stan, M.R., Huang, W., Velusamy, S., Sankaranarayanan, K., Tarjan, D.: Temperature-aware microarchitecture. In: Proc. ISCA (2003)
28. Wen, X., Vishkin, U.: FPGA-based prototype of a PRAM on-chip processor. In: Proc. Comp. Front. (2008)
29. Wilton, S., Jouppi, N.: CACTI: an enhanced cache access and cycle time model. IEEE J. Solid-State Circuits 31(5), 677–688 (1996)