Multicore vs Manycore: The Energy Cost of Concurrency

Martin Groen and Vincent Gramoli

University of Sydney, Australia
mgro7657@uni.sydney.edu.au,vincent.gramoli@sydney.edu.au

Abstract. In this paper, we study the relation between performance and energy in concurrent programs. As energy efficiency became a key challenge of the computing industry, it is crucial to seek solutions that achieve high performance at a reasonable carbon footprint. We show, however, that energy is dramatically impacted by concurrency and it remains difficult to predict the energy consumed even when the application and the thermal design power are given, due to the number of threads running or their level of contention.
To this end, we evaluated concurrent algorithms on a 2.1 GHz multicore and a 1.2 GHz manycore platforms. Our results show that even though the throughput on manycore is lower than the throughput on multicore, we could not find a single concurrent algorithm where the multicore offers consistently a higher performance per watt than the manycore. More importantly, we identified some benchmarks on which the manycore offers up to 4.3× more operations per second per watt than the multicore.

Keywords: power consumption; energy; manycore; concurrency

1 Introduction

As the complex processing cores require a superlinear growing thermal design power (TDP) to achieve linear performance improvement, increasing concurrency requires to slow down the cores [1]. The limitation stems from the sub-40 nm size of transistors where the Dennard scaling stops applying [2]: the power consumption of a processor is no longer proportional to its area because of current leakage. To lessen the leakage, Intel designed 22 nm Tri-gate transistors, however, this problem will soon limit concurrency by requiring powering off some of the complex cores to let others compute at full speed, a phenomenon known as *dark silicon* [3].

Adopting *manycores*, the concept of placing more simpler cores per chip [4], is thus necessary to keep scaling concurrency within the same power envelope. To understand whether it is worth trying scaling concurrency one has to first answer the question: Does the energy needed to reach some performance on multicore m exceed the energy needed to reach the same performance on manycore M? This question is not simple. On the one hand, m is generally known to have higher energy consumption per unit of time than M [5], but on the other hand m is

also known to run at higher clock frequencies hence executing more instructions than M over time. Manycores have proved instrumental to run concurrent applications at a high performance per watt—examples include key-value stores [6,7]. These applications have, however, been genuinely re-engineered for the considered manycore platform. Hence, it is hard to compare the original multicore implementations to the resulting manycore implementations.

In this paper, we evaluate the performance per watt under concurrency. In particular, we compare the number of benchmark operations executed per second and per watt on a traditional 32-way Intel Xeon *multicore* platform of 32 nm complex cores running at 2.1 GHz and on a less conventional 36-way Tilera TileGx *manycore* platform of 40 nm simpler cores running at 1.2 GHz.

We ported Synchrobench on Tilera to compare the performance obtained on both platforms. Synchrobench is a benchmark suite executing muti-threaded insert/delete/lookup operations to stress-test concurrent data structures using various synchronization techniques [8]. The C/C++ Synchrobench benchmark-suite [8] was originally designed for x86-64 multicores while the Tilera platform provides a manycore architecture with a reduced instruction set and runs a port of the version 3.10 of the Linux kernel and GCC v4.4.6. Unlike previous manycore applications whose multi-threading was carefully re-factorized to run efficiently on Tilera [7], we only ported the Synchrobench benchmark suite with minimal modifications.

One may think that benchmarking performance is sufficient as the energy can be determined with the Thermal Design Power (TDP) provided by its manufacturers. For example, our Intel multicore consumes more power (95 W TDP for 16 hyperthreaded complex cores) than our Tilera manycore (28 W TDP for 36 simple cores). This comparison is not always easy: in particular, multicore manufacturers offer different definitions of TDP [9] and may even provide a *Configurable TDP* (cTDP) that adapts the performance and the energy consumptions at runtime (AMD offers the Turbo Core technology while Intel offers Turbo Boost).

Moreover, as we will show in this paper, the power consumption of a machine is dramatically affected by the concurrent algorithm. The power consumption depends on the number of cores that run and at which clock frequency but it also depends on whether some simultaneous multithreading technology (like hyperthreading) is enabled on these cores. To accurately report these power measurements, we plugged a hardware power meter on our existing multicore and manycore platforms.

As expected, at similar thread count, all applications run significantly faster on the multicore platform than on the manycore platform. Yet, when looking at the performance per watt attained by both machines, our results are surprising: there is no benchmark where the multicore machine achieves consistently higher performance per watt than the manycore. We also observed that there exist benchmarks where the manycore offers significantly higher performance per watt than the multicore. This is interesting as it shows, for the first time, that the power consumption of state-of-the-art algorithms can compensate the

performance advantage of multicores. In other words, even though the highest performance is obtained while running concurrent algorithms on the multicore platform, running them on the manycore platform provides higher performance within the same power envelope.

In Section 2, we present the problem of measuring the performance per watt of concurrent applications on multicore and manycore architectures. In Section 3, we present our manycore and multicore experimental settings. In Section 4, we present the performance and energy consumption of our platforms. In Section 5, we relate the energy consumed and the synchronization technique used. In Section 6, we discuss the related work and in Section 7 we conclude the paper.

2 How to Measure Energy under Concurrency

To evaluate the performance and energy consumption of the manycore platform, we choose the multicore platform as the baseline.

Figure 1 reports the performance and energy consumed by the 32-way multicore platform as observed directly on the power socket when running the lock-free linked list Synchrobench benchmark (Algo.21 [8]) with 64K elements and 10% attempted update, namely the portion of invoked updates (even the ones that return unsuccessfully without writing as described in [8]). The dotted line indicates the throughput T given by Synchrobench when running the benchmark for one minute at different thread count. The bar chart indicates the power consumed in watts E during the experiment as the average over all values read every second on a dedicated power meter (the detailed settings are presented in Section 3). The solid line indicates the *performance per watt* $P = \frac{T*1000}{E}$ *ops per sec*$/W$ as the number of operations per second divided by the watts. The value reported at thread count 0 corresponds to the machine idle, i.e., not running any experiment.

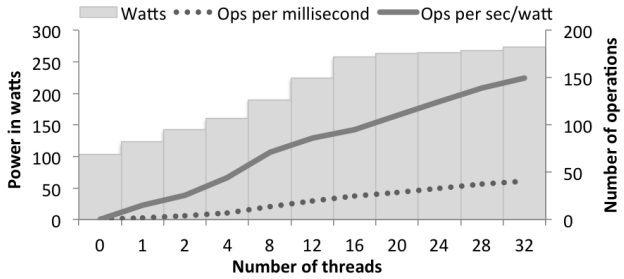

Fig. 1. Power and throughput depending on the level of concurrency

First, we can observe that the power consumption increases with the level of concurrency. The power consumed keeps increasing with the number of threads even when the number of threads exceed the number of cores (16). We can see

however that the power increases faster below 16 threads than above 16 threads. This is due to activation of one new core with each new running thread up to 16: we noted a scattered thread pinning strategy, hyperthreading kicking in after 16 threads. Second, the performance increases as the number of hardware threads used increases, confirming the performance scalability of this particular benchmark on multicore as already noted [8]. Finally, we observe that the performance per watt increases also steadily up to the highest hardware thread count, indicating that the multicore machine delivers an energy proportional computation [10] on this particular benchmark. This is not always the case as the performance of several algorithms does not necessarily increase to the highest hardware thread count, as explained in Section 4.

As in Figure 1, we carefully observed that the highest performance per watt for a given workload on both the mutlicore and the manycore platforms was always obtained at the thread count where the performance was the highest. In other words, the energetic overhead is never higher than the performance drop. Hence, in the remainder of the paper and when not explicitly mentioned, we report the performance per watt observed at the thread count that maximizes performance.

3 Energy and Concurrency Settings

In this section, we present the multicore and manycore platforms, the power monitoring tools and the algorithms used. The multicore and the manycore platforms are both 64-bit platforms made available for purchase in 2012. The multicore machine is a 32 nm Xeon platform based on Intel's x86 architecture with a complex instruction set whereas the manycore is a TILExtreme platform with 40 nm manycore Tile-Gx processors based on the Tilera architecture with reduced instruction set. These two platforms use a 3-level cache.

Platform	Description	CPU	#CPU	#cores per CPU	#hw threads	clock frequency	CPU TDP	Power (idle)	Power (loaded)
M	TILExtreme	Tile-Gx	4	36	144	1.2 GHz	28 W	240 W	294 W
m	SandyBridge	Intel Xeon	2	8	32	2.1 GHz	95 W	103 W	277 W

Table 1. The specification summary of our manycore (M) and multicore (m) platforms

Multicore. The multicore machine is a SandyBridge-EN of 2 8-core Intel Xeon E5-2450 offering a total of 32 hardware threads with hyperthreading enabled running at 2.1 GHz but that could be overclocked at 2.9 GHz with enabled TurboBoost [11]. Intel offers Turbo Boost 2.0 so that processors may "operate at a power level that is higher than its TDP configuration".[1] Note that this ap-

[1] http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html.

proach is shared by other multicore manufacturers: AMD proposes the Turbo Core technology to increase similarly the core frequency within the thermal and power limits of the accelerated processing unit.[2] Each processor has a TDP of 95 W.[3] It features transistors of size 32 nm.

Manycore. The manycore machine is a TILExtreme, a four 36-core Tile-Gx processors running at 1.2GHz. It features 16 fans that run at a speed of 3000 to 16000 rpm that cannot be disabled or tuned individually [12]. The details are summarized in Table 1. There is no coherence across two different TileGx sockets. The 36 cores of each Tile-Gx are organized into a 6×6 mesh of tiles where each cache line has a dedicated "home" core. Upon level-2 local cache miss, a core request the cache line from the home core local level-2 cache so that the union of all home core level-2 caches represents an 9 MiB level-3 cache. The cache coherence is maintained through a distributed directory that is more energy efficient than a bus-snooping cache coherency protocol.

3.1 Preliminary Power Measurements

We measured the performance of our platforms using a power metering tool.

Watt metering. We used the Watts Up? .NET watt meter 100-250 V, 50/60 Hz and 15 amps to perform our power measurements. This device has an accuracy of $\pm 1.5\%$ when reporting consumption above 60 W like in our case. Note that the same device was previously used to report power consumption in other studies [6]. All power measurements were collected for both the muticore and manycore machines in the same room with a steady temperature of 20.8°C cooled using an independent air conditioning system whose power consumption was not accounted in our measurements.

Power consumption under full load. The power consumption at full load was measured with the Synchrobench lock-free skip list running with parameters u10-i65536-r132K-d60000[4] with the number of threads set to the maximum number of hardware threads available. Because the fans cannot be disabled and tuned individually on the Tilera [12], we run the full load on the four sockets of the Tilera (144 cores) and divided the energy consumption by four to get an estimate of the energy consumed per socket. It is important to remark that a single socket machine could consume more than a fourth of this overall power due to the consumption of components shared by the four sockets. To confirm that the shared consumption was not impacting our results, we measured the power consumption of the machine with all the sockets shut-down in hardware

[2] http://www.amd.com/en-us/innovations/software-technologies/turbo-core.
[3] http://ark.intel.com/products/64611/Intel-Xeon-Processor-E5-2450-20M-Cache-2_10-GHz-8_00-GTs-Intel-QPI.
[4] 10% update, initial size of 65536, value range of 132K and a duration of 60 seconds [8].

and observed 87 watts. We then confirmed that manycore would still reach higher performance per watt than multicore even if each fan consumed less than 0.8 W on this heavy workload. We selected the multicore platform as the baseline for our experiments as it is the most common platform of the two. We noticed that the power consumption of this platform when idle was 103 watts, which is close to the 95 W TDP announced by the manufacturer.

	Data Structure	Ref.	Synchronization
skip lists	Optimistic skip list	[13]	`spin-lock`, `mutex`
	Rotating skip list	[14]	`lockfree CAS`
	Elastic skip list	[15]	`ESTM`
	Fraser skip list	[16]	`lockfree CAS`
	No hot spot skip list	[17]	`lockfree CAS`
	Sequential skip list	[8]	∅
linked lists	Lazy linked list	[18]	`spin-lock`, `mutex`
	Harris's linked list	[19]	`lockfree CAS`
	Reusable linked list	[20]	`ESTM`
	Lock-coupling linked list	[21]	`spin-lock`, `mutex`
	Sequential linked list	[8]	∅
hash tables	Lock-free hash table	[22]	`lockfree CAS`
	Elastic hash table	[15]	`ESTM`
	Sequential hash table	[8]	∅
binary trees	Speculation-friendly tree	[23]	`ESTM`
	Transactional red-black tree	[24]	`ESTM`
	Sequential binary search tree	[8]	∅

Table 2. Port of Synchrobench-C/C++ to the Tilera manycore

3.2 Porting Synchrobench-C/C++ to Manycore

To understand whether the concurrent programs and the synchronization techniques impact energy efficiency, we run the Synchrobench [8] benchmark suite on both the multicore and the manycore machine. Synchrobench is a benchmark suite designed to evaluate the performance of synchronization techniques like compare-and-swap (CAS), spin-lock, mutex and transactional memory (TM), and data structure implementations on multicore machines.

To evaluate the performance on manycore, we ported 17 benchmarks out of the 19 C/C++ benchmarks of Synchrobench-v1.1.0-alpha to the Tilera architecture. We also ported the TM library implementing elastic transactions, ESTM [25]. We restricted our study to C/C++ because the only other available version of Synchrobench is in Java[5] and we know that the experimental measurements are more predictable than in Java especially when running different JVMs [8].[6]

The oldest benchmarks of Synchrobench used the `atomic_ops` library from HP[7], however, this library supports only IA32 and x86-64 and was adapted for

[5] https://sites.google.com/site/synchrobench/download.
[6] The TILExtreme runs a port of Java 1.6.
[7] https://github.com/ivmai/libatomic_ops/.

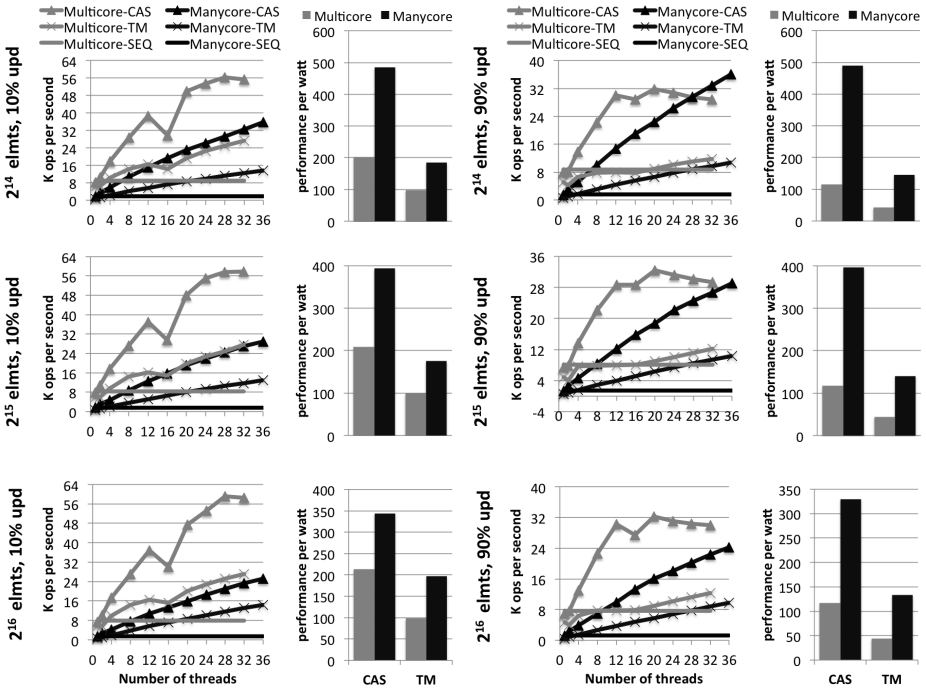

Fig. 2. Operations per second per watt of multicores and manycores running the concurrent hash tables benchmark

SPARC but not for Tilera—we had to manually port some of its operations to the Tilera architecture, as listed in Table 2. Some other benchmarks rely on the recent C/C++11 atomic intrinsics, however, as of today neither `stdatomic` nor the latest versions of GCC are supported on Tilera.[8] We decided not to port the remaining benchmarks because of the changes they would induce: some of the benchmarks of Synchrobench were only designed to run on 64-bit Intel and featured a 128-bit wide compare-and-swap that does not exist on Tilera [26] and adapting them would have affected the veracity of our comparisons on different architectures.

4 The Energy of Multicore and Manycore

In this section, we show that with their lower clock frequencies, manycore can have a substantially higher performance per watt than multicore in different workloads. We also show that there is no single benchmark where multicore can provide higher performance per watt than manycore across all synchronization techniques.

[8] http://www.tilera.com/scm/.

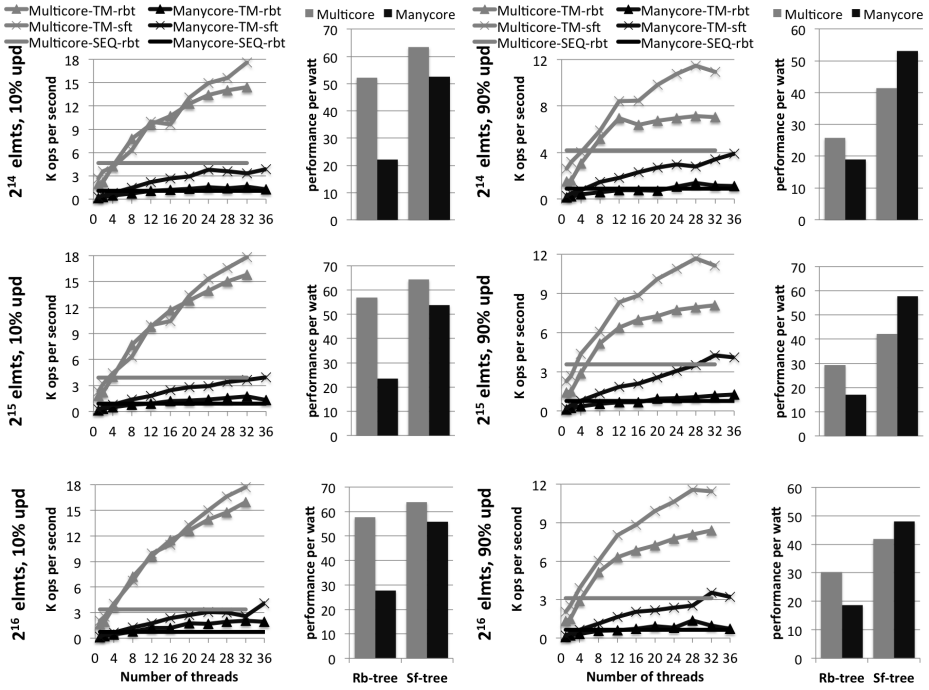

Fig. 3. Operations per second and per watt of multicore and manycore running the concurrent binary search trees benchmark

Raw performance. Figure 2 (resp. Figure 3) represents the performance achieved and the energy dissipated when running the hash table (resp. binary search tree) benchmarks of Synchrobench-C/C++ on our multicore and manycore platforms. In these figures, each binary tree or hash table implementation is synchronized with compare-and-swap (denoted by CAS), transactional memory (denoted by TM) or nothing being only able to run sequentially (denoted by SEQ). The different binary search trees algorithms are either of type red-black tree (denoted rbt) or of type speculation-friendly tree [27] (denoted sft). Both figures indicate that concurrent algorithms perform generally better on the multicore than on the manycore. This is expected given the lower clock frequency of the mancycore machine (1.2 GHz) compared to the multicore machine (2.1 GHz). However, we can also see that the performance of the manycore can be higher than the one of the multicore in some cases (cf. top-right of Figure 2). This is due to the contention that induces cross-socket communication on the multicore and that does simply require low-latency network-on-chip communication on the manycore.

Higher performance per watt for the manycore. The hash table benchmark (Figure 2) clearly shows higher performance per watt on the manycore than on the multicore (across different sizes and update ratios). In particular, on 90% updates and with 2^{12} elements, the hash table benchmark runs 4.3×

more operations per watt than the multicore at maximum thread counts. Note that the speedup is 3.9× when the thread count is 32 on both machines. The reason is probably due to the low contention of hash table and the fact that the manycore platform have high speed core-to-core communication compared to the multicore machine. In addition, the time needed for a core to access the memory or the level-1 cache on the manycore are faster than on the multicore. For example accessing the level-1 cache of the Tilera takes 1.7 μsec (2 cycles at 1.2 GHz) while it takes 2.4 μsec (5 cycles at 2.1 GHz) on the Xeon. For other data structures, whether the multicore or the manycore is more suitable depends on many parameters, like the synchronization technique used to synchronize the data structure, the level of contention and the size of the data structures. We discuss the impact of the synchronization technique used in Section 5.

5 The Energy of Synchronization Techniques

To get a broader view of the performance per watt delivered by the manycore and the multicore, we ran the other Synchrobench benchmarks.

Figure 4 depicts the performance per watt obtained on the multicore and the manycore for the Harris linked list that uses CAS for synchronization. We used this benchmark as an example to illustrate that both manycore and multicore can achieve better performance per Watt results at different thread counts. For clarity and given that we had only 32 hardware contexts on the multicore, we did not represent the performance obtained on the manycore at 36 threads. The other Synchrobench parameters used for this benchmark are `-u10-i16384-r32768`, indicating an initial size of 2^{14} elements and an attempted update ratio of 10%.

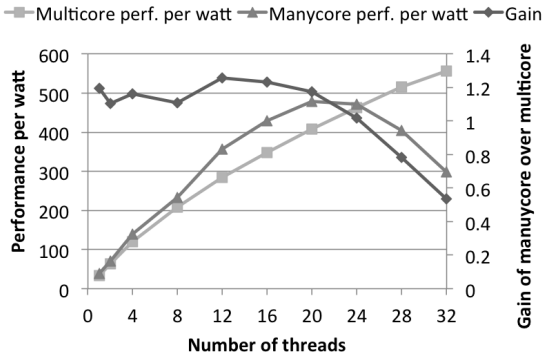

Fig. 4. Performance per watt improvement of the manycore over the multicore

First, we can observe that the performance per watt delivered by the manycore does not scale up to 32 threads (triangle-dotted line) while the one delivered by the multicore scales with the level of concurrency (square-dotted line). In addition, the peak performance per watt delivered by the manycore is comparatively higher than the peak performance per watt delivered by the multicore. This indicates that the multicore presents some advantage in terms of performance per watt for this particular benchmark. Finally, we observe that the performance per watt delivered by the multicore is, however, not consistently higher than the one delivered by the manycore. In particular, between 1 and 24 threads, the per-

formance per watt obtained from the multicore is higher than the one obtained from the multicore. Although not depicted here, we ran additional experiments and identified some skip list benchmarks with similar differences: the peak performance per watt is higher on manycore whereas, at some thread counts, the manycore delivers higher performance per watt.

We conclude that the multicore does not consistently provide a higher performance per watt than the manycore on a given data structures. This is in contrast with the manycore offering consistently higher performance per watt than the multicore on all the binary search trees evaluated, whether they were synchronized with CAS, TM or simple running sequentially. We also observed, however, that this is not necessarily true when considering a data structure benchmark synchronized with a particular technique. Hence, we observed that the multicore would deliver a higher performance per watt than the manycore on the skip list synchronized with CAS or TM and on the linked list synchronized with TM, but not on the skip list synchronized with locks, the linked list synchronized with CAS and the linked list synchronized with locks.

6 Related Work

A study on the impact of concurrency on power consumption [28] shows that running two cores instead of one could, on some workloads, double the power overhead and that simultaneous multi-threading could save energy on recent hardware and in-order processors. The focus of this study is on managed languages showing, for example, how seeminlgy singly-threaded Java applications actually exploit multiple cores through the JVM.

Some research work focuses on algorithms to model theoretically their carbon footprint [29, 30]. The first study [30] shows that for matrix multiplication and the n-body problem, the energy consumption remains constant as the number of processors increases and the runtime decreases. The second study raised the question of the relevance of designing algorithms under the constraint of energy efficiency [29]. It does not present experimental measurements but rather exploits energy models applied to graphical processing units. These studies do not model the energy consumed by non-deterministic executions.

A recent work simulated the impact of the MSI cache coherence protocol on the energy consumed by data structure algorithms that experience non-deterministic executions [31]. The authors propose new lease and release instructions to minimize cache invalidation in lock-based and lock-free structures. Simulations of their instructions on the Graphite multi-processor simulator indicate a substantial reduction of the energy consumption, compared to the classic MSI cache coherence protocol without lease/release.

The energy consumption of both simple and complex cores was modelled in the context of distributed heterogeneous platforms [32]. To validate their results, the authors measure the consumption of high performance computing applications on clusters of Intel Xeon and ARM Cortex-A9 nodes. FAWN [6] is an in-memory key-value store well-tuned for running on 21 single-threaded winpy

nodes using flash storage to retrieve data that cannot fit in memory. FAWN achieves a peak 350 key-value queries per Joule. With 21 nodes, FAWN achieves 350 key-value queries per Joules. As the goal of our study was to compare concurrent programs running on multicore and manycore platforms released the same year, we minimized the changes of our benchmarks while porting them to manycores.

7 Conclusion

We measured the performance and energy consumption of a multicore and a manycore when running concurrent algorithms. As expected, these algorithms run faster on the multicore but can achieve better performance per watt on the manycore. There are several directions for future work. First, it would be interesting to isolate the power consumption of each individual components, like fans and CPUs, by separating them physically or by using dedicated software toolsets. Second, it would be interesting to broaden the scope of benchmarks to see whether the same results hold for IO-bound applications.

Acknowledgments. This research was supported under Australian Research Council's Discovery Projects funding scheme (project number 160104801) entitled "Data Structures for Multi-Core". Vincent Gramoli is the recipient of the Australian Research Council Discovery International Award.

References

1. Esmaeilzadeh, H., Blem, E., Amant, R.S., Sankaralingam, K., Burger, D.: Power challenges may end the multicore era. Commun. ACM **56**(2) (February 2013) 93–102
2. Dennard, R.H., Rideout, V., Bassous, E., Leblanc, A.: Design of ion-implanted mosfet's with very small physical dimensions. IEEE Journal of Solid-State Circuits **9**(5) (1974) 256–268
3. Esmaeilzadeh, H., Blem, E., St.Amant, R., Sankaralingam, K., Burger, D.: Dark silicon and the end of multicore scaling. In: ISCA. (June 2011) 365–376
4. Borkar, S., Chien, A.A.: The future of microprocessors. Commun. ACM **54**(5) (May 2011) 67–77
5. Borkar, S.: Thousand core chips: a technology perspective. In: DAC. (2007) 746–749
6. Andersen, D.G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., Vasudevan, V.: Fawn: A fast array of wimpy nodes. In: SOSP. (2009) 1–14
7. Berezecki, M., Frachtenberg, E., Paleczny, M., Steele, K.: Many-core key-value store. In: IGCC. (July 2011) 1–8
8. Gramoli, V.: More than you ever wanted to know about synchronization: Synchrobench, measuring the impact of the synchronization on concurrent algorithms. In: PPoPP. (2015) 1–10
9. Intel: Measuring power processor: TDP vs. ACP (2011) Intel White paper.

10. Barroso, L.A., Holzle, U.: The case for energy-proportional computing. Computer **40**(12) (2007) 33–37
11. Rotem, E., Naveh, A., Rajwan, D., Ananthakrishnan, A., Weissmann, E.: Power management architecture of the 2nd generation intel core microarchitecture, formerly codenamed sandy bridge. In: HotChips. (2011)
12. Tilera: UG410 - TILExtreme-Gx Platform User's Guide (May 2013) Release 1.1 Doc. N. UG410.
13. Herlihy, M., Lev, Y., Luchangco, V., Shavit, N.: A simple optimistic skiplist algorithm. In: SIROCCO. (2007) 124–138
14. Dick, I., Fekete, A., Gramoli, V.: A skip list for multicore. Concurrency and Computation, Practice and Experience (May 2016)
15. Felber, P., Gramoli, V., Guerraoui, R.: Elastic transactions. In: DISC. Springer (2009) 93–107
16. Fraser, K.: Practical lock-freedom. PhD thesis, University of Cambridge (2004)
17. Crain, T., Gramoli, V., Raynal, M.: No hot spot non-blocking skip list. In: ICDCS. (2013) 196–205
18. Hellor, S., Herlihy, M., Luchangco, V., Moir, M., Scherer, W.N., Shavit, N.: A lazy concurrent list-based set algorithm. Parallel Processing Letters **17**(04) (2007) 411–424
19. Harris, T.L.: A pragmatic implementation of non-blocking linked-lists. In: DISC. (2001) 300–314
20. Gramoli, V., Guerraoui, R.: Reusable concurrent data types. In: ECOOP. Volume 8586 of LNCS. (2014) 182–206
21. Herlihy, M., Shavit, N.: The Art of Multiprocessor Programming. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2008)
22. Michael, M.M.: High performance dynamic lock-free hash tables and list-based sets. In: SPAA, New York, NY, USA, ACM (2002) 73–82
23. Crain, T., Gramoli, V., Raynal, M.: A contention-friendly binary search tree. In: Euro-Par. Springer (2013) 229–240
24. Minh, C.C., Chung, J., Kozyrakis, C., Olukotun, K.: Stamp: Stanford transactional applications for multi-processing. In: IISWC, IEEE (2008) 35–46
25. Felber, P., Gramoli, V., Guerraoui, R.: Elastic transactions. In: DISC. (2009) 93–107
26. Intel: Intel 64 and IA-32 architectures software developers manual – volume 2A: Instruction set reference, A-M (2007)
27. Crain, T., Gramoli, V., Raynal, M.: A speculation-friendly binary search tree. In: PPoPP. (2012) 161–170
28. Esmaeilzadeh, H., Cao, T., Yang, X., Blackburn, S.M., McKinley, K.S.: Looking back and looking forward: Power, performance, and upheaval. Commun. ACM **55**(7) (July 2012) 105–114
29. Choi, J.W., Vuduc, R.W.: How much (execution) time and energy does my algorithm cost? XRDS **19**(3) (March 2013) 49–51
30. Demmel, J., Gearhart, A., Lipshitz, B., Schwartz, O.: Perfect strong scaling using no additional energy. In: IPDPS. (2013) 649–660
31. Haider, S.K., Hasenplaugh, W., Alistarh, D.: Lease/release: Architectural support for scaling contended data structures. In: PPoPP. (2016)
32. Ramapantulu, L., Loghin, D., Teo, Y.M.: An approach for energy efficient execution of hybrid parallel programs. In: IPDPS. (May 2015) 1000–1009