Improving Energy Efficiency On Asymmetric Multiprocessing Systems

Paul E. McKenney
Linux Technology Center
IBM Beaverton

Dietmar Eggemann
ARM Ltd. Cambridge

Robin Randhawa
ARM Ltd. Cambridge

Abstract

As battery-powered embedded devices move towards multicore processors, multicore energy efficiency is becoming critically important. To this end, ARM recently announced its big.LITTLE architecture, featuring a multicore chip with both high-performance processors and high-efficiency processors running a single instance of the Linux operating system. Prior work has shown the benefits of running performance-critical code on the high-performance processors, while confining other processing to the high-efficiency processors. This paper builds on this work by showing that for some important mobile workloads, pre-existing Linux-kernel tuning parameters originally designed for real time, high-performance computing, and SMP energy efficiency can further reduce the amount of non-performance-critical code running on the high-performance processors, resulting in energy efficiency gains in excess of 10%.

1 Introduction

Battery lifetime is crucially important to battery-powered devices such as smartphones, including multicore smartphones. These devices run workloads with very low average CPU utilization, but often experience short bursts of high-utilization work required for good user experience [17]. One approach for these workloads is asymmetric multiprocessing, as discussed in the next section.

2 Big.LITTLE

The big.LITTLE architecture is an asymmetric multiprocessing architecture developed by ARM, Ltd. [7].

Here, the high-performance processors are Cortex-A15s and the high-efficiency processors are the Cortex-A7s. The Cortex-A15s run about twice as fast as the Cortex-A7s, which in turn run instructions about three times as efficiently as do the Cortex-A15s. As noted earlier, this configuration suggests running non-performance-critical code on the Cortex-A7s.

Unfortunately, performance-critical code might invoke non-performance-critical deferred cleanup operations. One example of such an operation is deferred work from read-copy update (RCU), which is overviewed in the next section.

3 Read-Copy Update

Read-copy update (RCU) is a synchronization mechanism [13] that is often used as a replacement for reader-writer locking for read-mostly data structures in the Linux kernel (see Figure 2) [11], and has more recently been implemented in user space as well [3].

RCU coordinates updaters with readers executing in *RCU read-side critical sections* demarked by `rcu_read_lock()` and `rcu_read_unlock()`. RCU does not coordinate among updaters, which must use some other synchronization mechanism (e.g., locking). One major advantage of RCU is that its read-side primitives are exceedingly fast, and can in fact incur zero overhead in real-world usage. This advantage in turn implies a high degree of freedom from deadlock, and also implies that updaters can in no way exclude, block, or delay readers. Updaters must therefore carry out their updates with care, by preventing future readers from accessing a given data element, then waiting for all pre-existing readers to complete. Sections 3.1 and 3.2 show examples for insertion and deletion, respectively.

3.1 RCU Element Insertion

The four states shown in Figure 3, with time advancing from top to bottom, demonstrate how a new element may be referenced by an initially `NULL` global pointer named `gptr` despite the fact that there are concurrent readers. We start out in state 1 with the `NULL` `gptr`, and then use

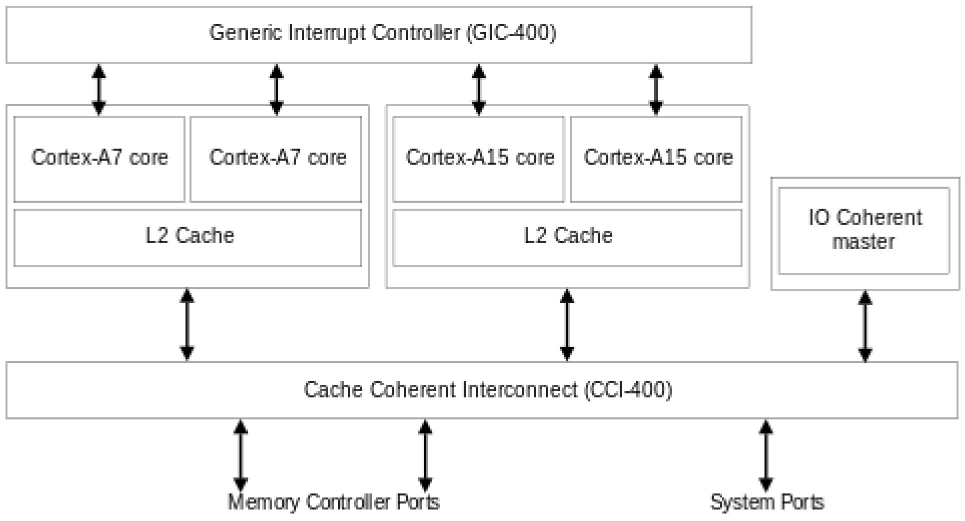

Figure 1: ARM big.LITTLE Schematic

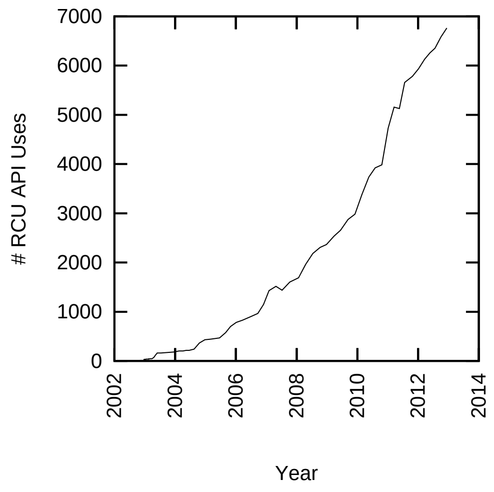

Figure 2: RCU Usage in the Linux Kernel

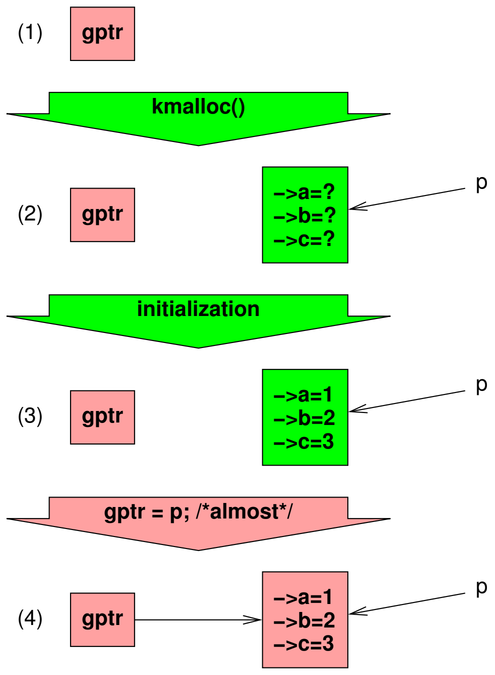

Figure 3: Insertion With Concurrent Readers

`kmalloc()` to allocate a new structure with indeterminate contents, transitioning to state 2. The pointer is colored red to indicate that any RCU reader might access it at any time, and the new structure is colored green to indicate that it is inaccessible to readers. We transition to state 3 by initializing the new structure and to state 4 by storing a pointer to it in `gptr`, so that the new structure is now accessible to readers (and thus colored red). This store operation relies on atomicity, in other words, concurrent readers can see either the old `NULL` pointer or the pointer to the new structure, but they must not see a "mash-up" of the two pointers. Old versions of C/C++ may use volatile casts and memory barriers to implement this atomic store, while new C11/C++11 compilers can use release operations on atomic variables.

This example illustrates how updaters can insert data into linked structures despite concurrent readers.

3.2 RCU List Deletion

Figure 4 shows a four-state process for deleting from an RCU-protected linked list. In state 1, we have a list containing elements A, B, and C, all of which might be ac-

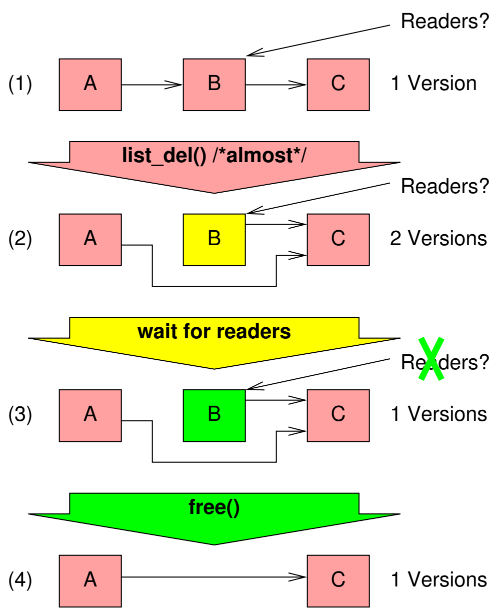

Figure 4: Deletion From Linked List With Concurrent Readers

cessed by a concurrent reader at any time. We transition to state 2 by deleting element B, at which point new readers no longer have a path to it, so that only pre-existing readers can be referencing element B, which is now colored yellow. Note that the path from element B to element C remains intact, enabling readers that are still referencing B to advance through the remainder of the list. We transition to state 3 by waiting for all pre-existing readers, after which there can no longer be any readers referencing element B, which is now colored green. At this point, it is safe to transition to state 4 by freeing element B.

Given the ability to insert into and remove from linked structures, we can see that RCU can be used to carry out a wide range of updates to a wide range of linked data structures, without forcing readers to execute expensive operations. However, the wait-for-readers operation has energy-efficiency consequences that are discussed in the next section.

3.3 RCU and Energy Efficiency

RCU operates via a state machine driven by the scheduling-clock interrupt. This means that if an otherwise idle processor has one or more wait-for-readers operations pending, its scheduling-clock interrupt will continue until all such operations have completed. This series of scheduling-clock interrupts will result in a closely spaced series of wakeups from idle, which will in turn result in increased energy consumption. This paper looks at two methods for reducing these wakeups, which are covered in the next section.

4 Reducing RCU-Induced Wakeups

The first method for reducing RCU-induced wakeups is to reduce the frequency of invocation of the RCU state machine during idle periods, thereby enforcing longer idle durations. This is accomplished by allowing the scheduling-clock interrupt to be disabled during idle periods even when the processor in question has wait-for-readers operations pending, and substituting a timer with a period that is four times longer than that of the scheduling-clock interrupt. This method is enabled in recent Linux kernels using `CONFIG_RCU_FAST_NO_HZ=y`. Interestingly enough, this method was designed to improve energy efficiency not in asymmetric multiprocessors, but rather in symmetric multiprocessors.

The second method for reducing RCU-induced wakeups is to offload RCU processing from the Cortex-A15 processors using `CONFIG_RCU_NOCB_CPU=y`. Of course, this processing has to happen somewhere, and where it does happen it will consume energy, but given that the Cortex-A7 processors are three times as energy-efficient as are the Cortex-A15 processors, offloading from the Cortex-A15 processors should improve energy efficiency. This second method was designed to reduce OS jitter for compute-intensive applications and to improve real-time response. It was never intended to improve energy efficiency, much less on asymmetric multiprocessors.

Both methods were implemented by one of the authors (Paul), and both are now available in the Linux kernel.

The next section compares these two approaches.

5 Measured Performance

We compared the RCU-processing-offload and enforced-idle approaches to a reference kernel with stock RCU and idle settings. All of the tests included Morten Rasmussen's big.LITTLE scheduler modifications [15]. We used six different benchmarks, each of which performs a fixed amount of work. We therefore measured the energy consumed (in Joules) and the duration of each run (in seconds), taking the mean value from five runs of each combination of benchmark and software configuration. We ran these benchmarks on a CoreTile Express A15x2 A7x3 [1], which is an ARM big.LITTLE system that is popularly referred to as TC2.

The benchmarks are as follows:

- cyclictest: This benchmark sets a series of timers and measures the resulting timer jitter [18]. Although this benchmark was intended to test real-time systems, it provides a good example of a low-utilization workload.

- sysbench: This benchmark is intended to test a system's ability to support database workloads [9]. We included it in order to verify overall performance.
- andebench: This benchmark was developed by The Embedded Microprocessor Benchmark Consortium to evaluate Android smartphones [17]. We ran this in two-thread (andebench2) and eight-thread (andebench8) configurations.
- audio: This benchmark emulates audio playback.
- audio+bbench: This benchmark combines audio playback with the bbench web rendering benchmark [8].

These benchmarks ran on real hardware, and the energy consumption was directly measured.

The results are shown in Table 1. The "Reference" columns show the mean energy (in Joules) and test time (in seconds) for five runs of each benchmark using the reference kernel, that is, with neither RCU processing offload nor enforced idle. The boldfaced entries in the other columns highlight cases where the range of values overlap with their reference counterparts. This highlights a limitation of this data: System energy consumption can vary with a number of factors, most notably increasing with the temperature of the chip electronics. We accounted for this effect by running the system for ten minutes in order to bring the temperature up to its steady-state value before running any benchmarks. Note that all of the test-duration measurements show overlap, which indicates that the energy-conservation methods do not unambiguously degrade performance. The raw data may be found in the accompanying technical report [12].

The "RCU Processing Offload" columns show the same data for a kernel built to offload RCU processing from the Cortex-A15 processors via the `CONFIG_RCU_NOCB_CPU=y` kernel parameter, along with the percentage improvement over over the reference kernel. The improvements in energy efficiency are substantial, especially for cyclictest and the audio+bbench benchmarks. The measurements for the andebench8 and audio benchmarks overlap with those from the reference runs, so we rejected those values as insufficiently different.

The "Enforced Idle" columns again show energy, duration, and improvement data, but for a kernel built to enforce idle periods via the `CONFIG_RCU_FAST_NO_HZ=y` kernel parameter. This approach provides even better energy efficiency, with several benchmarks showing double-digit improvements compared to the reference kernel (though the cyclictest and andebench2 results are insufficiently different than those of the reference run), but also showing greater test-duration increases for a number of the benchmarks. These duration increases are likely due to the fact that enforcing idle increases the time spend waiting for readers, however, we rejected all of the duration measurements as insufficiently different from the reference runs.

Both approaches show substantial energy-efficiency improvements, which raises the question as to whether combining them would further improve energy efficiency. Unfortunately, preliminary experiments indicate that combining these approaches does no better than each can do separately. This should not come as a surprise, given that both act to reduce busy periods on the Cortex-A15 processors, and that it is not possible to remove the same busy period twice. Furthermore, running both methods incurs their combined overhead, which can actually degrade energy efficiency. Determining which of these two methods is better is future work.

6 Related Work

Energy efficiency in asymmetric multiprocessors has been a topic of research for some time. Kumar et al. [10] proposed a feedback approach to dynamically migrate processing among four different DEC Alpha processors, ranging from EV4 to EV8. This feedback approach takes samples every 100 million instructions, and resulted in significant energy savings, but at the cost of significant performance degradation. Furthermore, the energy savings were modeled rather than measured.

Grant and Afsahi [5, 6] use a modified scheduler to identify and offload non-performance-critical housekeeping work to separate low-frequency CPUs in CPU-intensive numerical applications. This offloading reduces OS jitter, thus in some cases improving performance while also reducing energy consumption. Offloading non-performance-critical housekeeping work is now standard practice for this class of application.

Fedorova et al. [4] describe a scheduler that distinguishes between sequential and parallel application-execution phases via the number of runnable threads: An application with only a single runnable thread is considered to be sequential, and is thus run on the fast CPU, while applications with many runnable threads are assumed to be parallel, and are thus run on the slow CPUs. Slow CPUs are emulated by reducing the clock frequency on selected CPUs in an SMP system. They also describe methods of detecting memory and scalability bottlenecks, but do not directly measure energy consumption.

In addition, there have been a number of analytic calculations of and simulations of performance and energy efficiency for asymmetric multiprocessors [14, 16].

	Reference		RCU Processing Offload				Enforced Idle			
Benchmark	E (J)	T (s)	Energy	Benefit	Time	Benefit	Energy	Benefit	Time	Benefit
cyclictest	1.75	3.13	1.47	15.98%	3.23	**-3.38%**	1.47	**16.13%**	3.13	**-0.07%**
sysbench	32.68	9.14	31.61	3.29%	8.99	**1.68%**	31.12	4.77%	8.99	**1.70%**
andebench2	59.51	20.54	57.91	2.70%	20.37	**0.83%**	57.99	**2.57%**	21.59	**-5.09%**
andebench8	174.04	45.87	170.37	**2.11%**	46.19	**-0.70%**	166.91	4.09%	46.60	**-1.59%**
audio	7.87	30.01	7.39	**6.05%**	30.03	**-0.04%**	6.28	20.16%	30.02	**-0.04%**
audio+bbench	99.05	156.29	93.02	6.09%	155.58	**0.45%**	86.95	12.21%	160.75	**-2.85%**

Table 1: Measured Energy Consumption and Performance on TC2

All the preceding work involves CPU-intensive workloads, and thus does not necessarily apply to the low-utilization workloads found in the battery-powered embedded arena. Morten Rasmussen [15] filled this gap with a study demonstrating performance and energy-efficiency improvements for low-utilization workloads. This work combines a priori knowledge of properties of software making up the Android system with Paul Turner's per-entity load-tracking enhancement to the Linux scheduler [2]. This combination results in substantial improvements in performance and energy efficiency. Despite views to the contrary [16], a priori knowledge is practical and valuable in the specialized embedded arena.

However, Morten's work does not address tiny units of deferred work that are the subject of this paper.

7 Conclusions and Future Directions

We have shown that enforced idle (`CONFIG_RCU_FAST_NO_HZ=y` kernel parameter) is effective at reducing energy consumption, but that it might also reduce performance by increasing RCU's grace-period interval. We have also shown that RCU processing offload (`CONFIG_RCU_NOCB_CPU=y` kernel parameter) is also effective at reducing energy consumption, but with no perceptible performance reduction. Preliminary results show that combining enforced idle and RCU processing offload do not result in further improvements in energy efficiency. Both enforced idle and RCU processing offload are generally useful improvements that also happen to benefit energy efficiency on asymmetric multiprocessors.

We hypothesize that energy efficiency would be further improved if other deferred-work mechanisms (including timers, work queues, inter-processor interrupts, and softirq handlers) were to offload their work from the high-performance Cortex-A15 processors onto high-efficiency Cortex-A7 processors. We further hypothesize that the same is true of equivalent mechanisms in user-level applications and utilities. Regardless of whether or not these hypotheses hold true, we have demonstrated that significant energy-efficiency improvements are possible for asymmetric computer systems given modest modifications to the software running on them. Such improvements will become increasingly important as battery-powered devices move to multicore configurations.

Acknowledgements

We are indebted to Jim Wasko for his support of this effort.

Legal Statement

References

[1] ARM LTD. CoreTile Express A15x2 A7x3 technical reference manual. `http://www.arm.com/files/pdf/DDI0503B_v2p_ca15_a7_reference_manual.pdf`, July 2012.

[2] CORBET, J. Per-entity load tracking. `http://lwn.net/Articles/531853/`, January 2013.

[3] DESNOYERS, M., MCKENNEY, P. E., STERN, A., DAGENAIS, M. R., AND WALPOLE, J. User-level implementations of read-copy update. *IEEE Transactions on Parallel and Distributed Systems 23* (2012), 375–382.

[4] FEDOROVA, A., SAEZ, J. C., SHELEPOV, D., AND PRIETO, M. Maximizing power efficiency with asymmetric multicore systems. *Commun. ACM 52*, 12 (Dec. 2009), 48–57.

[5] GRANT, R. E., AND AFSAHI, A. Power-performance efficiency of asymmetric multiprocessors for multi-threaded scientific applications. In *Proceedings of the 20th international conference on Parallel and distributed processing* (Washington, DC, USA, 2006), IPDPS'06, IEEE Computer Society, pp. 300–300.

[6] GRANT, R. E., AND AFSAHI, A. Improving energy efficiency of asymmetric chip multithreaded multiprocessors through reduced os noise scheduling. *Concurr. Comput. : Pract. Exper. 21*, 18 (Dec. 2009), 2355–2376.

[7] GREENHALGH, P. Big.LITTLE processing with ARM Cortex-A15 and Cortex-A7. `http://www.arm.com/files/downloads/big.LITTLE_Final.pdf`, September 2011.

[8] GUTIERREZ, A., DRESLINSKI, R. G., WENISCH, T. F., MUDGE, T., SAIDI, A., EMMONS, C., AND PAVER, N. Full-system analysis and characterization of interactive smartphone applications. In *2011 IEEE International Symposium on Workload Characterization (IISWC)* (San Diego, CA, USA, November 2011), IEEE, pp. 81–90.

[9] KOPYTOV, A. Sysbench: A system performance benchmark. `http://sysbench.sourceforge.net/`, 2004.

[10] KUMAR, R., FARKAS, K. I., JOUPPI, N. P., RANGANATHAN, P., AND TULLSEN, D. M. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture* (Washington, DC, USA, 2003), MICRO 36, IEEE Computer Society, pp. 81–92.

[11] MCKENNEY, P. E., BOYD-WICKIZER, S., AND WALPOLE, J. RCU usage in the linux kernel: One decade later. Technical report paulmck.2012.09.17, September 2012.

[12] MCKENNEY, P. E., EGGEMANN, D., AND RANDHAWA, R. Improving energy efficiency on asymmetric multiprocessing systems. Technical report paulmck.2013.03.07a `http://rdrop.com/users/paulmck/techreports/AMPenergy.2013.03.07a.pdf`, March 2013.

[13] MCKENNEY, P. E., AND SLINGWINE, J. D. Read-copy update: Using execution history to solve concurrency problems. In *Parallel and Distributed Computing and Systems* (Las Vegas, NV, October 1998), pp. 509–518.

[14] MORAD, T., WEISER, U., KOLODNYT, A., VALERO, M., AND AYGUADE, E. Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors. *Computer Architecture Letters 5*, 1 (jan.-june 2006), 14 –17.

[15] RASMUSSEN, M. Update on Big.LITTLE scheduling experiments. `http://www.linuxplumbersconf.org/2012/wp-content/uploads/2012/09/2012-lpc-scheduler-task-placement-rasmussen.pdf`, August 2012.

[16] RODRIGUES, R., ANNAMALAI, A., KOREN, I., AND KUNDU, S. Scalable thread scheduling in asymmetric multicores for power efficiency. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on* (oct. 2012), pp. 59 –66.

[17] THE EMBEDDED MICROPROCESSOR BENCHMARK CONSORTIUM. AndEBench. `http://www.eembc.org/andebench/about.php`, 2012.

[18] WILLIAMS, C. rt-tests. Available: `http://git.kernel.org/cgit/linux/kernel/git/clrkwllms/rt-tests.git` [Viewed August 10, 2012], 11 2012.

A Raw Data

Table 2 shows the raw data used to produce Table 1. Each set of five data samples is sorted into increasing order to make it easy to see overlap between samples. All energy measurements are in Joules and all time measurements are in seconds.

	Reference		RCU Processing Offload		Enforced Idle	
Benchmark	Energy	Time	Energy	Time	Energy	Time
cyclictest	1.62	3.12	1.42	3.12	1.40	3.12
	1.68	3.12	1.42	3.12	1.42	3.13
	1.74	3.12	1.46	3.12	1.44	3.13
	1.80	3.12	1.48	3.13	1.45	3.13
	1.94	3.15	1.59	3.67	1.65	3.14
sysbench	32.48	8.98	31.44	8.98	31.06	8.97
	32.54	9.00	31.50	8.99	31.10	8.98
	32.79	9.03	31.65	8.99	31.12	8.99
	32.80	9.03	31.70	8.99	31.16	9.00
	32.81	9.68	31.74	9.00	31.17	9.01
andebench2	59.36	20.29	57.67	20.27	56.75	20.31
	59.38	20.32	57.86	20.29	56.81	20.31
	59.52	20.33	57.90	20.39	56.84	20.41
	59.64	20.36	58.01	20.41	59.73	23.39
	59.69	21.39	58.11	20.49	59.84	23.51
andebench8	171.40	45.27	167.72	45.29	164.92	45.62
	173.45	45.46	168.11	45.52	165.61	45.65
	173.77	45.48	171.42	46.64	167.01	46.89
	175.08	46.45	171.72	46.73	167.40	47.09
	176.49	46.68	172.86	46.77	169.64	47.74
audio	7.73	30.00	6.84	30.02	5.50	30.02
	7.85	30.01	7.35	30.02	5.85	30.03
	7.85	30.01	7.46	30.03	6.04	30.03
	7.94	30.01	7.57	30.03	6.77	30.03
	7.97	30.02	7.73	30.03	7.24	30.03
audio+bbench	96.15	152.60	89.94	152.69	83.96	151.79
	98.67	155.18	92.03	152.80	85.63	155.76
	99.11	157.14	93.35	154.93	86.69	156.77
	99.86	157.82	93.75	158.71	87.56	160.54
	101.43	158.74	96.01	158.80	90.93	178.89

Table 2: Raw Data For Energy Consumption and Performance on TC2