How SIMD Width Affects Energy Efficiency: A Case Study on Sorting

Hiroshi Inoue

IBM Research – Tokyo
19-21 Nihonbashi Hakozaki-cho, Tokyo, 103-8510, Japan
E-mail: inouehrs@jp.ibm.com

Abstract: This paper studies the performance and energy efficiency of in-memory sorting algorithms. We put emphasis on the SIMD (single instruction multiple data) mergesort implemented with different SIMD widths. By evaluating the performance, power, and energy with various hardware configurations (achieved by changing the memory bandwidth, number of cores, and processor frequency), our results show that SIMD can reduce power in addition to enhancing the performance, especially when the memory bandwidth is not sufficient to fully drive the cores. We also show that balancing the computation power and the memory bandwidth is important to minimize the total energy consumption. (Keywords: SIMD, sorting, memory bandwidth, energy efficiency)

Sorting is one of the most fundamental operations in many workloads. Due to its importance, a large number of sorting algorithms have been studied. Recently, multiway mergesort implemented with SIMD instructions has been used as a high performance in-memory sorting algorithm [1, 2]. Radix sort is another widely used sorting algorithm due to its smaller computational complexity. Satish *et al.* [3] conducted detailed comparisons of these two algorithms on CPUs (using SSE) and GPUs. They reported that on both platforms, the radix sort slightly outperformed the SIMD mergesort, but they predicted that the SIMD mergesort will be the algorithm of choice for future processors. Here, a wider SIMD and a lower bandwidth-to-compute ratio of future processors are the key for the superior performance of the SIMD mergesort.

In this paper, we evaluated the performance of sorting algorithms on a commodity PC with emphasis on energy efficiency in addition to execution time. We compared the performances of optimized implementations of the SIMD and non-SIMD mergesort, the radix sort, and the quicksort (the parallel version of std::sort included in gcc). We implemented the SIMD mergesort using 4-way SIMD (SSE) and 8-way SIMD (AVX) to study the effect of the SIMD width. We used the SIMD combsort, also implemented with AVX or SSE, for sorting small blocks [1]. For the radix sort, we implemented a cache-conscious radix sort [4], which combines the MSB-radix sort and LSB-radix sort to improve the memory-access locality. We also applied the local-buffer-based optimization proposed by Satish *et al.* [3] to maximize the write combining opportunities in the processor. For the evaluation, we used a desktop PC equipped with a Core i7 4770 (Haswell) processor, which had four 3.4-GHz cores and eight hardware threads by Hyper Threading. With turbo boost enabled, the frequency was increased to 3.9 GHz (for executions with 1 or 2 cores) or to 3.7 GHz (with 4 cores). It was also equipped with 8 GB of system memory (two 4-GB DDR3-1333 DIMMs for dual channel operation). In some evaluations, we removed one of the DIMMs installed in the system to reduce the system memory bandwidth by half (with single channel). We call the first configuration *full bandwidth configuration*, and the second one *half bandwidth configuration*. The system ran Redhat Enterprise Linux 6.5 as the operating system. We compiled all the programs using gcc-5.2 with the –O3 option. We measured the average system-level power using an external power meter (Yokogawa WT210). We evaluated the execution time and power consumption of each algorithm while repeatedly sorting 256 million 32-bit random integers (1 GB in total).

Figure 1 shows the execution time, the average power, and the total energy (power × execution time) using 1 to 8 threads. Figure 2 shows the number of executed instructions and CPI (cycles per instruction) as measured by the performance counter. Regardless of the number of threads, the mergesort with AVX achieved the best absolute performance, and the radix sort was a close second best. These two algorithms outperformed the quicksort (std::sort) by more than 7 times with 1 thread and by more than 4 times with 8 threads. Compared to these huge performance boosts, the increases in the system power were not so significant; the power for the mergesort with AVX and the radix sort were higher than that of the quicksort by only up to 12.4% and up to 6.3% respectively. As a result, these two algorithms executed the entire sorting with the smallest total energy among the tested algorithms regardless of the the number of threads. Among three implementations of mergesort with different SIMD widths, scalar (1 way), SSE (4 way), and AVX (8 way), the two SIMD versions achieved much higher absolute performance than the scalar version. With 1 thread, for example, the AVX version yielded a 9.7x speedup over the scalar version, and the SSE version yielded a 6.8x speedup. Since the increases in the system power were much smaller than the speedups, the SIMD versions were much more energy efficient than the scalar version. By increasing the SIMD length twice, the AVX version was 42% and

14% faster than the SSE version with 1 thread and 8 threads, respectively. The performance gain from the wider SIMD was less significant with 8 threads because the execution performance was bounded by the system memory bandwidth; therefore, the CPI of the AVX version drastically degraded for large thread counts, as shown in Figure 2, due to longer memory stalls. The system power was 3.1% higher for AVX compared to that of SSE with 1 thread in trade for the higher performance. Interestingly, with 8 threads, the power was 1.8% lower for AVX compared to SSE despite AVX achieving a 14% higher performance. As shown in Figure 2, we reduced the number of executed instructions by 1.74x using AVX over SSE. When the memory bandwidth was sufficient to fully drive all cores, the reduced instructions resulted in better performance. However, when the memory bandwidth limited the performance by increasing the memory access latency, the reduced instructions resulted in reduced power in the processor.

To investigate how the memory bandwidth affects the energy efficiency of the SIMD mergesort, Figure 3 shows the power (x-axis) and the throughput (y-axis) for mergesort with two memory bandwidth configurations. When the memory bandwidth was reduced by half, the performances of the SIMD versions were degraded by about 40% due to the bandwidth bottleneck. The performance of the scalar version was not significantly affected by the memory bandwidth because of its lower performance and hence lower bandwidth requirement. The reduction in power by SIMD was more significant with the half bandwidth configuration than with the full bandwidth configuration. When using 4 or 8 threads with the half bandwidth configuration, the scalar version showed the highest system power despite having the lowest performance. The AVX version yielded the highest performance and the lowest power. This result shows that the power reduction by SIMD can be much larger on systems with lower bandwidth-to-compute ratios. Since future systems are predicted to have lower bandwidth-to-compute ratios compared to today's systems, the SIMD instructions will play a key role to achieve lower power as well as higher performance in the future.

For more insight into the effects of the bandwidth-to-compute ratio on energy efficiency, we controlled the processor's core frequency by DVFS (SpeedStep) in addition to changing the bandwidth configurations. Figure 4 illustrates the power and performances of the mergesort with AVX with two memory bandwidth configurations. The basic trends were common with two configurations. Using more cores with a lower frequency improved the energy efficiency as expected, i.e., this yielded higher performance with the same power or lower power while keeping the same performance. The SMT capability gave performance gains only with the low processor frequencies for the mergesort. The lowest total energy was achieved when using four cores of 2.5 GHz and 1.5 GHz with the full and half bandwidth configurations, respectively. The lowest energy of the full bandwidth configuration was better than that of the half bandwidth configuration by about 17.5%. Although we do not show it here, the scalar version achieved the lowest energy with 3.4 GHz even with the half bandwidth configuration. This means that it is important to balance the bandwidth and the computation performance to achieve the best energy efficiency. Figure 5 shows the differences in the total energy between the full and half bandwidth configurations. When the bandwidth requirements were large, i.e., with a higher frequency and a larger thread count, using the full bandwidth configuration reduced the energy by more than 30%. On the contrary, when the bandwidth requirements were small, the full bandwidth configurations increased the total energy by a few percent. In this experiment, we reduced the memory bandwidth and energy by removing a DIMM from one channel, but using memory DVFS [5] is an alternative way. When we did not use SIMD, the half bandwidth configuration was better even with the highest frequency and 8 threads. The SSE version fell between the two. Therefore, to minimize energy consumption, we need to control the memory bandwidth and balance it with the demands of the running applications.

[1] H. Inoue, T. Moriyama, H. Komatsu, and T. Nakatani. AA-Sort: A New Parallel Sorting Algorithm for Multi-Core SIMD Processors. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, pp. 189–198, 2007.

[2] J. Chhugani, A. D. Nguyen, V. W. Lee, W. Macy, M. Hagog, Y.-K. Chen, A. Baransi, S. Kumar, and P. Dubey. Efficient implementation of sorting on multi-core SIMD CPU architecture. In *Proceedings of VLDB Endow.*, 1 (2), pp. 1313–1324, 2008.

[3] N. Satish, C. Kim, J. Chhugani, A. D. Nguyen, V. W. Lee, D. Kim, and P. Dubey. Fast sort on CPUs and GPUs: a case for bandwidth oblivious SIMD sort. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pp. 351–362, 2010.

[4] D. J. González, J.-L. Larriba-Pey, and J. J. Navarro. Communication conscious radix sort. In *Proceedings of the 13th International Conference on Supercomputing*, pp. 76–82, 1999.

[5] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pp. 31–40, 2011.

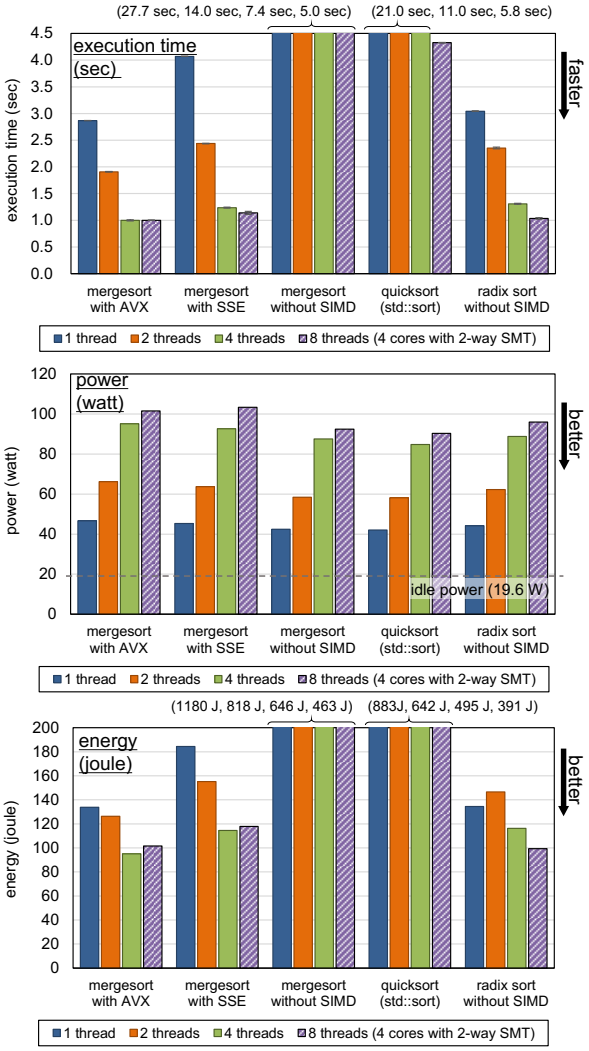

Fig. 1 Execution time, power, and energy with various sorting algorithms with and without SIMD.

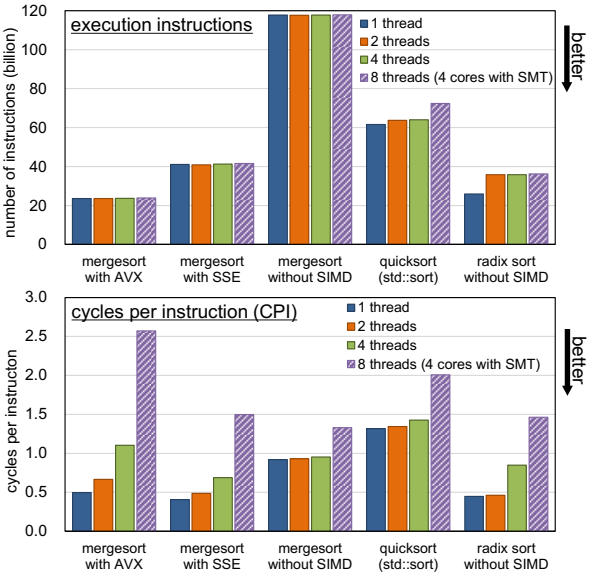

Fig. 2 Number of executed instructions and CPI with various sorting algorithms as measured by HPM.

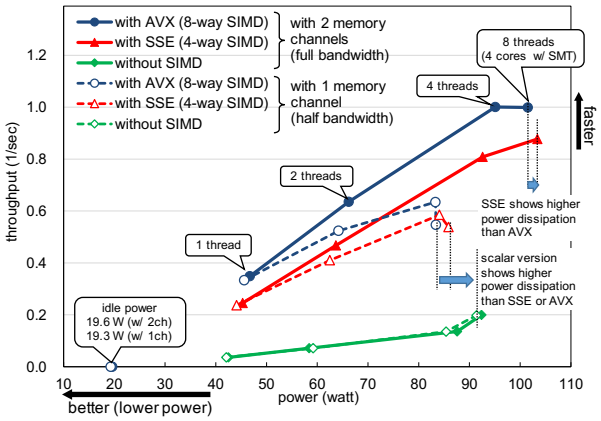

Fig. 3 Power and performance of mergesort with and without SIMD with two bandwidth configurations.

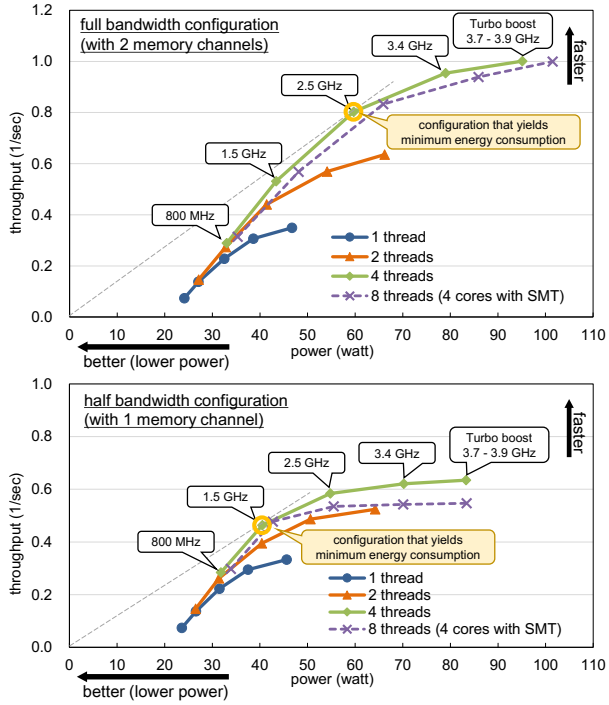

Fig. 4 Power and performance of mergesort with AVX (8-wide SIMD) with various core frequencies with two bandwidth configurations.

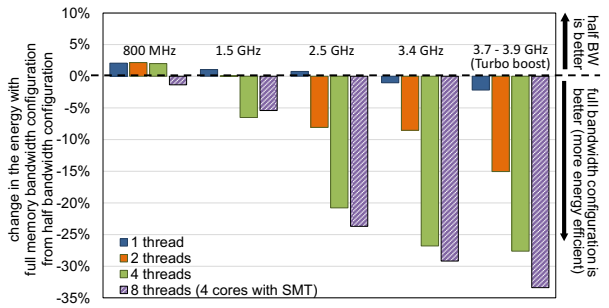

Fig. 5 Comparisons of total energy consumption with mergesort implemented with AVX for full (w/ 2 memory channels) and half (w/ 1 memory channel) bandwidth configurations.