Power and Performance Characterization, Analysis and Tuning for Energy-efficient Edge Detection on Atom and ARM based Platforms

Paul Otto, Maria Malik, Nima Akhlaghi, Rebel Sequeira, Houman Homayoun, Siddhartha Sikdar
Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, USA
{potto, mmalik, nakhlagh, rsequei2, hhomayou, ssikdar}@gmu.edu

The de facto standard for embedded platforms with medium to low computing demands are ARM with Thumb ISA and Intel Atom with the X86 ISA with multiple cores. Operating these architectures in the milliwatts range while running realtime computer vision corner detection algorithms is a challenging problem. We present the analysis of power, performance and energy-efficiency measurements of Harris corner detection across a wide range of voltage and frequency settings, multicore/multithreading strategies, and compiler and application optimization parameters to find how the interplay of these parameters affect the power, performance and energy-efficiency. Our measurement of results on state-of-the-art embedded platforms demonstrate that a systematic cross-layer optimization at the application level (Sobel filter type, aperture size, number of image tiles), compiler level (branch prediction, function inlining) and system level (voltage and frequency setting, single core vs multicore implementation) significantly improves the energy-efficiency of corner detection, while meeting its real-time performance constraints. This cross-layer optimization improves the energy-efficiency of Harris corner on Atom and ARM by 89.5% and 87.2%, respectively.

Keywords—Harris corner Detection, Embedded Platforms, Energy Delay Product; Algorithm Optimization; System optimization

I. INTRODUCTION

Corner detection algorithms are at the heart of many real-time computationally intensive computer vision systems. These systems are used in rehabilitation, fall detection [1], tissue analysis [2], and surgical assistance [3]. The real-time processing requirements of corner detection algorithms becomes challenging on resource-constrained embedded systems with milliwatt power budgets. Because of its multiple uses, focusing on optimizing blocks of the Harris corner algorithm has received strong interest in recent years [7][8]. Recent works have used high performance multicore CPU [9], FPGA [11], or even GPU cores to speed up edge detection [6, 10]. While these architectures offer immense speed benefits for high performance, they come with significant power cost. Even embedded GPUs that are emerging in mobile platforms such as Nvidia Tegra with 192 CUDA cores are operating at a 5~10 watt power range, making them an undesirable choice for low-power computing.

In this work, through methodical investigation of power and performance measurements, and comprehensive system level and application level characterization, we demonstrate that carefully tuning optimization parameters at the application, compiler, and system levels can significantly improve the energy-efficiency of Harris corner detection on embedded low power platforms. We chose the state-of-the-art Intel Atom with X86 ISA and ARM with Thumb ISA as our processing platforms, because they are predominantly being used in today's low power embedded devices. We used the Energy-Delay^X Product (ED^{X}P) to understand how near real-time performance constraints for edge detection affects the optimization parameters across different microarchitectures. Below, we set forth our methods in Section II, the results in Section III, our discussion of the key results in Section IV, and our conclusions in Section V.

This work was supported in part by Grant Number CSR 0953652 and CPS 1329829 from the National Science Foundation.

II. METHODS

The algorithm under test is the Harris corner and edge detection algorithm in OpenCV[5], which determines if a given image block region is an edge or corner[4]. First, spatial gradients of an image are computed by a Sobel filter of a given aperture length. Increasing the aperture size reduces noise, but blurs edges. The edge/corner test is applied to image blocks. Profiling the code with the Intel VTune Amplifier XE determined that the Sobel filter function used the most processing time. So, the Sobel filter's aperture size was the focus of the application level optimization. Varying the image block region size did not impact performance.

We conducted our study independently on the Intel Atom quad core C2758 at 2.4 GHz, a 64-bit high-end processor with dual-issue out-of-order micro-architecture, and NVIDIA's "4-Plus-1" 2.32GHz ARM quad-core Cortex-A15 CPU. Both platforms have four active processing cores. For the multithreading strategy a tiling algorithm was added to split the image into equal dimensioned rectangles with overlap. The rectangles were allocated to the cores using OpenMP.

Power consumption is measured using the Watts up PRO power meter (ThinkTank Energy Products, VT) for the entire system. To estimate core power consumption, the system idle power is subtracted from the measured power when the applications are running.

The optimization was performed in three stages: system (processor frequency), architecture (compiler level), and application-level (aperture size) to explore execution time and power folded into the ED^{x}P metric. EDP was used except when ED^{3}P was applied to quantify the near real time

performance metrics. Table 1 details the optimization parameters.

TABLE I. OPTIMIZATION PARAMETERS

Processor Frequency	Compiler Optimization Level (gcc)
F1 = 1.2 GHz	O0 = No optimization
F2=1.8 GHz	O1 = Branch predicitions
F3=2.4 GHz	O2 = O1+Jump, loop, and label alignment
	O1+O2+Inline functions
Application Implementation	
P1	Single core with baseline aperture filter
P2	Single core with symmetric aperture filter
P3	Multicore with symmetric aperture filter
Aperture Filter Size	
A3-A11	Aperture size 3 through 11

For optimization configuration comparisons P1 was used as the baseline for power and performance. P2 includes an optimized filter on a single core implementation. P3 builds off of P2 and was written as a multithreaded, multicore program. Finally, we analyzed the effect of single core vs multicore on EDP by running P1, P2 and P3 with the fixed parameters F2, O3, and A11. The lowest EDP configuration was selected to analyze the effect of the application parameter and aperture size on EDP.

III. RESULTS

We began the cross-layer system and application-level optimization procedure by varying the processor frequency. We studied three levels of processor frequencies 1.2 GHz (F1), 1.8 GHz (F2) and 2.4 (F3) for the single core case P2 and the multicore case P3. The system level compiler optimization was constrained to O3, so the concurrent system level optimization of processor frequency and multithreading strategy (core count) could be investigated. Additionally, the aperture size which is an application level parameter was constrained to A11. Fig. 1B shows that for the ARM processor, multicore and single core modes produce similar EDP values for different frequencies. Also the lowest frequency (F1) has the lowest EDP.

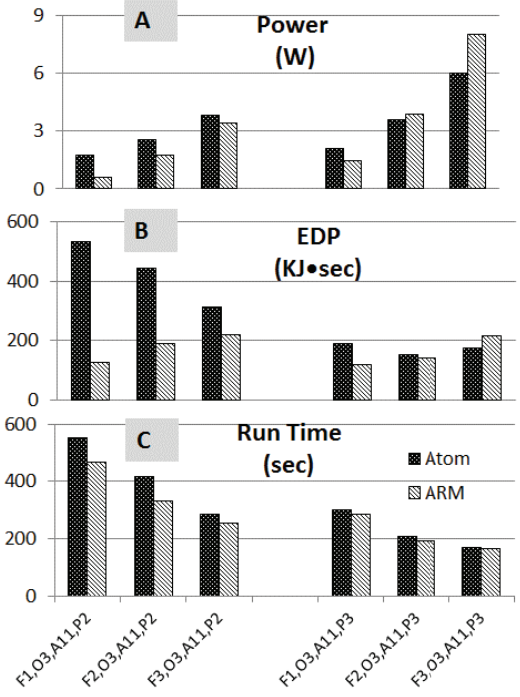

Fig. 1. Evaluation of Atom and ARM single core and multicore measures at multiple processor frequencies. (A) Measure of adjusted power in watts, (B) Measure of EDP in Joule seconds, (C) Measure of program run time in seconds.

The Atom processor results showed a different trend. While the power results (Fig. 1A) share a similar trend to the ARM for the single vs multicore mode, the EDP results (Fig. 1B) have an opposite trend compared to the ARM for the single core setting. Additionally, the multicore EDP results do not show a significant dependence on frequency. We further study impact of frequency setting on EDP and ED^3P (Fig. 2). ED^3P metric has a greater run time penalty than EDP and represents a more near-real time constraint. The results (Fig. 2) show that the ARM has a minimum ED^3P at F2, while the Atom processor shows a minimum at both F2 and F3.

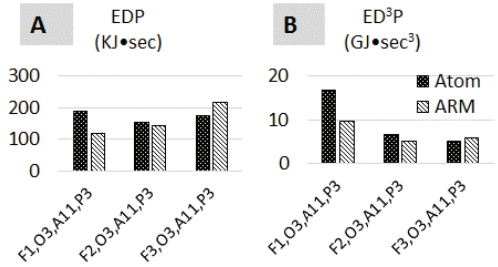

Fig. 2. EDP and ED^3P sensitivity to operating frequency

The run time performance of the application (Fig. 1C) shows that the Atom multithreaded multicore results were almost twice as fast as the single core results and that the L2 cache misses per 1000 instructions was 0.106 for a single core and 0.142 for four cores. This translates into a 34% increase in L2 cache misses going from single core to multicore.

TABLE II. COMPARING RUN TIME TO JUMP INSTRUCTION RATIO ON ATOM

Optimization Level	Run Time (sec)	Total Instructs (count)	JMP Instructs (count)	Ratio JMP to Total
O0	513	13798	338	0.02
O1	225	13798	850	0.08
O2	216	10278	853	0.08
O3	207	12489	965	0.08

Fig. 3 computes the increase in adjusted power for the single core runs (Fig. 3A) and the multicore runs (Fig. 3B). Adjusted power is computed by scaling the measured power to account for the change in voltage due to the change in frequency. The increase in adjusted power for the multicore implementation is twice that of the single core implementation, showing power sensitivity to frequency twice as high in multicore implementation. The increase in power with frequency is consistent for both processors.

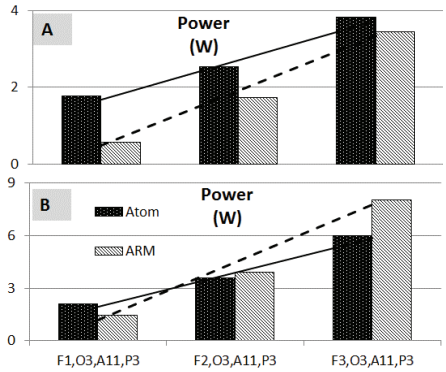

Fig. 3. Comparision of power change over frequency for Atom and ARM (A) single core power slopes Atom: 1.0W/freq step and ARM 1.4 W/freq step, (B) quad core power slopes Atom: 2.0W/freq step and ARM 3.3 W/freq step

The system optimization results can be summarized on the far left part of Fig. 3. This highlights the EDP measurements for the multicore implementation where it was found that the ideal processor frequency was F2 for both the ARM and Atom in light of Fig. 1, Fig. 2, and Fig. 3. Next, we explore the system level and compiler optimization settings for the multicore implementation (P3) by constraining the aperture size to A11 and varying the optimization settings between O0, O1, O2, and O3. The EDP results (Fig. 4A) show little benefit in EDP between O1, O2, and O3. However, there is a 4x reduction in EDP for both the ARM and Atom when changing from no optimization (O0) to basic optimization (O1). It is hypothesized this is due to the use of conditional jump instructions in the O1-O3 optimizations. Table IV shows a 56% decrease in execution time for the Atom processor going from O0 to O1. At the same time, the ratio of conditional jump to total instruction count increased from 0.02 to 0.08.

Next, the single core implementation P1 was compared with P2 and P3. The difference between P1 and P2 is that P2 uses a symmetric coefficient filter design, while P1 implements a coefficient agnostic filter algorithm. The results (Fig. 4A) show that the ARM processor showed little reduction in EDP between P1 and P3, while the Atom processor EDP dropped by nearly 60%. This result can be explained by examining Fig. 2C (F2, O3, A11, P2) and (F2, O3, A11, P3) where Atom's run time decrease was greater than the ARM's, while the ARM's power increase was greater than Atom's. This combination lead to a greater EDP drop for Atom than for the ARM.

Finally, we show (Fig. 4A) as the aperture decreases the EDP decrease. Aperture induced algorithm inaccuracies were measured as the change in total features within a cluster as the aperture size was varied. The standard deviation of the average cluster count over frames was computed as 0.49. This means that on average between any change in aperture size the total feature count in a cluster either will not change or will change by the addition or deletion of a single feature. This consistency shows that while aperture parameter impacts the power efficiency significantly, it does not reduce the effectiveness of the corner detector. Fig. 4(B) shows the expected monotonic trend in energy, power, EDP and execution time with frequency. The exception is the EDP at 1.2 GHz for the Atom processor, which is explained by the increase in execution time as can be seen in (Fig. 1C).

IV. Discussion

Cross-layer system, architecture, and application-level optimizations using EDP as a metric reveal that an energy-efficient Harris corner detector requires running the ARM at a low frequency with single core, while the corner detector is most efficient on the Atom in a multicore mode at low frequency. This result was unexpected and demonstrates that the implementation choice is not as simple as choosing single

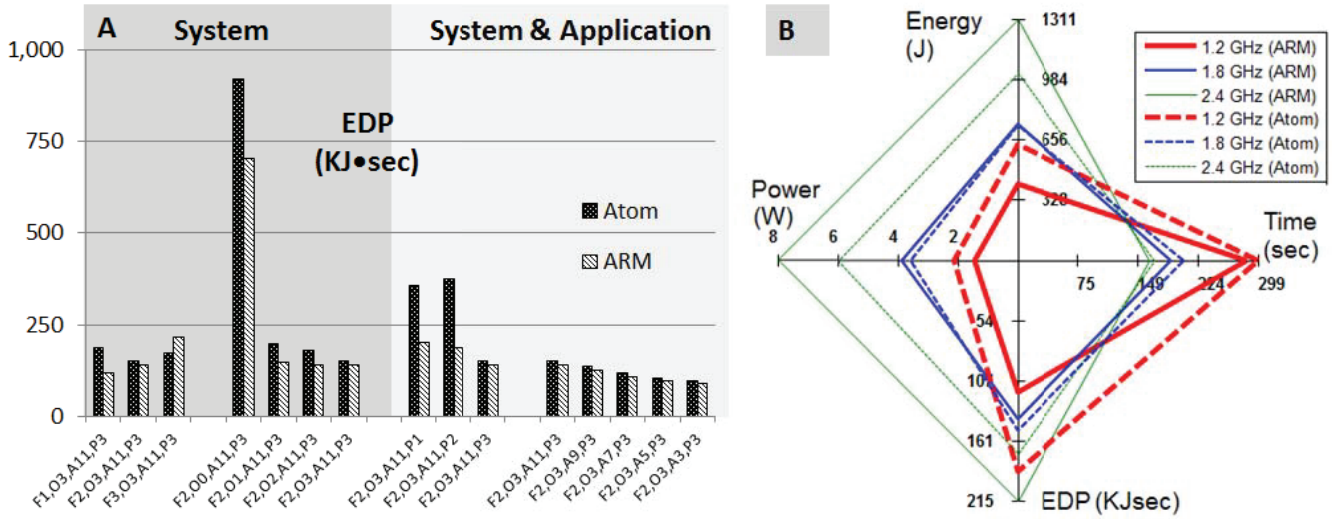

Fig. 4. (A) Summary of system and system + application optimization results for the ATOM and ARM processors. (B) Summary of Atom and ARM measurements for multicore implementation, fixed optimization level O3, and an aperture of size 11 for multiple frequencies.

core vs multicore. This is in part due to a less than expected decrease in run time performance when switching to the multicore implementation (Fig. 1C) for both architecture and in particular ARM. The expected runtime decrease was 4x, but L2 cache misses from multiple cores attempting to simultaneously access a memory subsystem are causing the less than expected performance increase. Another unexpected performance result is that neither processor's power increased by a factor of 4 going from single core to a quad core implementation (Fig. 4A). The average power ratio between the quad core and single core at a particular processor frequency was 1.4 for the Atom processor and 2.4 for the ARM with a standard deviation of 0.2 and 0.1, respectively. These results suggest that both the single core's and the quad core's implementation performances are similarly affected by operating frequency because the ratio was constant over processor frequency. However, the power change as a function of frequency for the multicore was twice as much as for the single core, demonstrating high sensitivity of power to frequency in multicore implementation compared to single core implementation.

The system level compiler optimization revealed an unexpected result: while compiler optimization is important for both the Atom and ARM processors, there is a law of diminishing returns. Optimization levels O2 and O3 did not offer significant EDP performance increases over O1. However, no optimization, O0, had a 4x higher EDP than O1. This difference is most likely due to the performance benefit of a few simple optimizations such as branch predictions enabled in O1.

Finally it was found that at the application level changing block size has no significant impact on performance and EDP. However, tuning aperture size results in an EDP reduction of 30% with a minimal impact on feature detection because the change in total feature count in a cluster had a standard deviation of 0.49. This is an important finding because the EDP metric does not measure the change in application result accuracy when changing an application parameter.

V. Conclusion

Energy-efficient yet high performance processing of corner and edge detection algorithms is required for effective deployment of computer vision application in a mobile environment. In this paper through methodical investigation of power and performance measurements on state-of-the-art ARM based and Atom based embedded platforms, we demonstrate how carefully tuning optimization parameters at the application, compiler, system, and microarchitecture levels can significantly improve the energy-efficiency of Harris corner edge detection applications. The results show how dividing the optimization process into system and application optimizations provide a logical sequence for monotonically decreasing EDP. Furthermore we make the following key findings: (1) EDP reduction through parallelization is dependent on the hardware microarchitecture/ISA. The Atom processor benefited by a 50% EDP reduction, while the ARM's improvement was only 25%. (2) Multitcore/multithreading strategy, i.e. increasing the number of active cores, is not the sole critical metric when choosing energy efficient platforms based only on hardware parameters. The optimization decision should include processor frequency with the core count. At 1.2 GHz the ARM processor was most efficient as a single core, while the Atom was in quad-core mode. However, when the frequency was increased to 1.8 GHz, both processors had minimal EDP values in quad-core mode. (3) Compiler optimization can greatly impact EDP. The addition of basic compiler optimizations reduced EDP by a factor of 4. (4) Application-level parameters with OpenMP significantly affect EDP and are microarchitecture dependent. Optimization of the application parameters reduced the EDP by 73% for the Atom processor and 56% for the ARM. This translates into a decrease in runtime of 63% (Atom) and 54% (ARM) with a corresponding energy consumption decrease of 26% (Atom) and 5% (ARM). The microarchitecture choice varied the EDP change by 17%. (5) The ED^xP results show that operating frequency should be carefully tuned depending on the underlying microarchitecture/ISA to maximize energy-efficiency while meeting a near-real time constraint.

Overall, a cross-layer optimization at the application, compiler, system, and architecture levels improves the energy-efficiency of Harris corner detection by 89.5% and 87.2%, on embedded Atom and ARM platforms, respectively.

VI. References

[1] H. Zhou and H. Hu, "Human motion tracking for rehabilitation—A survey," *Biomed. Signal Process. Control*, vol. 3, no. 1, pp. 1–18, Jan. 2008.

[2] H. Alemdar and C. Ersoy, "Wireless sensor networks for healthcare: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2688–2710, Oct. 2010.

[3] D. R. Uecker, Y. F. Wang, C. Lee, and Y. Wang, "Laboratory Investigation:Automated Instrument Tracking in Robotically Assisted Laparoscopic Surgery," *Computer. Aided Surg.*, vol. 1, no. 6, pp. 308–325, Jan. 1995.

[4] C. Harris and M. Stephens, "A combined corner and edge detector.," in *Alvey vision conference*, 1988, vol. 15, p. 50.

[5] G. Bradski, "The OpenCV Library," *Dr Dobbs J. Softw. Tools*, 2000.

[6] L. Teixeira, W. Celes Filho, and M. Gattass, "Accelerated Corner-Detector Algorithms.," in *BMVC*, 2008, pp. 1–10.

[7] C. G. Kim, J. G. Kim, and D. H. Lee, "Optimizing image processing on multi-core CPUs with Intel parallel programming technologies," *Multimedia. Tools Appl.*, vol. 68, no. 2, pp. 237–251, Nov. 2011.

[8] Z. E. M. Osman, F. A. Hussin, and N. B. Z. Ali, "Optimization of Processor Architecture for Image Edge Detection Filter," in *2010 12th International Conference on Computer Modelling and Simulation (UKSim)*, 2010, pp. 648–652.

[9] F. Hosseini, A. Fijany, and J.-G. Fontaine, "Highly Parallel Implementation of Harris Corner Detector on CSX SIMD Architecture," in *Euro-Par 2010 Parallel Processing Workshops*, M. R. Guarracino, F. Vivien, J. L. Träff, M. Cannatoro, M. Danelutto, A. Hast, F. Perla, A. Knüpfer, B. D. Martino, and M. Alexander, Eds. Springer Berlin Heidelberg, 2011, pp. 137–144.

[10] S. Luo and J. Zhang, "Accelerating Harris Algorithm with GPU for Corner Detection," in *2013 Seventh International Conference on Image and Graphics (ICIG)*, 2013, pp. 149–153.

[11] M. F. Aydogdu, M. F. Demirci, and C. Kasnakoglu, "Pipelining Harris corner detection with a tiny FPGA for a mobile robot," in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2013, pp. 2177–2184.