Automatic Runtime Selection of Best Hardware for Data-Parallel JavaScript Kernels via Lifelong Profiling

Younghwan Oh Channoh Kim Xianglan Piao Jae W. Lee
Sungkyunkwan University
Suwon, Korea
{garion9013, channoh, xianglan0502, jaewlee}@skku.edu

ABSTRACT

This paper introduces a runtime framework that automatically selects the best hardware for a data-parallel JavaScript kernel. Assuming an application runs repeatedly, which is common in a personalized mobile device, the runtime system builds the performance curve of a kernel across multiple invocations with various settings of a user-selected kernel parameter (e.g., input size). In an early stage of deployment with a small number of data points, the runtime system dispatches the kernel to multiple devices concurrently to accelerate performance characterization and takes the outcome from the fastest device, to not compromise user experience. With enough data points gathered, the runtime system dispatches the kernel only to the device predicted to perform the best for the given input, thus minimizing computational waste. Our preliminary evaluation with two devices—a CPU and a GPU—demonstrates that the proposed system matches the performance of the best performing device at each input point.

1. INTRODUCTION

With widespread adoption of complex, compute-intensive web applications, performance demands for JavaScript programs are higher than ever. To meet these demands, it is key to efficiently exploit abundant hardware resources provided by modern multi-processor SoCs (MPSoCs). Today's MPSoCs typically integrate multi-core CPU, GPU, and accelerators on a single die, thus offering multiple choices to execute a function.

To harness the performance portential of these parallel, heterogeneous processing substrates, parallel programming frameworks have been proposed for JavaScript [1–3]. As interactive, media-rich applications become more and more popular on the web, it is particularly important for JavaScript to execute data-parallel workloads efficiently. For this, several frameworks adopt the OpenCL backend with different levels of abstraction, such as WebCL [1] and River Trail [2].

Even if data parallelism within an application is exposed in the form of kernels, it is challenging to achieve robust performance over widely varying execution environments. The execution time of a kernel is affected by multiple runtime factors such as program inputs, deployment platforms, system load, and so on. As a result, choices made at development time with an anticipated range of these factors easily become suboptimal outside that range. For example, it is not uncommon for a data-parallel kernel to perform poorly on GPU for small inputs—even worse than a sequential version running on CPU due to significant startup overhead.

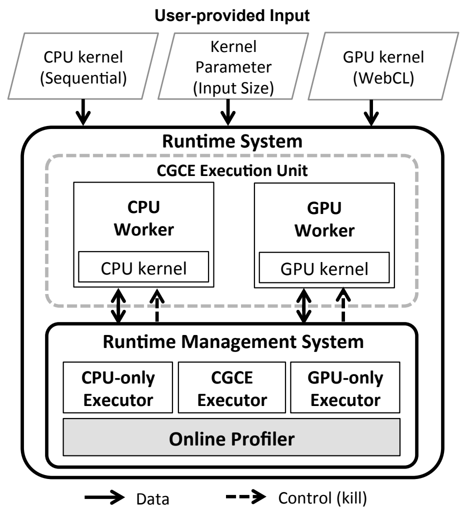
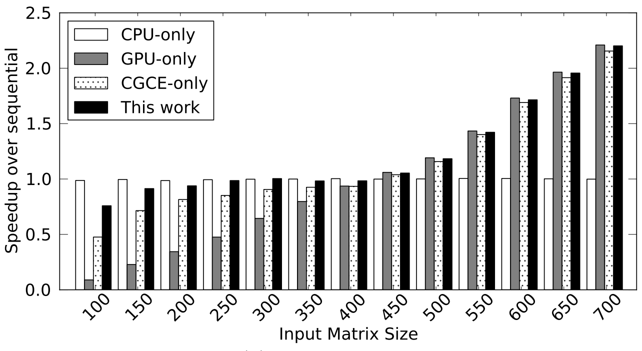


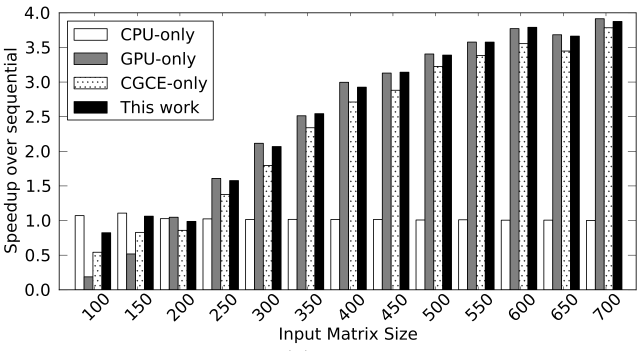
Figure 1: Overall system structure

The problem of *performance fragility* is more severe in JavaScript than in conventional static languages, say, C. The execution environment of JavaScript is highly dynamic with Just-In-Time (JIT) compilation, dynamic typing, and asynchronous event handling, which often yields unintuitive performance behaviors. This generally leads to a greater performance variability. In addition, the JavaScript compilation time adds up to the total execution time, thus degrading the effectiveness of compile-time profiling-based tuning (e.g., PetaBricks [5]). *N-way* programming model [6] overcomes this limitation via runtime selection. However, this benefit comes with significant waste due to redundant computation, making it difficult to apply the technique on a resource-constrained mobile device.

To address this challenge we propose a runtime framework for lifelong profiling that automatically selects the best performing device for a given data-parallel JavaScript kernel. The proposed framework provides a JavaScript API to take multiple versions of a kernel optimized for different devices and exposes a kernel parameter representing the characteristics of the input to the kernel (e.g., input matrix size for a matrix multiply kernel). The runtime system constructs a performance curve of the kernel throughout its lifetime by logging execution results for multiple devices and input data. Using this performance curve, the runtime system can predict the best performing device for the given kernel and input data to achieve robust performance with minimal degradation of performance and energy efficiency.



(a) Gramschmidt



(b) Syrk

Figure 2: Speedups over original sequential execution for two Polybench benchmarks

2. DESIGN AND IMPLEMENTATION

Figure 1 shows the structure of the proposed framework. Our prototype uses only two versions of the kernel, optimized for a CPU and a GPU, respectively, but the framework can be easily extended to support additional devices and kernel versions. The CPU kernel is the original sequential version, and the GPU kernel is a parallel WebCL version. The proposed API allows the user to declare the kernel parameter variable to track for profiling, and we use the input size as default kernel parameter. The runtime system is encapsulated in a JavaScript object, and its constructor takes the two kernel versions and the kernel input as arguments.

There are three modes of kernel execution: CPU-only, GPU-only, and CPU-GPU Competitive Execution (CGCE). Initially, with no existing performance curve, the system executes the kernel in the CGCE mode. The runtime system dispatches the kernel to both CPU and GPU, and takes the output from the faster device. Two Web Workers [3] are created to manage CPU and GPU execution, respectively. To minimize worker creation overhead, the workers are reused across kernel invocations like a thread pool. A performance curve is built by gathering input size-execution time pairs over multiple executions of the kernel.

As the system gathers enough data points to make a high-confidence prediction, it dispatches the kernel only to the device predicted to perform better for the given input, to run in either CPU-only or GPU-only mode. In these two modes, the kernel is executed on the main process, instead of running on a worker, to avoid the overhead of communication and synchronization. Therefore, we can achieve high performance comparable to the best of the two devices over a variety of kernel inputs. CGCE execution performs slightly worse than the best of CPU-only and GPU-only execution due to the overhead of managing competitive execution.

We employ a simple mode selection algorithm assuming at most one cross point exists between the performance curves of the two devices as the input size increases. For the two benchmarks we used, CPU-only mode favors small inputs, and GPU-only mode large inputs. Therefore, kernel inputs can be classified into three categories based on its size: CPU-only, GPU-only and undecided. Initially, all inputs fall in the undecided region. As CPU (GPU) wins in CGCE execution, the CPU (GPU) region is expanded to cover the minimum and maximum input sizes for which the CPU (GPU) wins.

3. PRELIMINARY RESULTS

We prototype the proposed system on WebKit with Samsung's WebCL support [4]. We use two Polybench benchmarks for which there is a cross point between CPU-only and GPU-only performance curves as the input size varies: `gramschmidt` and `syrk`. Note that, for programs that favor one device type for all inputs, there is no need for runtime selection. We take an average of 5 measurements for each data point on a system with Intel's 3.3GHz Core i5-2500 CPU and Nvidia's 1.4 GHz GeForce GT 530 GPU.

Figure 2 shows program speedups normalized to sequential execution of the out-of-the-box JavaScript version. It includes performance for each of the three execution modes without runtime selection, denoted by CPU-only, GPU-only, and CGCE-only, as well as our runtime selection framework. To evaluate the system's adaptivity across multiple program invocations, we generate 5 random permutations of input matrix sizes to guide the order of invocations.

The results show that the performance curve of our framework effectively tracks that of the best performing device with varying input sizes. For example, the cross point of CPU-only and GPU-only execution falls between input sizes of 400 and 500 for `gramschmidt`, and the proposed system correctly constructs the performance curve to favor CPU-only for inputs whose size is equal to or smaller than 400. For inputs whose size is equal to or greater than 450, the system favors GPU-only. Inputs whose size falls between 400 and 450 still remain undecided. CGCE-only also tracks the best of both, but with expensive computation waste by discarding the results from the slower device. In the future, we plan to expand this framework to handle multi-kernel applications and support multiple versions of a kernel (e.g., CPU parallel version with Web Workers) for a single device.

4. ACKNOWLEDGMENTS

This work was supported in part by the Korean Evaluation Institute of Industrial Technology funded by the Ministry of Science, ICT & Future Planning (KEIT-10047038) and the IT R&D program of MKE/KEIT [KI001810041244, Smart TV 2.0 Software Platform].

5. REFERENCES

[1] Parallel Computing on the Web. `http://webcl.nokiaresearch.com`.
[2] River Trail. `http://github.com/RiverTrail/RiverTrail`.
[3] Web Workers. `http://www.w3.org/TR/workers`.
[4] WebCL for WebKit. `http://github.com/SRA-SiliconValley/webkit-webcl`.
[5] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe. Petabricks: A language and compiler for algorithmic choice. In *Proc. of PLDI*, 2009.
[6] R. E. Cledat, T. Kumar, and S. Pande. Efficiently speeding up sequential computation through the n-way programming model. In *Proc. of OOPSLA*, 2011.