# Power-Aware Heterogeneous Computing Through CPU-GPU Hybridization

Kyle Siehl
School of Engineering and Computer Science
Washington State University
Vancouver, WA 98685
Email: kyle.siehl@wsu.edu

Xinghui Zhao
School of Engineering and Computer Science
Washington State University
Vancouver, WA 98685
Email: x.zhao@wsu.edu

***Abstract*—Graphic Processing Units (GPUs) have recently been widely used in general purpose computing, aiming for improving the performance of applications. However, this performance gain often comes with higher power consumption. In this paper, we present Archon, a framework for power-aware CPU-GPU hybridization. Specifically, Archon takes user's programs as input, automatically distribute the workload between CPU and GPU, and dynamically tunes the distribution ratio at runtime for an energy-efficient execution. Experiments have been carried out using a matrix multiplication application, and the results show that Archon can provide considerable energy savings, comparing to the CPU-only and GPU-only executions. These energy savings are achieved without extra efforts from the programmers.**

***Keywords*—Heterogeneous Computing, Power-Aware Computing, GPGPU, Tuning, Optimization.**

## I. Introduction

Across the history of computing, Moore's law [1] has governed the continual improvement of computer performance. Unfortunately, this steady drumbeat has been ground to a halt by the "power wall" [2], which is fundamentally caused by the fact that power consumption of chips increases nonlinearly with clock frequency [3]. As a result, computer architects are shifting hardware design to multicore architectures, and using dynamic voltage and frequency scaling (DVFS) to trade processing speed for power savings. The extreme end of this idea of adding cores, rather than increasing clock frequency, is exemplified by the graphics processing unit (GPU). While a more traditional CPU design will usually have between one and eight cores in the neighborhood of 4 GHz, current consumer-facing GPUs may have as many as 2048 cores running as slowly as 1 GHz. However, for many problems, these limitations are acceptable, and the performance benefits of accepting them can be enormous.

Although GPUs were originally designed to accelerate graphics operations, they have recently been used for general purpose computing, such as scientific applications. The use of GPUs in general applications provides orders of magnitude performance gain over the traditional CPU execution. It has been shown that including both GPU and CPU in general computations may also provide energy savings [4]. However, how to achieve both the performance and energy advantages is non-trivial. In this paper, we present Archon, a framework for power-aware heterogeneous computing through CPU-GPU hybridization. The framework takes user's program as input, uses a simple energy prediction model as a guideline, and dynamically distributes the computation's workload between GPU and CPU at run-time, aiming for optimizing both performance and energy consumption.

## II. Archon: CPU-GPU Hybridization Framework

Archon's basic workflow is shown in Figure 1, with user-provided code in blue and Archon's code in red. First, the user must have implementations of their algorithm for both the CPU and the GPU. The user writes a wrapper function, which calls into Archon's rebalance procedure to determine what fraction of the data each processor should get. The client program then splits some portion of the data (in whatever way is appropriate for the problem at hand) and hands the stated fractions of the data to worker threads, which perform the relevant computations while being wrapped with Archon's instrumentation code. This instrumentation code measures statistics about the running code (currently only the time taken), which is later used to determine optimally balanced splits. After these worker threads have both completed their task, the client code may have to combine the work of the CPU and the GPU, depending on the problem. Having done so, the client determines if it is finished with the problem; if it is not finished, it goes back to the first step, in which it rebalances and hands more data to the worker threads.

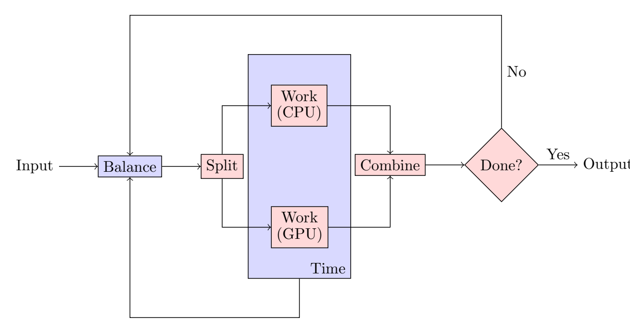



Fig. 1. Archon Workflow, with user code in red and Archon code in blue

In general, Archon's main task is to select a balancing factor $\alpha \in [0, 1]$, which represents a fraction of the workload to offload to the GPU. The remainder, $1-\alpha$, is left to the CPU. As the CPU and the GPU in Archon framework do their work in parallel, the execution time for an application is the greater of runtimes of the two subroutines. That is, $T = \max(T_c, T_g)$. Since $T_c$ increases as $T_g$ decreases, and vice-versa, the point at which their maximum is lowest will then occur when $T = T_c = T_g$. In order to solve this equality, we need to know how $T_c$ and $T_g$ grow as a function of $\alpha$. We can model this on the assumption that they grow linearly to $\alpha$; that is, we can imagine that there are runtimes $W_g$ and $W_c$, which are the total runtimes the problem would take if offloaded entirely to the GPU or CPU, and that $T_g = \alpha W_g$ and $T_c = (1 - \alpha)W_c$. This gives us the following:

$$T_g = \alpha W_g \tag{1}$$

$$T_c = (1 - \alpha)\, W_c \tag{2}$$

Armed with these equalities, we can then find $\alpha$ such that $T_c = T_g$. Note that $W_g$ and $W_c$ are not known at runtime. However, we can estimate them using the initial Equations 1 and 2, as follows:

$$W_g = \frac{1}{\alpha} T_g \tag{3}$$

$$W_c = \frac{1}{1 - \alpha} T_c \tag{4}$$

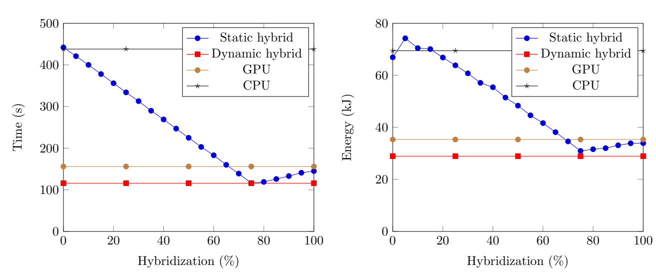
This allows us to find $W_g$ and $W_c$ using $T_g$ and $T_c$, which we can measure at runtime.

In Archon, equations (3) and (4) are then used as a guideline at runtime to fine-tune the distribution ratio between the CPU and the GPU. Specifically, Archon runs the computation for a short period of time (1 second by default), measures performance data, calculates the optimal distribution, and then adjusts the ratio for the next time interval.

## III. Experimental Results

To evaluate the effectiveness of our approach, we implement the classic problem of matrix multiplication using Archon. For each experiment, we run it in 4 settings: 1) Static hybrid, in which we use Archon with fixed pre-defined workload distribution ratio for CPU and GPU; 2) Dynamic hybrid, in which Archon dynamically tunes the hybridization ratio at runtime; 3) GPU, in which only GPU is used for the execution; and 4) CPU, in which only CPU is used.

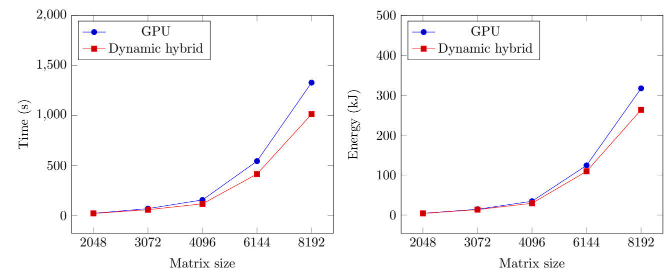
These experiments were run on a machine with an AMD FX-8350 CPU (running at 4.0 GHz), 32 GB RAM, and an nVidia GTX 660 with 2 GB GDDR5 RAM and 930 CUDA cores running at 1033 MHz.




(a) Performance (b) Energy Consumption

Fig. 2. Hybrid Execution Vs. Non-Hybrid Execution

The performance and energy consumption of a $4096 \times 4096$ matrix multiplication computation in the 4 set of experiments are shown in Figure 2. Note that for the static hybrid setting, we run the experiment 21 times with predefined distribution ratios spaced at 5% intervals, from 0% GPU to 100% GPU. As shown in Figure 2, in terms of performance, the static hybrid outperforms the pure-GPU execution for splits between 70% and 100%. Additionally, the dynamic hybrid experiment performs about the same as the best static hybrid setting. In terms of energy, the results are similar. These results show that the dynamic hybridization provided by Archon can effectively improve both the performance and the energy consumption without extra efforts from the users.




(a) Performance (b) Energy Consumption

Fig. 3. Scalability Evaluation

To evaluate the scalability of Archon, we run the matrix multiplication computation with various sizes, ranging from 2048 to 8192. The performance and the energy consumption of the Dynamic hybrid execution and pure GPU execution are shown in Figure 3. For relatively small problems, the performance and energy benefits of using Archon are limited. However, as the problem increases in size, the performance/energy gains grow as well; at the largest shown size, $8192 \times 8192$, hybridization drops us from 1328 s and 317 kJ to 1012 s and 263 kJ, for a speedup of approximately 24% and an energy savings of 17%.

## IV. Conclusion

With the growing popularity of using GPUs in general purpose computing, the energy consumption of these highly parallel processors has gained increasing interest. In this paper, we present Archon, a framework for optimizing the time and energy consumption of hybridized CPU-GPU applications. Archon takes a user's program as input, and dynamically tunes the workload distribution across CPU and GPU during runtime. We evaluated the effectiveness of this approach using a matrix multiplication application, and showed that Archon can greatly improve both performance and energy savings without extra efforts from the users. In addition, Archon is a generic framework that can work with any applications.

## References

[1] R. R. Schaller, "Moore's Law: Past, Present and Future," *Spectrum, IEEE*, vol. 34, no. 6, pp. 52–59, 1997.

[2] H. Sutter, "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software," *Dr. Dobb's Journal*, vol. 30, no. 3, March 2005.

[3] S. Cho and R. G. Melhem, "Corollaries to Amdahl's Law for Energy," *Computer Architecture Letters*, vol. 7, no. 1, pp. 25–28, 2008.

[4] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh, "Energy-Aware High Performance Computing with Graphic Processing Units," in *Workshop on Power Aware Computing and System*, 2008.