International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume 16, Number 1, pages 20–51, January 2026

Performance Models for Non-Uniform Heterogeneous Platforms and their Validation Using Structural Equation Modeling

Steven D. Harris
Dept. of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO, 63130, USA

Roger D. Chamberlain
Dept. of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO, 63130, USA

and

Christopher D. Gill
Dept. of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO, 63130, USA

Received: July 28, 2025
Revised: October 27, 2025
Accepted: November 23, 2025
Communicated by Masahiro Shibata

Abstract

Non-uniform heterogeneous computing architectures are increasingly common in modern computing platforms targeting mission-critical edge-cloud and embedded applications. The multiplicity and diversity of processor and memory characteristics they provide offers unprecedented opportunities for improving performance of those applications through customized mapping of platform resources to the applications' execution requirements. However, experience with these new platforms is limited compared to previous homogeneous multiprocessor platforms, and new analyses and empirical evaluations are needed to realize their potential more fully. In this paper we present principled analysis techniques and evaluate their effectiveness empirically using structural equation modeling methods in the context of the Orange Pi 5 platform, working towards new abstractions and coordination models to navigate the feature-rich landscape reshaping edge-computing and embedded systems architectures.

Keywords: asymmetric multiprocessing, ARM big.LITTLE, structural equation modeling

1 Introduction

In addition to common functions like data analysis and network processing, cyber-physical tasks—including sensing, perception, and real-time processing and control—increasingly target edge-cloud and embedded platform domains. Pairing these requirements with high demands for inference-capable devices has inspired domain-specific platforms replete with feature-rich component collections. Vendors including ARM (big.LITTLE), Intel (Power/Efficiency), Qualcomm (Kryo), and MediaTek (Dimensity), have all adopted heterogeneous chip designs to support edge-targeted workloads.

The execution flexibility offered by these platforms allows for multiple execution profiles depending on contextual goals (e.g., battery life, utilization, latency). Yet, unexplored levels of potential performance may be overlooked (e.g., due to a lack of time-intensive profiling). These designs also introduce new orchestration challenges due to the very execution flexibility offered by these emerging platforms. Where a task will execute may not be known a priori. Therefore, platform characteristics that dictate performance must be reasonably characterized, and their effects understood, in order to optimize consumption metrics (e.g., energy and utilization) and enforce constraints (e.g., deadlines).

Asymmetric multiprocessing platforms (AMPs) require radically different approaches to manage the compute diversity of this polymorphic architectural design space. In this paper, we use the Orange Pi 5 (an ARM big.LITTLE system) as a vehicle to explore these novel architectural topologies. By highlighting performance analysis findings on this platform we illustrate the need for new methodologies to navigate the feature-rich landscape reshaping edge-computing architectures, and provide initial results in that direction.

Choices about its execution profile, consisting of run-time settings (e.g., numbers and speeds of cores, execution schedules, dataset sizes) may impact the performance of any task. As opposed to uniform processors, identifying a feasible execution profile for an AMP, even without optimizing performance, requires additional analysis for each compute unit (core) type. This heterogeneity breaks the traditional uniformity of relationships between workloads and compute units: core heterogeneity augments our traditional notion of compute semantics, making simply defining the number of compute units of a given speed no longer sufficient. Instead, it is necessary to specify further details of each compute unit as a *type*. No longer is it simply a question of volume (number of cores) but also constitution (types of cores).

AMPs intrinsically require profiling, a multistage process, making design-space exploration for even "small" parameter sets time-prohibitive. Yet, there is structure underlying many workload-to-platform relationships that is understood in the homogeneous case and can be leveraged toward heterogeneous analysis. We posit that the workload-to-platform relationship in the case of AMPs is knowable and the structure applies to entire classes of platforms. Using the Orange Pi 5 as an example, we propose and evaluate performance models that exploit this underlying structure. Through a rational and principled investigation of this platform, we endeavor to answer the following questions for a given workload:

- How are the performance parameters characterized?
- Are there equivalence relationships between components?
- Can we classify the performance behavior?
- Can we find a model that captures platform behavior for accurate performance prediction?

One of the perennial challenges in developing models of this type is the fact that numerous assumptions are made during the modeling process. E.g., what parameters are relevant? what interactions are sufficiently substantial that they need to be included in the model?

To help address this issue, we turn to structural equation models (SEM), a set of data analytics techniques that represent the state of the art in multivariate statistical analysis. We use SEM as a tool to help us assess the assumptions made in developing the first principles performance model.

2 Background and Related Work

Given the variety of multiprocessor systems available, it is important to consider the inherent parallelism execution units deliver for high performance workloads at scale. The practice of mapping workload threads onto uniform cores can be considered as an instance of the knapsack problem [6, 33].

Unfortunately, there is no known polynomial-time algorithm that can determine if a valid workload mapping is optimal. Typically, heuristics can determine good (though possibly sub-optimal) configurations, however, and simply considering the behaviors of each device can be crucial to determine the most appropriate mapping. Workloads can be sensitive to changes in data types, data sizes, and (processor- or memory-) bounds on their execution. Consequently, a great deal of investigation into platform architectures has enabled characterization of performance impacts of specific design choices on workloads:

- Microarchitecture – the microarchitectural properties of a core clearly have a profound impact on performance assessment, as investigated in [1, 5, 8, 22, 25, 28], including instruction pipelining, cache hierarchy, branch prediction, and data path widths.
- Software orchestration – the heterogeneous multi-core landscape presents unique challenges in task orchestration and scheduling. Some of the more prominent challenges include: task mapping and granularity, dynamic workload management, overhead management, and security and isolation [23, 27, 29, 31, 35].
- Modeling approaches – the techniques used for performance modeling cover a wide range. Novel approaches suggested in [2, 7, 15, 16, 20, 21] include: reducing the cost of simulations, statistical parameter modeling, sampling, and application-specific modeling.

Elaborating on some of the literature cited above, Pagani et al. [27] emphasize the importance of a management layer between the system software and the acceleration platform that takes task-specific considerations into account to facilitate shared-memory communication, improve hardware reconfiguration, and support scheduling.

Shirazi et al. [29] identify that the numbers of instructions for workloads are increasing and particular operations in the workloads take considerable time. Instead of using generic execution units for these work-intensive instructions, it would be worthwhile to architect custom execution units that exclusively operate on these sub-tasks of the workload.

Trimberger et al. [31] clearly outline how the physical hardware is the very first line of defense for securing any platform and expose several weaknesses and fortifications that impact platform security. Yet, few if any of the considerations are taken into account for the impact that each design choice may have on performance, scheduling, and timing behavior.

Yue et al. [35] consider the affinity of workloads for available execution units. Instead of tailoring a platform to a specific workload, the authors seek to understand how the workload operates on the given platform and classify workloads according to their affinity to the specific processor features and capabilities. Affinity becomes a necessary gauge for identifying the appropriate platform with a given set of constraints.

Moving from classification schemes with emphasis on real-valued performance metrics (e.g., execution time, IPC, cache-misses, etc.) to qualitative performance scales (e.g., capable/incapable, slow/fast) supports resource planning and workload scheduling. Given the pervasiveness of energy-constrained computing environments at the edge, the identification of workload affinity becomes ever more important. Obtaining sufficient measurements in a polymorphic design space can be an intractable problem, but regression modeling as proposed in [21] is a powerful tool to assess performance accurately for arbitrary processor configurations in large micro-architectural spaces.

It also is becoming popular to leverage statistical inference and machine learning for approximating solutions in large intractable design spaces. Gaining traction in computer performance modeling, statistical models have been shown in [15] to detect trends in performance metrics, which can be used to establish baseline performance, automate anomaly detection, and discover sources of bottlenecks. These findings suggest that any behavior both undesirable and constrained is reflected by the platform performance data.

Methods to explore the exponentially increasing design space efficiently remain an open problem. As supported by the work of [16], design space exploration can be accelerated by identifying important design parameters, utilizing targeted sampling, and leveraging fast and accurate predictions afforded by artificial neural networks and other structures. Such methods applied to these design spaces demonstrate broad applicability to relevant workload modeling challenges.

The tasks we consider here are compact (e.g., the empirical results we present are for a matrix-matrix multiply kernel). There has been considerable work in modeling the performance of applications that are composed from compact tasks. For streaming data applications, queuing theory [10], flow analysis [3], and network calculus [9] have been applied to heterogeneous systems.

Looking specifically to the ARM big.LITTLE platform, Wang et al. [32] develop a model for convolutional neural network inference that they then use to guide placement of tasks to both types of cores. Zhu and Reddi [36] use statistical inference to model mobile web browsing performance with the goal of meeting latency constraints while consuming the least amount of energy. Butko et al. [4] developed a simulation model of the ARM big.LITTLE platform that predicts both performance and power consumption.

3 Hypotheses

A prerequisite to finding a satisfactory workload mapping configuration is to identify the appropriate compute units. In the homogeneous case, this is simply an integer value. In the case of AMPs, a vector of settings (number, type, power-level) may be required. To find those settings, an investigation into workload-to-platform interactions is needed, to identify and characterize platform behaviors. The desired settings then can be selected to induce workload behaviors that meet contextual performance goals. Our experiments indicate that a rich dense performance topology exists for different classes of devices, which we can leverage for orchestration. We state several hypotheses regarding our AMP investigation:

There is discoverable structure in workload-to-platform relationships
: We posit that workload behavior will naturally cluster into groups of similar metrics partitioned by both the type and number of compute units (i.e., *configurations*) including overlapping performance wherein different configurations of settings exhibit similar performance characteristics for a given workload.

There are equivalence relationships across components
: We expect that the platforms' diverse execution options offer more than one configuration that is capable of meeting the workload requirements. Therefore overlapping performance configurations will imply the existence of component equivalence relationships.

Existence of a generalizable model $\overrightarrow{P} = F(\overrightarrow{A}, \overrightarrow{H}, \overrightarrow{S})$
: We assume that the relationship between platform components and workload parameters can be modeled rigorously. We propose a model to measure performance $\overrightarrow{P}$ (e.g., execution time, throughput, energy usage) that is a function of: an application vector $\overrightarrow{A}$ with given elements (e.g., problem size, cache locality, etc.); a hardware platform $\overrightarrow{H}$ described by a vector (e.g., ISA, cores, memory properties, microarchitecture properties, etc.); and system software (e.g., scheduler, thread pool, etc.), traditionally managed by the operating system and denoted by the vector $\overrightarrow{S}$.

Existence of the inverse model $(\overrightarrow{A}, \overrightarrow{H}, \overrightarrow{S}) = F^{-1}(\overrightarrow{P})$
: Given the generalizable model, we suppose the existence of an inverse of that model. The inverse function taking the performance metric as input, should provide application, hardware, and system software configuration mappings meeting the input performance criteria. A valid inverse relationship will be a powerful discrimination tool, enabling categorization by run-time performance, and providing a selection of configurations meeting the performance requirements. Note that strictly speaking, while we use functional notation, this relationship is not necessarily a function, supporting the potential for multiple distinct outputs given the same input.

In the next sections, we describe a methodology and show empirical results to evaluate the hypotheses put forth above. This includes the development of a first principles model of F. We then follow that with the use of structural equation modeling to assess the assumptions made during the development of the first principles model.

4 Methodology

Single board computers are promising foundational devices through which to explore and understand the complexities of emerging architectures. These devices highlight a realistic range and capacity of platforms that may be found in practice for many industry applications.

In this paper we consider the Orange Pi 5 architecture, which is representative of important capabilities of platforms operating at the edge. We use the Orange Pi 5 as a baseline frame of reference for investigation into the larger AMP design space.

4.1 Orange Pi 5 Configurations

The Orange Pi 5 offers a variety of compute nodes ranging from high-performance ("Big") cores to energy-efficient ("Little") cores, each with unique performance implications. As shown in Figure 1, the platform utilizes a Rockchip RK3588S big.LITTLE architecture comprised of two physical (Big & Little) islands, with private L1/L2 caches, and one logical (Dual) island composed of at least one core from *each* physical island. All cores are connected via the last-level-cache. A few relevant performance variables and associated components are listed in Table 1 offering a plethora of configurations.

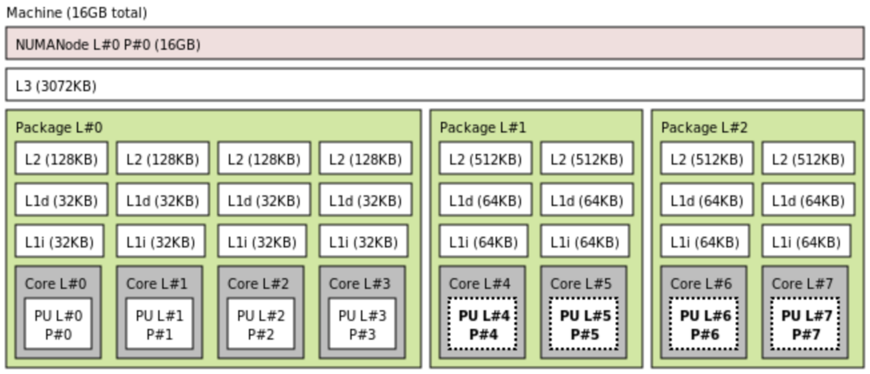


Figure 1: Orange Pi 5 Core Configuration

Table 1: Orange Pi 5 Specifications

Variables	**Performance Components**
Number of Cores	ARM Quad-core A76 + Quad-core A55
Island Selection	High-Performance (Big) + Energy-Efficient (Little)
Core Frequencies	2.4 GHz down to 1.8 GHz
Power Levels	Conservative, Power Save, Performance

4.2 Workload

To exercise the platform, we selected a matrix multiply workload of the form: $C \leftarrow \alpha AB + \beta D$, where $C, D \in \mathbb{R}^{M\times N}$, $A \in \mathbb{R}^{M\times K}$, and $B \in \mathbb{R}^{K\times N}$. The workload, authored in C++, is a common OpenMP parallelized "three-loop" naïve matrix multiply implementation [12]. It does not make use of data reuse patterns (in-cache) or other optimization which leverage prior hardware knowledge. In that respect, the workload reflects a common, general matrix multiply (GEMM) computation, parallelized with OpenMP for threads in the range $\{1, ..., n\}$, where n is the maximum number of cores. The dimensions ($M = K = N$) of the square matrices are in the range $\{64, 128, ..., 8192\}$. This workload is representative of computationally intensive requests at the edge, as it is an essential component of many machine learning applications.

4.3 Performance Model

Our goal is to develop a performance model from first principles, examine how well that model fits the data, refine the model as needed, and validate the assumptions that are present in the model via structural equation modeling techniques. To put it in the perspective of Section 3, our performance vector $\overrightarrow{P}$ may be a simple metric such as execution time, T_e; our hardware vector $\overrightarrow{H}$ is (N_B, N_L), the number of Big cores and the number of Little cores; our application vector $\overrightarrow{A}$ is the matrix dimension, $M = K = N$ (we will use M in what follows); and the system software $\overrightarrow{S}$ entails one thread for each core.

As such, a few initial relationships are readily determined. First, since our application is matrix-matrix multiplication, we anticipate the work to be accomplished, W, is cubic in the size of the matrices, M.

$$W = a_3M^3 + a_2M^2 + a_1M \tag{1}$$

Here we are saving a scaling constant and assuming work is in units of time on the executing processor. As the model evolves, it will be constrained to units of work on a Little processor.

Given the excellent parallelization of the matrix multiplication application, this can then yield

$$T_e = \begin{cases} W/N_L & \text{if Little} \\ W/N_B & \text{if Big} \end{cases} \tag{2}$$

when executing exclusively on Little cores or Big cores.

We quickly observed, empirically, that the execution time on Big cores was very close to a constant fraction of that on Little cores. The next piece of the model attempts to exploit this fact.

Essentially, we would like to predict platform performance for a range of configurations based on the characterization of the smallest execution unit, in this case, one Little core. Expressing the composite number of cores as the sum of Little cores plus scaled Big cores gives us:

$$N_C = N_L + \frac{N_B}{R_{BL}} \tag{3}$$

where R_{BL} is a Big-to-Little execution time ratio.

Returning to Equation (2), and substituting the composite number of cores for either N_L or N_B, this can then yield

$$T_e = \frac{W}{N_C} \tag{4}$$

which, when expanded out, gives

$$T_e = \frac{a_3M^3 + a_2M^2 + a_1M}{N_L + \frac{N_B}{R_{BL}}}. \tag{5}$$

Our second empirical observation is that this Big-to-Little execution time ratio, while reasonably effective, does depend upon the workload. One approach to dealing with this variability is to condition the ratio on the working set size, S_{WS}, e.g.,

$$R_{BL} = \begin{cases} R_{BL.Low} & \text{if } M < S_{WS} \\ R_{BL.High} & \text{if } M \geq S_{WS} \end{cases} \tag{6}$$

where the effective working set size transition is empirically determined. We will see an application of these techniques in the experimental section to follow directly.

Table 2 summarizes the notation used throughout the paper.

Table 2: Notation and Description

Symbol	Description
$a_i, \quad 1 \leq i \leq 3$	Cubic coefficients
$\overrightarrow{A}$	Application vector; (M)
$\overrightarrow{H}$	Hardware vector; (N_B, N_L)
M	Matrix dimension
N_B	Number of Big cores
N_C	Number of composite cores
N_L	Number of Little cores
$\overrightarrow{P}$	Performance vector; (T_e)
R_{BL}	Big-to-Little ratio
S_{WS}	Working set size
T_e	Execution time
W	Application work

5 Experiments

5.1 Parameters: Island + Matrix Dimension

The first set of experiments can be viewed as a coarse-grain workload-to-platform assessment. Starting with island selection, we iterate through all available configuration options to identify the performance implications for the workload based on parameter (island, matrix dimension) selection.

Figure 2 illustrates workload execution time by matrix dimension. Colored by island selection (Big, Dual, Little), the results indicate that Little cores have the longest execution time, with Big and Little combinations (i.e., Dual island) taking up the middle ground, followed by Big cores having the shortest execution time. Notice that many of the data points overlap for different island selections.

Coarse-grained analysis is a wonderful tool for orienting the performance of a platform. However, once oriented, further exploration is required in the direction of the performance goals. These introductory experiments confirm that there is a structure to the workload-to-platform performance. If the overlap is valid, it also suggests equivalence relationships between configuration components.

5.2 Parameters: Island + Matrix Dimension + Cores

Contrasting the coarse-grained analysis of the two parameters as shown in Figure 2 is an example of a finer-grained analysis shown in Figure 3, utilizing additional parameters to include the number of cores (and threads, though to limit the data density we only present cases where the core counts match thread counts). The additional parameters provide clarifying insights on the prior performance assessment parameters.

To ease configuration differentiation, each set of experiments is labeled using the format: **Island[Cores,Threads]**. For example, the lowest performance configuration occurs for workloads utilizing one core and one thread executing on any Little island core (i.e., Little[1,1]).

With execution time on the y-axis and the matrix dimension on the x-axis for Figure 2 and Figure 3, each point represents the empirical workload execution time. In Figure 3, a cubic trendline is fit to each configuration, indicated by solid lines (i.e., the coefficients a_i in Equation (1) are separately determined for each configuration). In all cases, the measurements fit the cubic line regression function with an R-squared value of 0.9989, comfortably close to a perfect fit. This function is consistent with the run-time for a standard (3-loop) matrix multiplication computation.

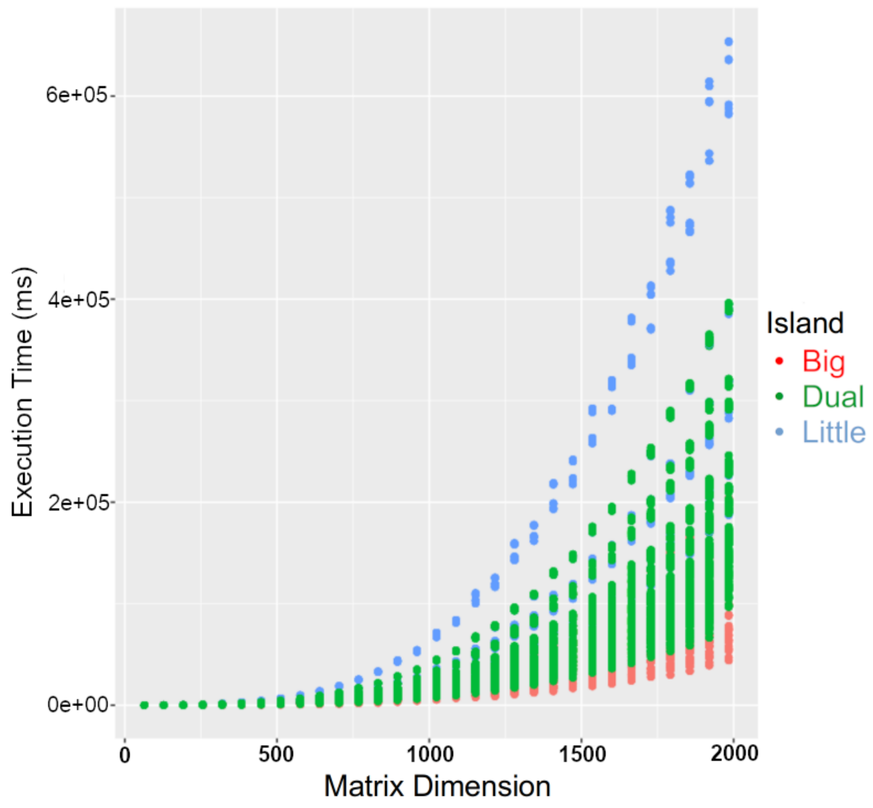


Figure 2: Orange Pi 5 Matrix Multiplication by Island

We see, clearly and as expected, that matrix multiplication has a cubic execution time relationship with the workload.

More interesting is that Figure 3 also exhibits substantial performance overlap between distinct configurations, pointing to an underlying fundamental structure to the platform performance: this is in direct support of the first and second hypotheses articulated in Section 3. We then want to quantify, classify, and order the discovered performance. To do so, we will normalize performance to the smallest compute granularity (i.e., minimum workload performance) on the platform to use as the basis for measurement in the discussion below.

5.3 Performance Normalization

Given the observed performance, we denote a single-threaded Little core as the finest compute granularity. The Little core hardware capabilities referenced in Figure 1, along with the demonstrated performance as shown in Figures 2 and 3 support this choice for smallest compute granularity. Next we scale the performance of each configuration to that of one single-thread Little core, reflected in Table 3. With respect to the smallest compute granularity, each configuration has a speedup, an average ratio (across all matrix dimensions) of target configuration execution time to the reference execution time (Little[1,1]), and the standard deviation of this ratio. The speedup is linear as core count increases, and is relatively fixed as we transition from a Little island to a Big island. Little cores have less variability across matrix dimensions as opposed to their Big core counterparts, as reflected in the standard deviation (SD). Supporting the notion of normalization to Little core configurations, the SD is at-least one order of magnitude smaller than any Big core configuration.

At this stage, in lieu of a comprehensive series of statistically significant measurements and analysis confirming the relationships, let us assume that the performance trends are valid (we will re-examine this assumption below). This is a simple assumption often made at run-time for orches-

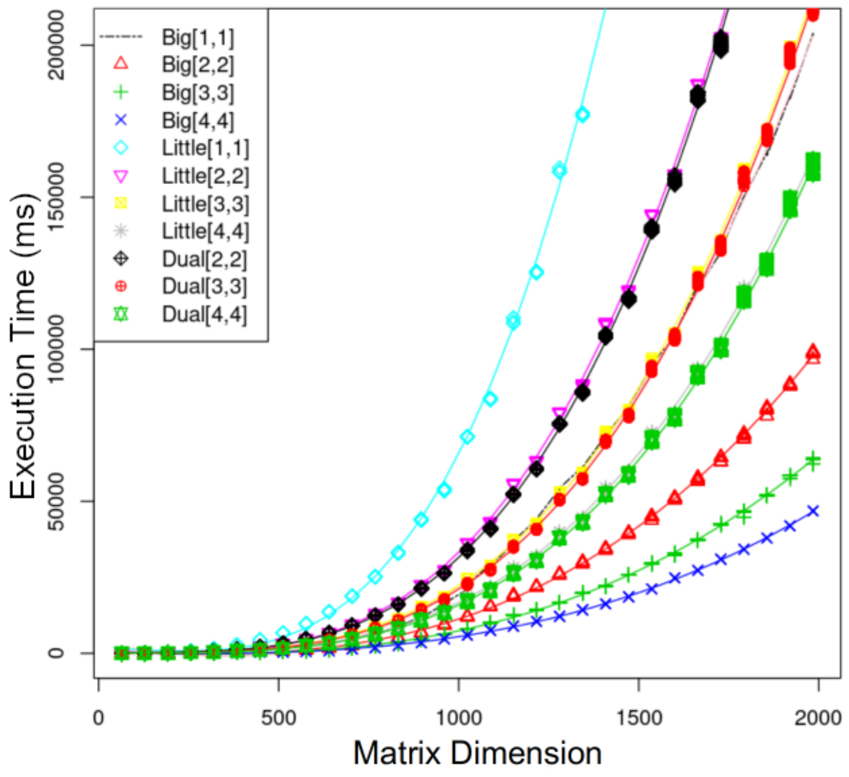


Figure 3: Orange Pi 5 Matrix Multiplication by Configuration

tration decisions. Given the configuration relationships, we should be able to model the execution requirements in terms of the matrix dimension (fit to a Little[1,1] configuration, i.e., using the coefficients a_i from that individual configuration), the number of cores, and the Big-to-Little ratio, utilizing Equations (1), (3), and (4). The predictive modeling is shown in Figure 4 with the measured execution times presented as data points and each line showing the modeled (predicted) time for the configuration. The presentation is reversed for Big[1,1] with actual values shown as lines and predictions denoted with data points to differentiate the overlapping performance trends (with Little[3,3]).

Our first observation is that, for many purposes, this model does a good job of predicting the performance across all configurations tested. While there is some separation between modeled and measured results for large problem sizes on the speediest configurations, overall it does a good job.

Looking a bit deeper into the set of configurations that don't match the model so well, the obvious prediction deviations are readily observable for the Big island multi-core configurations. Notice in Figure 4 that for Big[2,2] through Big[4,4] the predictions fall below the actual execution time. In contrast, notice that the single core Big island configuration (i.e., Big[1,1]) oscillates between over- and under-prediction, suggesting a parameter dependent performance change across the matrix dimensions. We investigate these discrepancies next.

5.4 Parameter Characterization

The prediction results in Figure 4 and the normalized ratios calculated in Table 3 indicate a measure of data variability. It can be challenging to see from this vantage point how the variation is impacting our predictive ability. Thus, we need to expand our perspective and reorient the data. We will change perspectives and look at the normalized Big-to-Little ratios graphically.

Figure 5 shows the Big-to-Little ratio (actually, the ratio of execution time for all Big and Little configurations relative to Little[1:1]) as a function of the matrix dimension. This plot reveals a

Table 3: Big-to-Little Normalized Ratios

Configuration	Speedup	Average Ratio	Standard Deviation
L[1,1]	100%	1	0
L[2,2]	200%	0.50	0.010
L[3,3]	300%	0.33	0.005
L[4,4]	400%	0.25	0.004
B[1,1]	300%	0.32	0.089
B[2,2]	600%	0.15	0.022
B[3,3]	1000%	0.10	0.029
B[4,4]	1400%	0.07	0.016

hidden performance nuance that would have otherwise escaped our attention. The normalized ratio for Big[1,1], has distinct ratio intervals depending on matrix dimension. There appear to be stable performance ratios for intervals of matrix dimensions between approximately (300, 800) and (1200, 1700). For the sake of discussion, we could base an assessment around the inflection point occurring at matrix dimension 1000. We see a performance dip in other Big core configurations as well, though the effect seems to diminish with increasing core numbers. These results suggest that proper performance prediction will require normalization over subset intervals of workload parameters to maintain performance portability.

To ground this idea, we again normalize all of the experiments to the lowest execution unit performance (i.e., Little[1,1]). If our assumptions are correct, we should be able to predict the performance of other configurations by simply using the Big-to-Little performance ratio, denoted R_{BL}. However, the ratio is no longer assumed to be constant across the entire range of matrix dimensions. As illustrated in Figure 5, the speedup for all Big cores varies across matrix dimensions. This is particularly pronounced for the Big core configuration using one-core and one-thread (i.e., Big[1,1]), where the behavior only begins to stabilize around matrix dimension 300, but shifts again around matrix dimension 1000. Initially, the performance for the Big[1,1] configuration is concave. Around matrix dimension 1000, there is an inflection point shifting performance to a convex profile. Our model accommodates the different behavior before and after this point, with the ratio being conditional on the matrix dimension, e.g.,

$$R_{BL} = \begin{cases} R_{BL.Low} & \text{if } M < 1000 \\ R_{BL.High} & \text{if } M \geq 1000. \end{cases} \tag{7}$$

Note that this is simply a re-expression of Equation (6) with an empirically determined threshold of 1000. This analysis depends on profiling the performance behavior across a wide range of configurations and workloads. We hypothesize that while additional workload parameters may improve the accuracy of our model, a smaller set of key parameters may be sufficient to characterize and predict system performance. In Section 5.5 we explore a few of the system parameters to understand which components give clarifying insight into the dynamic platform behavior.

5.5 System Parameter Correlation

We continue our investigation by probing lower-level interactions to identify parameters that are critical to effectively modeling platform performance. Using a wide range of metrics garnered during execution from the Linux performance measurement tool `perf`, we investigated the pairwise-correlations between a number of metrics. The results presented above suggest that the matrix dimension plays an important role in the overall performance, but are there other predictors within the platform that support this notion?

For each configuration, we took samples of `perf` metrics to understand the overall behavior during workload execution. Looking at the pairwise correlations illustrated in Figures 6 and 7, we can see clearly that the correlations change dramatically across workload parameters. Notice in

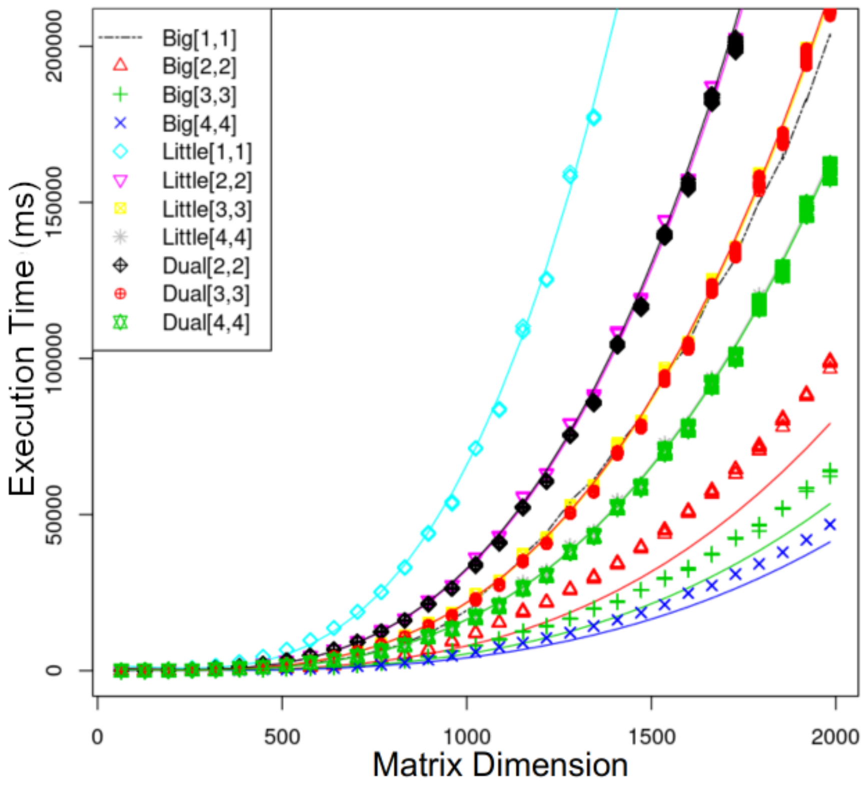


Figure 4: Static Big-to-Little Ratio

Figure 6 that the L1 data cache miss-percentage is negatively correlated with cycles, instructions, ipc, and L1 data cache metrics (e.g., loads and misses). This combination of negative correlations suggests data starvation occurring at the core level. All instruction related metrics rates would decrease as they wait on data to load into the L1 data cache.

If we are correct in our starvation assumption, the correlation should change as we increase the amount of available data. Increasing the workload size (i.e., matrix dimension) supports data availability to the cores. As we look to the larger matrix size in Figure 7, the negative L1 data cache miss-percentage correlation with cycle rate, cycles, instructions, ipc, and L1 load and miss metrics improves. Notice that negative correlation pivots from L1 data cache miss percentage metrics to context switch and Last-Level-Cache (LLC) Load metrics. While negative correlation with L1 caches persists, notice that for the larger size matrix dimension the L1 data cache miss percentage is actually improving. This suggests that our starvation bottleneck has moved above the core level (beyond L1/L2 caches) to the LLC. While far from definitive, the above supports our suggestion that the conditional in Equation (6) referencing an empirically determined working set size, denoted S_{WS}, is a reasonable notion.

5.6 Re-parametrization

The data clearly suggest that the performance varies with the workload size. Therefore, by setting a threshold, we can condition on the matrix dimension in a coarse-grained fashion as depicted in Equation (7), where we partition the performance into stable regions. Instead, suppose that we went a step further and created a Big-to-Little ratio as a function of the specific matrix dimension M, denoted $R_{BL.M}$, and Equation (3) becomes:

$$N_{C.M} = N_L + \frac{N_B}{R_{BL.M}} \tag{8}$$

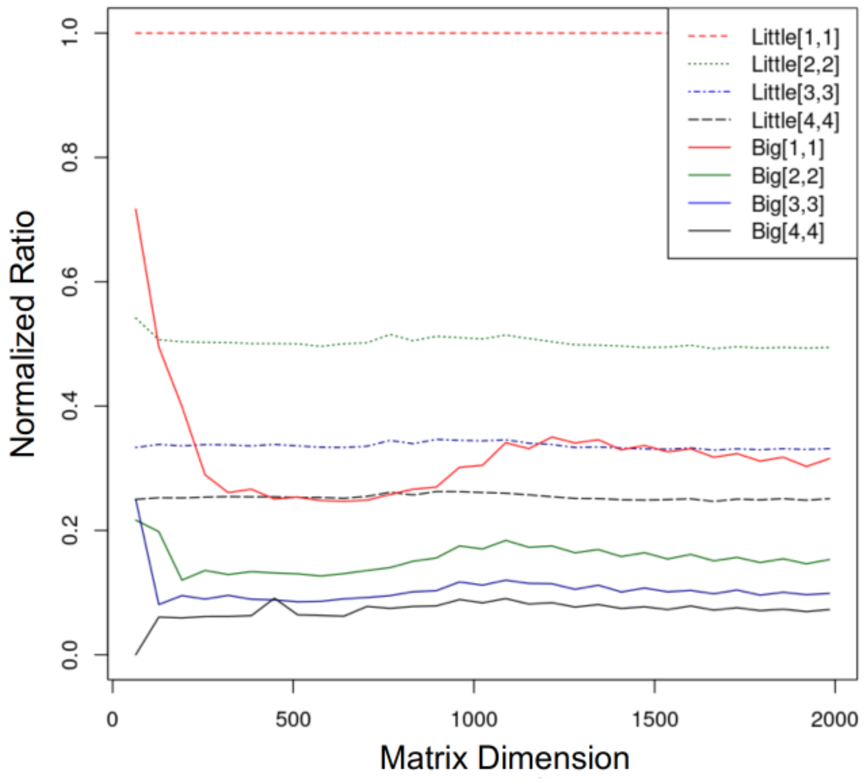


Figure 5: Normalized Execution Time Ratio

and Equation (4) becomes:

$$T_e = \frac{W}{N_{C.M}}. \tag{9}$$

Thus, using Equation (9), we are able to calculate and predict performance accurately using the fined-grained Big-to-Little ratio for all independent configurations. As there may be some confounding behavior using the joint (dual) cores, we will not perform that analysis in this work. As depicted in Figure 8 each data point reflects actual experimental values with lines being used to predict the performance using the re-parameterized Big-to-Little Ratio. The exception is in the case of the Big[1,1], where the actual data is a presented as a line, and the prediction is a data point. The line travels exactly through the center of each data point. We reversed plotting for Big[1,1] to distinguish between the overlapping performance with Little[3,3] and to illustrate how closely the predicted mappings follow the actual data.

Finally, taking the mean ratio across all matrix dimensions gives rise to the predictions shown in Figure 9. The distinction between these two graphs being that in Figure 8 the predictions follow the empirical performance precisely, given a fine-grained Big-to-Little ratio, while Figure 9, using the mean ratio, is an example of the course-grained approximation.

5.7 Can we validate the behavior?

The results are indicative of a clear relationship between the workloads and architecture. In the case of the Orange Pi 5, we found that there is performance overlap between combinations of Big and Little cores. Particularly, we have found that we can model the mean execution time for our workloads on one Big core as equivalent to that of three Little cores. Using the Big-to-Little ratio calculation, we identified a pair of configurations suggesting equivalent performance, particularly, the L[3,3] and the B[1,1] as shown in Table 4.

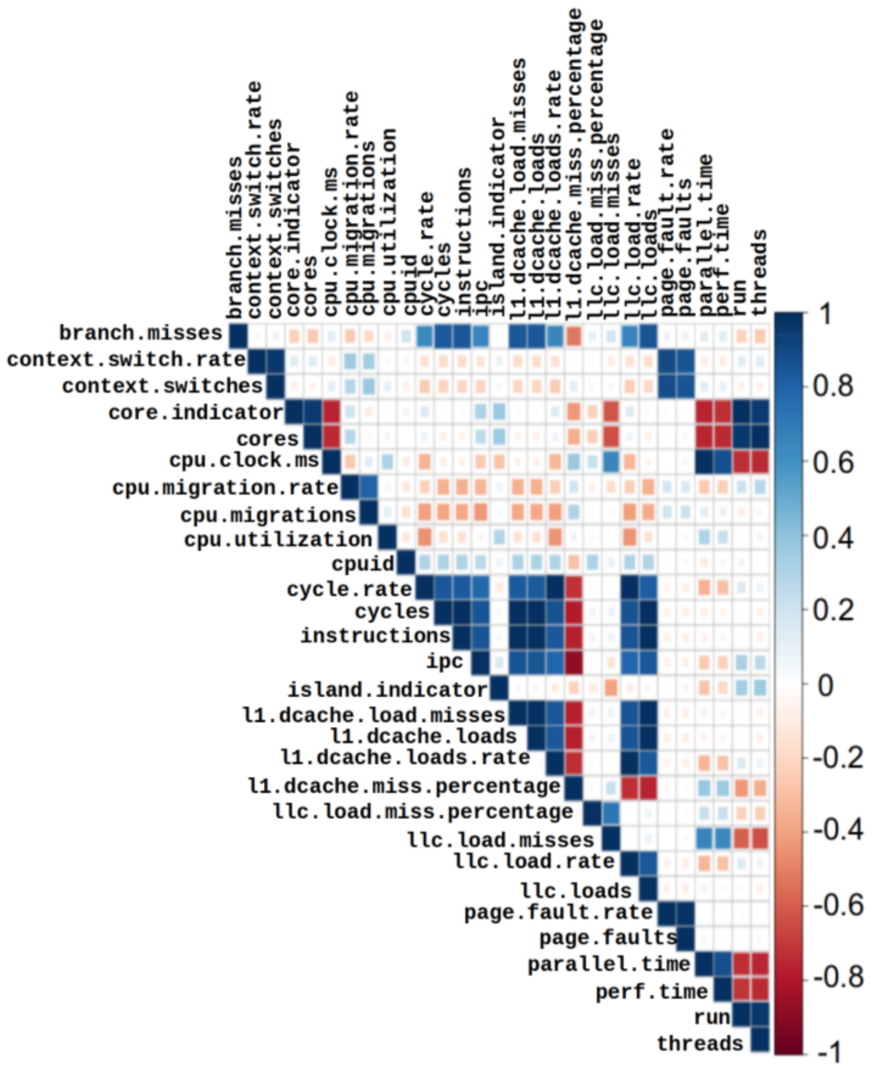


Figure 6: Correlations – 768 × 768 Matrices

Next, we wanted to determine if the mean execution time for one Big core and three Little cores are statistically different. Consequently, we perform a Welch two-sample t-test on the two means with the alternative hypothesis that the true difference in means is not equal to zero. Those results are shown in Table 5, showing the (P)aired and (U)npaired results, respectively. Both the paired and unpaired tests show significant results supporting the conclusion that the execution times are not statistically different from one another.

The above validation, however, is quite limited. While the agreement between the model and measured results is quite good, there are many assumptions that have been made in formulating the model for which we have, at best, only intuition to guide us. To help us assess the validity of these assumptions, we turn to a family of statistical analysis techniques called structural equation modeling.

6 Structural Equation Modeling

Structural equation modeling (SEM) techniques are widely used in social sciences, psychology, education, and business [11, 14]; however they are less frequently encountered in traditional engineering

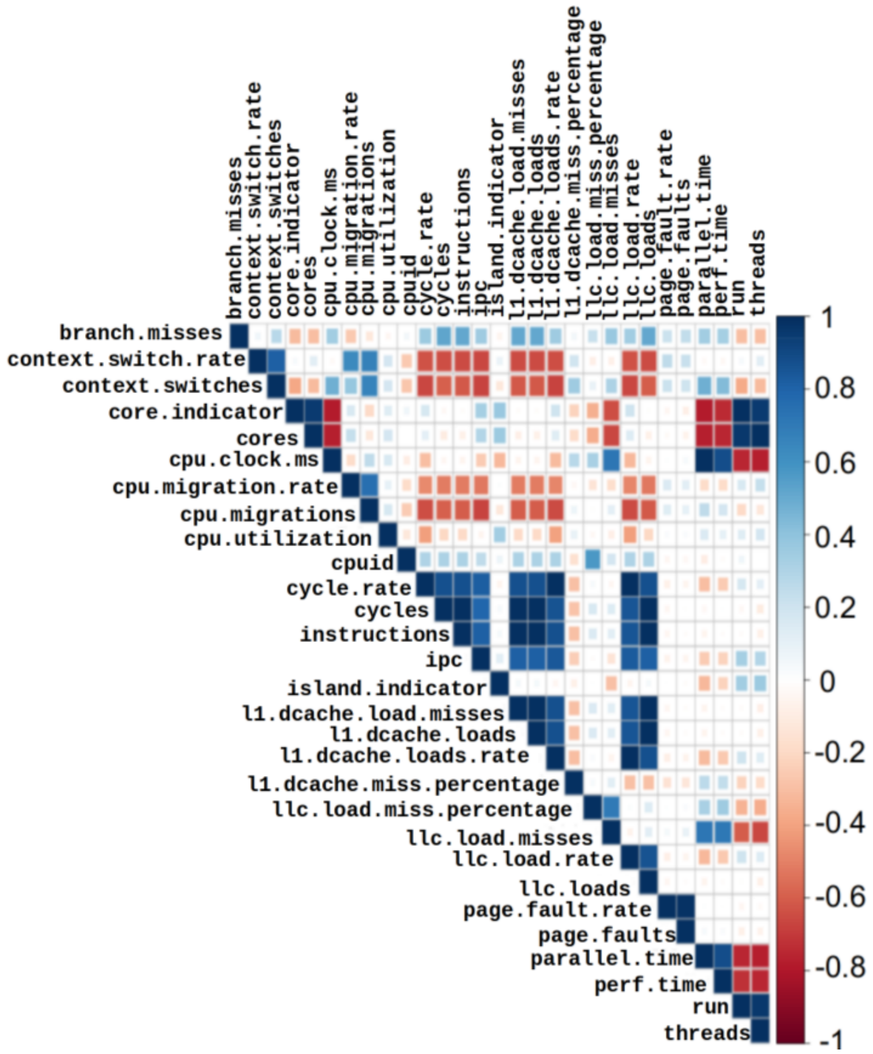


Figure 7: Correlations – 1216 × 1216 Matrices

models, which are dominated by first principles models (e.g., of the type described above in Sections 4.3 and 5.6). We will exploit some of the capabilities of SEM to assess the validity of the assumptions made in the development of the first principles model presented above.

6.1 Introduction to Structural Equation Modeling

Structural equation modeling is a powerful and flexible multivariate statistical framework that is used to analyze complex relationships between variables [13, 19]. SEM combines aspects of several traditional statistical methods, including path analysis, confirmatory factor analysis (CFA), and multiple regression, into a single cohesive approach and methodology to describe and articulate complex hierarchal structures.

SEM is focused on variations in, and relationships between, quantities themselves (e.g., variance, covariance, correlation, autocorrelation, etc.) rather than for example the means of those quantities. As described by Hoyle [14], often in SEM the goal of estimation is "the minimization of the difference between the observed covariance matrix and the covariance matrix implied by the model." This is a clear distinction from models (such as the one we presented above) which are trying to, e.g., minimize the sum of squared differences between model output and empirical observation. As such, we will not be using SEM to *replace* the model presented above. Instead, we will be using SEM to

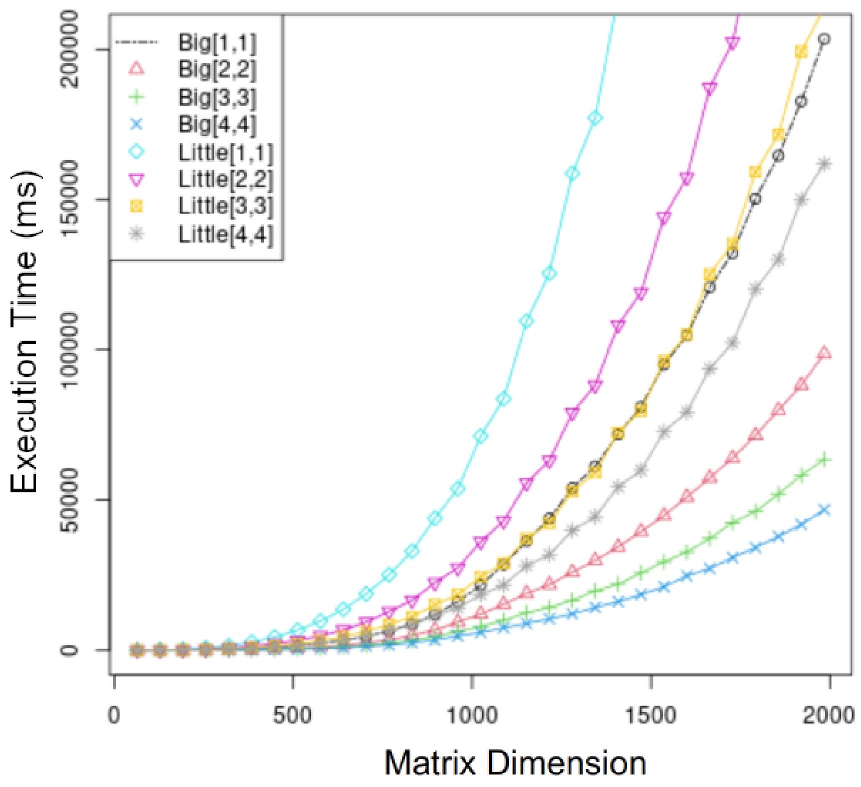


Figure 8: Re-parameterized Big-to-Little Ratio

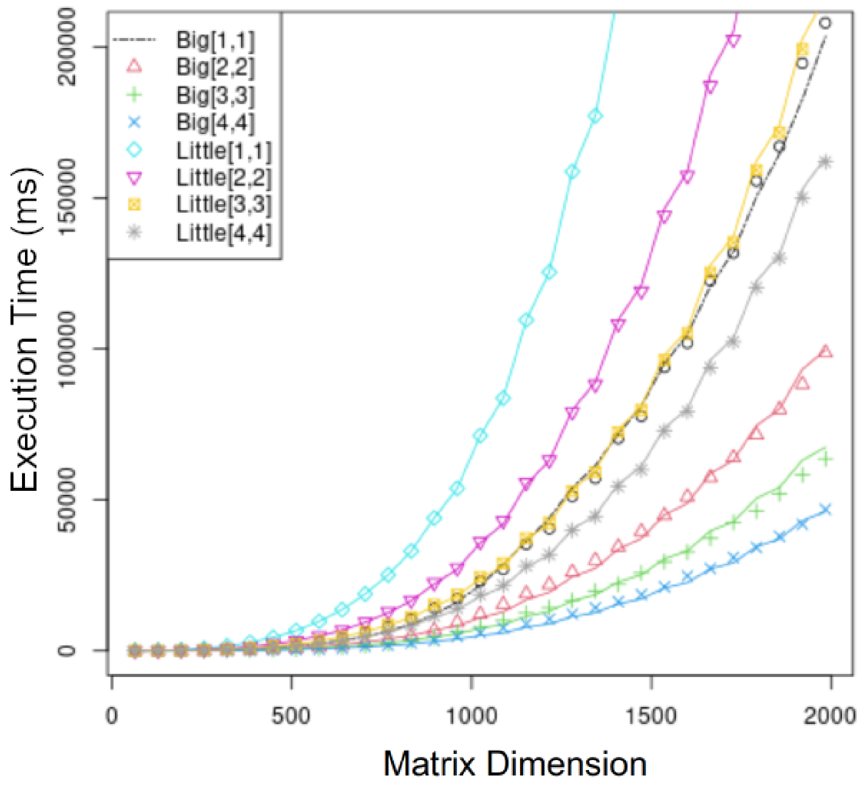


Figure 9: Complete Model

Table 4: Big-to-Little Normalized Ratios

Configuration	Speedup	Average Ratio
L[3,3]	300.00%	0.3361
B[1,1]	300.00%	0.3225

Table 5: Big-to-Little Normalized Ratio Tests

Configuration	t	df	p-value
(P) Big[1,1] $\approx$ Little[3,3]	5.4069	31	6.687e-06
(U) Big[1,1] $\approx$ Little[3,3]	5.4069	31	6.687e-06

assess the appropriateness of the assumptions that are inherent in it.

Some key characteristics of SEM:

- **Modeling Complex Relationships:** SEM gives researchers the flexibility to test hypothesized causal relationships among a set of observed and unobserved (latent) variables simultaneously. It can model direct and indirect effects, and accommodate situations where variables serve as both predictors and outcomes.
- **Latent Variables:** One notable distinguishing feature of SEM is its ability to incorporate "latent variables." These are theoretical constructs or concepts that cannot be directly measured (e.g., intelligence, customer satisfaction, motivation). On the contrary, they are inferred from multiple directly observed "indicator" variables (e.g., scores on different tests, survey questions, etc.). SEM explicitly accounts for measurement error in these observed indicators, providing more accurate estimates of the relationships between the underlying latent constructs.
- **Graphical Representation:** SEM models are often represented visually using path diagrams (similar to flowcharts). Circles and/or ovals typically represent latent variables, squares or rectangles represent observed variables, single-headed arrows indicate hypothesized causal effects, and double-headed arrows indicate correlations or covariances. This visual representation is elaborated upon in Section 6.2 below.
- **Model Fit Assessment:** A critical aspect of SEM involves evaluating the "goodness-of-fit" for the hypothesized model to the observed data. There are various fit indices (e.g., Chi-square (χ^2), Root Mean Square Error of Approximation (RMSEA), Comparative Fit Index (CFI), Tucker-Lewis Index (TLI)) that are used to determine how well the proposed model accounts for the relationships (covariances) observed in the dataset.

Essentially, SEM assists researchers in confirming or falsifying theories by testing how the relationships among variables, including abstract concepts, align with the empirical data. In our case, we will use SEM measurement models to examine the parameter selections made in our first principles model above and then follow that using SEM structural models to examine the refinement described in Section 5.6.

6.2 Graphical Representation Definitions

6.2.1 SEM Path Elements

With regard to the graphical summaries provided by SEM as depicted in Figure 10, there are several fundamental elements that we outline as follows:

- **Squares or Rectangles** (□): Represent observed variables (also called manifest variables, indicators, or measured variables). These are variables for which we have direct data (from measurements, experiments, or literature).

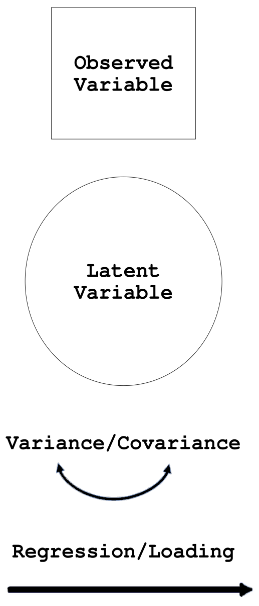


Figure 10: SEM Path Elements

- **Circles or Ovals** (○): Represent latent variables or error terms (i.e., residual variance). In contrast to observed variables, latent variables are constructs that cannot be measured directly but are inferred from *observed variables* with the *error terms* accounting for *leftover variance* not explained by other measurements and variables.
- **Single-Headed Arrows (→)**: Represent *regression paths* or factor loadings, also known as *direct effects* which are used to indicate a hypothesized causal relationship or a directional influence from one variable to another.
- **Double-Headed Arrows (↔)**: Are used to represent covariances or correlations within individual SEM elements (i.e., error terms) or across elements (i.e., between latent variables), but the relation does not imply a direct causal direction.

6.2.2 SEM Path Diagrams

Graphical representations of SEM models are powerful tools for both defining and analyzing complex relationships between variables. SEM is unique in that it combines aspects of factor analysis and regression analysis into one single analysis allowing for the estimation of multiple, potentially interrelated dependent variables and structures. It consists of two main parts: the measurement model and the structural model.

- **Measurement Model**:

  The measurement model in SEM defines how latent variables (unobserved constructs) are measured by their observed variables which is used to assess the validity and reliability of the measurement of these latent constructs. Essentially testing whether the observed variables adequately capture the underlying latent variable they are intended to represent. Confirmatory Factor Analysis (CFA) is the most common method used to test a measurement model. It confirms whether the observed data fits a pre-specified theoretical model of how indicators load onto latent factors.

- **Structural Model**:

  The structural model in SEM defines the hypothesized causal relationships and direct/indirect effects between the latent variables themselves to test theoretical relationships and pathways among these unobserved constructs by examining how changes in one latent variable are hypothesized to cause changes in another. After a measurement model's validity is established, the structural model is tested to see how well the hypothesized relationships among the latent variables fit the observed data which often involves examining path coefficients, R-squared values, and overall model fit indices.

- **Relationship Between Measurement and Structural Models**:

  The full relationship between the measurement and structural model reveal a deep level of interconnected components. As depicted in Figure 11, the SEM Path Diagram shows a regression model wherein the X is being regressed on latent variable Y. Notice that the latent structure X (an endogenous construct) has three directly measured (exogenous) factors (i.e., x_i's) each having a particular loading factor which numerically relates the impact and connection between the latent variable (X) and the observed variables (x_i's), comprising a measurement model for X. Contrasting this, we have a similar measurement model for Y with its associated factors (y_i's). The structural model, illustrated in the center, facilitates identifying the relationship between the two latent constructs. In a complete SEM, both models are estimated simultaneously, allowing for the relationships between latent variables to be studied while explicitly accounting for measurement error in their indicators (or factors). This ability to explicitly model and account for measurement error is a key advantage of SEM over traditional regression methods.

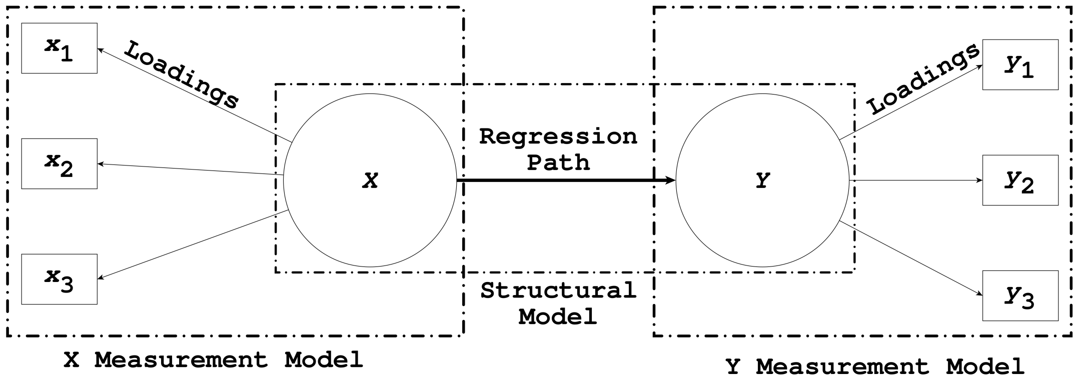


Figure 11: SEM Path Diagram

6.3 SEM Measurement Model

Our initial use of SEM techniques will be to assess the choice of parameters used to develop the model of Equation (4) presented in Section 4.3. The input parameters are M, N_L, and N_B, the matrix dimension, number of Little cores, and number of Big cores, respectively. The output parameter is T_e, execution time. Note that in SEM analysis, we are using the entire raw dataset as is, whereas in the earlier model we would utilize the mean execution time for a set of experiments with the same input parameter values. Also, in this use of SEM, we are not trying to replicate the predictions of the earlier first principles model. Rather, we are trying to assess the relative importance of the input parameters and their impact on the output parameter. Using traditional statistical methods to center and scale the dataset, we therefore can compare variables irrespective of scale, identify relative importance, and interpret the effect size, even when the original units of measurement differ.

The set of variables used in our SEM measurement model is as follows:

- **Main Effects**: Matrix Dimension, Little Island, Cores matrix_dimension_c (Centered Matrix Dimension (md_c))

island_L (The Little Island)

cores_i (Cores ranging from i = 2 to 4)

- **2-Way Pairwise Interactions**: Matrix Dimension x Little Island

  md_c_x_island_L (Matrix Dimension Centered x Little Island)

- **2-Way Pairwise Interactions**: Matrix Dimension x Cores

  md_c_x_cores_2 (Matrix Dimension Centered x Two Cores)

  md_c_x_cores_3 (Matrix Dimension Centered x Three Cores)

  md_c_x_cores_4 (Matrix Dimension Centered x Four Cores)

- **2-Way Pairwise Interactions**: Little Island x Cores

  island_L_x_cores_2 (Little Island x Two Cores)

  island_L_x_cores_3 (Little Island x Three Cores)

  island_L_x_cores_4 (Little Island x Four Cores)

- **3-Way Pairwise Interactions**: Matrix Dimension x Island x Cores

  md_c_x_island_L_x_cores_2 (Matrix Dimension Centered x Little Island x Two Cores)

  md_c_x_island_L_x_cores_3 (Matrix Dimension Centered x Little Island x Three Cores)

  md_c_x_island_L_x_cores_4 (Matrix Dimension Centered x Little Island x Four Cores)

We next examine the following measurement model postulating how all the parameters relate to execution time:

```
Execution Time = f(Main Effects + 2-Way Interactions + 3-Way Interactions)
```

The outcome of this analysis yields what is called the *standard estimate*. Standard estimates are indicative of the change in Execution_time (i.e., in standard deviations) for one-standard-deviation change in the predictor (the variables listed above). The standard error on the standard estimate gives a measure of the precision of that standardized estimate with a smaller standard error meaning a more precise estimate.

Starting with a pair of individual effects, the analysis shows:

- The matrix dimension has a strong impact. A one-standard-deviation increase in M is associated with a 0.705 standard deviation increase in T_e (standard error = 0.023).
- There are diminishing returns to increases in core count. E.g., going from 3 cores to 4 cores (irrespective of whether the cores are Big vs. Little) is associated with the smallest improvement in execution time (only 0.180 decrease in standard deviation of execution time for a one-standard-deviation change in core count, standard error = 0.010).

Neither of the above results are in any way surprising. In fact, they are precisely what one expects via intuition. However, they are now validated by statistical reasoning rather than simply justified by intuition.

We next examine some pairwise effects:

- We first assess the combined effect of matrix dimension and island. This is the strongest effect demonstrated, with a one-standard-deviation change in the interaction term between matrix dimension and island (Big vs. Little cores) associated with a 1.097 standard deviation change in execution time (standard error = 0.023).
- Pairing matrix dimension with core counts, we continue to see diminishing returns for additional cores.

All these effects (both individual and pairwise) are *statistically significant* (`pvalue` $\leq 10^{-3}$), meaning they are not likely due to random chance, given the large sample size.

Again, the results from statistical reasoning match our intuition. The combined effect of matrix dimension and Big vs. Little cores has a substantial impact on overall execution time. The above analysis gives credence to the model of Equation (4), plotted in Figure 4, which does a quite reasonable job of predicting performance over a wide range of configurations.

6.4 SEM Structural Model

We next turn to the use of SEM structural models to investigate the hypothesis that the relationship between working set size and cache is the proximate cause of the refinements to the model that were developed in Section 5.6.

6.4.1 Model 1: Intuitive Latent Variable Construction

Computer performance over the years has relied on the fact that computations are bound primarily in two dimensions *time* and *space*. We now consider the effect of those two dimensions and some model implications:

- **Time (Processor Core Performance)**: The time complexity typically refers to the number of operations that a computational process can complete. This relates directly to processor performance (e.g., speed), as a more performant processor core can execute more instructions per second, thereby reducing time on some real-world task.
- **Space (Memory Cache Efficiency)**: The space complexity typically refers to the amount of storage space (or memory) that a computational process requires to run. This relates to memory capacity, but even with infinite capacity, a limiting factor on memory is data availability or the speed with which the data can be retrieved. Consequently, cache performance has a direct impact on the speed at which the computation can make use of the storage space.

Given these fundamental considerations, we need to derive intuitive structures that reflect these considerations within an appropriate model. We therefore start by hypothesizing a pair of latent variables, `Core_Performance` and `Cache_Efficiency`, in an attempt to separate out the effects of the individual cores and their caches (which we suspect is the issue to be addressed in this stage of the development of the first principles model). We then select a subset of the values measured (some via `perf`) in Section 5.5: `ipc`, `l1_dcache_load_rate`, and `matrix_dimension` associated with `Core_Performance` plus `threads`, `cores`, and `llc_loads` associated with `Cache_Effects`. These relationships are shown graphically in Figure 12. The output adheres to the description given in Section 6.2. We see that there is a negative correlation (represented by the double-headed arrow) between `Core_Performance` and `Cache_Efficiency`, which implies that as the score on one latent variable tends to increase, the score on the other latent variable tends to decrease. In other words, they are inversely related.

Notice the single arrows (representing regression paths or direct effects) with the standardized parameter estimates shown on each line. Reading directly from Figure 12 we can see the impact of each parameter not only on the latent variables but the impact of each latent variable on the (latent) variable of interest, `Performance`. Notice that the R^2 value for `Performance` is only 0.619, which is not a good score.

This model explains a bit over half of the variability in the model but we postulate that the model is underspecified and there are additional constructs and/or factors that must be incorporated into the model to improve its accuracy. Essentially, our latent variable `Performance`, though influenced by `Core_Performance` and `Cache_Efficiency` is only regressed on one factor (`parallel_time`). We have shown above that we can have overlapping performance for one or more configurations (e.g., one Big core is equivalent to three Little cores) which indicates that a single factor is not enough to differentiate the performance variation. In the subsequent model, we will add an island parameter that will enable differentiation of `parallel_time` with respect to island thereby increasing our ability to express the platform behavior for the given factors.

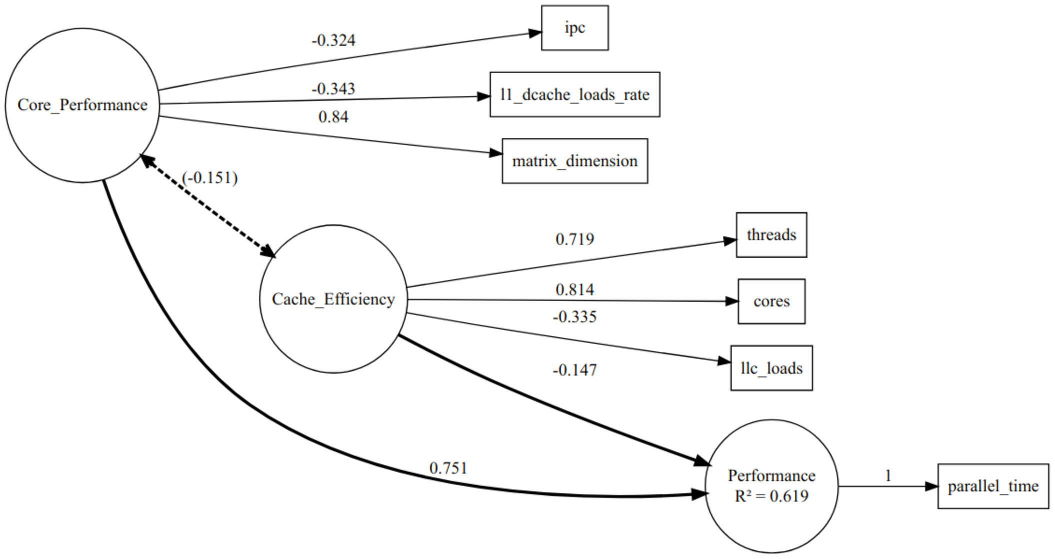


Figure 12: Initial Full SEM Model

6.4.2 Model 2: Evidence Directed Parameter Addition

Intuitively, our principled analysis methods would support the notion that given a `parallel_time` and including an island indicator may assist in better explaining the performance. In Figure 13 we test to see if it is the case, by regressing the latent construct `Performance` on an additional variable `island_numeric`, which is an indicator variable used to identify the island type (Big or Little). This is clearly beneficial to our model as the R^2 jumps up to an excellent 0.98 where we can infer that the parameters shown explain 98% of the performance. These methods are based off of a single analysis run. Next we will bootstrap the model to ensure we have the proper estimates.

6.4.3 Model 3: Bootstrapping for Parameter Significance

Given that our initial model had some challenges we added an additional parameter which adjusted the R^2 value but this is one of many models and we would like to understand the parameters in our model better before we seek to find the best model. To help us understand the parameters in our model, we will bootstrap the SEM model.

Bootstrapping, particularly for SEM models, is a power resampling technique used to estimate the precision of parameter estimates (like factor loadings, regression coefficients, or indirect effects) and to assess their statistical significance without relying on traditional parametric assumptions. Essentially, bootstrapping performs:

- **Resampling with Replacement**: From the original dataset of N observations, a new sample of size N is drawn with replacement. This resampling process is repeated many times, in this instance 1,000 times, creating a large number of *bootstrap samples.*
- **Estimation for Each Sample**: The full SEM model (both measurement and structural components) is estimated for each of these bootstrap samples. This results in a distribution of

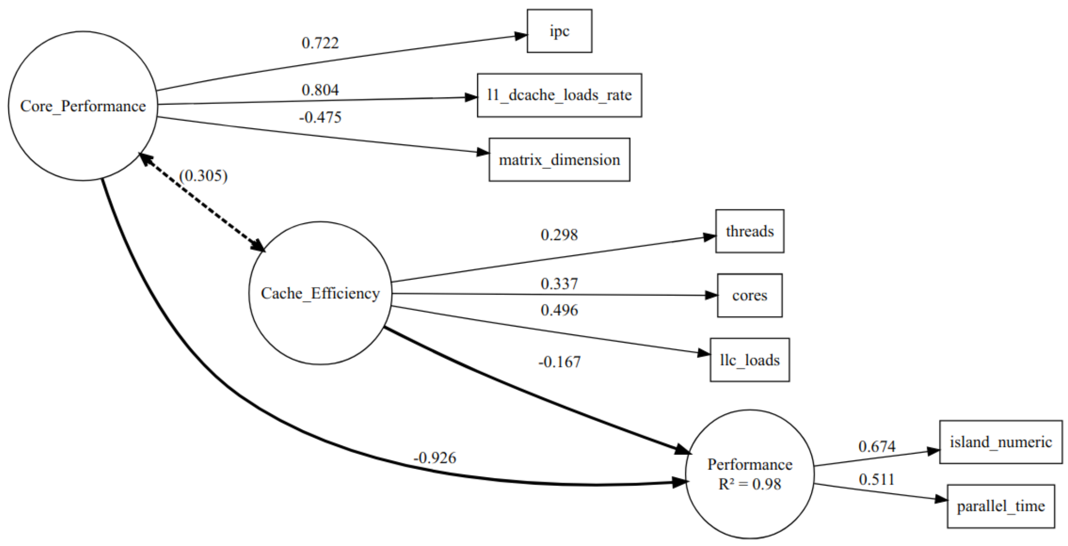


Figure 13: Specified SEM Structural Model

estimated parameters (e.g., a distribution of beta β_i values, a distribution of factor loadings, and a distribution of indirect effects) for each parameter in the model.

- **Inference**: As opposed to relying on theoretical sampling distributions (and assuming normality), bootstrapping uses the empirical distributions to derive standard errors, confidence intervals, and p-values. The standard deviation of a parameter's bootstrap distribution becomes its bootstrap standard error. Bootstrap confidence intervals—e.g., 95% bias-corrected and accelerated (BCa) confidence intervals—are constructed from the percentiles of the bootstrap distribution. If a confidence interval for a parameter does not include zero, that parameter is considered statistically significant.

As indicated in Figure 14, we find all the factors for each latent construct to be statistically significant (as shown by the triple stars $***$ next to each of the parameters' standardized estimates). Of particular importance for our purposes is that the relationship between the latent structures `Core_Performance` and `Cache_Efficiency` are also statistically significant. This provides a level of confirmation that the interaction of working set size and caches was a reasonable motivation for the choices made in the extended version of the first principles model of Section 5.6.

6.5 Performance Simulation for SEM Structural Model Confirmation

The SEM Structural Model is a powerful tool for understanding the relationship between the parameters. We were able to select an intuitive model and confirm the relationships through our empirical measurements. Yet, there is a subtle importance that needs to be promoted as it is often overlooked. The SEM models that we have outlined specify the relationships between the parameters, which may fit not only the current dataset but a large range of datasets for the platform, given that the model is based on the underlying relationships between the hardware-agnostic components. Moreover, though matrix multiplication is typically compute bound, to round out the analysis we would like to

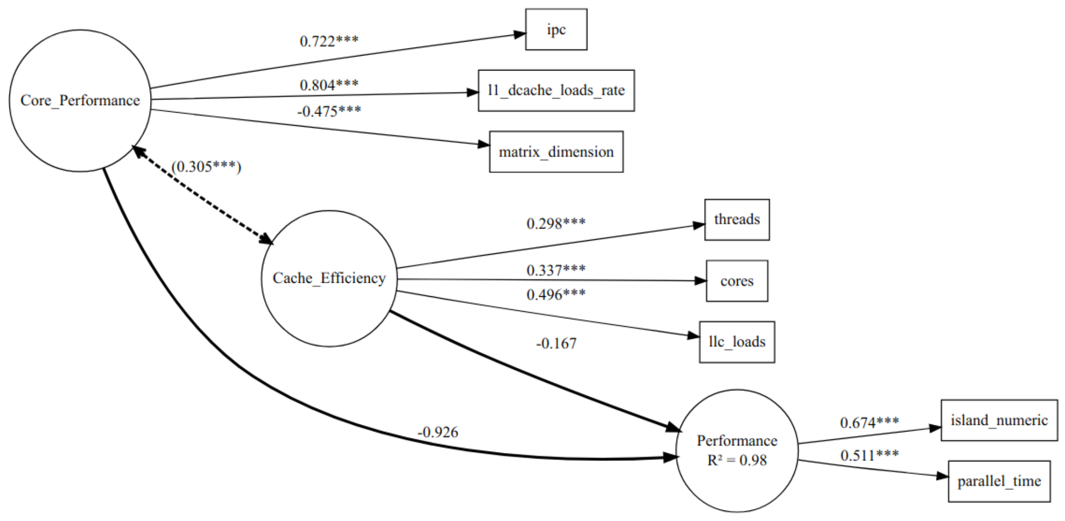


Figure 14: Bootstrapped Specified Model

understand how well this methodology would generalize for memory-bound workloads or the many irregular workloads which are commonly found in edge/embedded computing. Since our analysis is based on empirical measurements made from hardware-agnostic parameters, to understand memory-bound workloads, we only need to consider how the parameters change when the workload shifts from compute bound to memory bound. Suppose that we consider matrix multiplication workloads which are memory bound, what can we say about these parameters?

6.5.1 Relationship and Proportionality Changes

For the sake of discussion, let us consider only the model parameters used with the exception of *threads* and *cores* which will no doubt change the scaling behavior but will lose is ability to translate proportional speedup as in the memory-bound scenario, the memory subsystem lags behind causing these parameters to lose their strong impact on performance. We anticipate that as workloads shift from compute-bound workloads to memory-bound workloads we will experience a shift in metrics as outlined in Table 6.

Table 6: Anticipated Metric Shift

Model Parameters	**Processor to Memory Metric Shift**
Instruction Per Cycle (ipc)	Lower: CPU stalls
L1 Data Cache Load Rate (l1_dcache_loads_rate)	Higher: Working set size exceeding L1/L2 capacity
Last Level Cache Loads (llc_loads)	Higher: Data retrieval from L3/LLC
Matrix Dimension (matrix_dimension)	Higher: Matrices exhausting L1/L2/L3/LLC
Parallel Execution Time (parallel_time)	Higher: Memory latency and limiting bandwidth

6.5.2 CPU-bound to Memory-bound Metric Distributions

To support this notion, we have synthetically perturbed the existing dataset to shift parameters into memory-bound constraints, with the original dataset being shifted according to the memory-bound values shown in Table 7. Given the heterogeneity of the platform, we cannot assume the distribution of the data and therefore cannot simulate new data as the distributions may differ dramatically. Instead, we choose to permute the existing dataset to maintain the original distributions but changed the values so that they reflect the memory-bound constraints that we wish to investigate. To support our decision to rescale the data, for each parameter shift consideration, we will illustrate the simulated distribution overlaid with the original distribution, along with the mean values used for modeling a memory-bound workload against the original compute-bound dataset for comparison.

It is highly unlikely that a single constant of proportionality exists between compute-bound and memory-bound workloads. Yet, a simple model that either simulates a metric shift or perturbs the data in this way will prove both effective and useful in determining the efficacy of our SEM models in light of such considerations. For a naive 3-loop matrix multiplication workload shifted from compute-bound to memory-bound characteristics, we anticipate that the distributions for sampled metrics on the same platform would change according to prior performance analysis and intuitive mathematical reasoning as outlined in the compute-bound to memory-bound prediction given in Section 6.5.3 below.

Table 7: Model Parameters and Investigated Metric Shift

Shifted Model Parameters	Memory-Bound Values
Instruction Per Cycle (ipc)	Decrease by 75%
L1 Data Cache Load Rate (l1_dcache_loads_rate)	Increase by 800%
Last Level Cache Loads (llc_loads)	Increase by 250%
Matrix Dimension (matrix_dimension)	Increase by $10,000\%$
Parallel Execution Time (parallel_time)	Increase by $2,000\%$

6.5.3 Metric Predictions

Here, we make several predictions of impacts that we anticipate will result from a shift to a memory-bound workload. These predictions will then be validated in Section 6.5.4.

- **Prediction: Instruction Per Cycle (ipc)**

  Compute-Bound: The IPC would likely be high and tighly clustered as this reflects the processor execution many instructions per clock cycle because of the general availability of fast caches, registers, and memory subsystem enabling high utiliation of execution units.

  Memory-Bound: The IPC distribution may be significantly lower and broader. Particularly, the peak of the distribution would shift to the left as the processor stalls frequently waiting for data to be fetched from the slower main memory (DRAM) into cache. The broadening may also reflect variability in the memory access latency.

- **Prediction: L1 Data Cache Load Rate (l1_dcache_loads_rate)**

  Compute-Bound: This rate would most likely be high and tightly clustered. The CPU which running at high speed would rapidly request data from the L1 cache to furnish the execution units.

  Memory-Bound: This rate distribution may shift lower. Though the CPU continues to request data, the effective rate would decrease due to the frequent stalling of the processor. The distribution shape may become wider due to the variable stalling behavior caused by memory access times.

- **Prediction: Last Level Cache Loads (llc_loads)**

  Compute-Bound: The LLC loads distribution would most likely be lower or moderate. With data moving through the caches, a CPU-bound workload will make use of data more effectively within the L1/L2 caches resulting in fewer compulsory and/or cacpcity misses reaching the slower LLC.

  Memory-Bound: The LLC loads distribution would most likely be significantly higher. The shape may be similar but translated to a higher mean. The memory-bound workload would be characterized by poor data locality and frequent lower level (L1/L2) cache misses ultimately resulting in a high request volume that must be addressed by the LLC.

- **Prediction: Parallel Execution Time (parallel_time)**

  Compute-Bound: The distribution would be expected to be lower (given faster execution times) and potentially more skewed to the right. This would be due in part to potential uniformity of matricies and a lack of any significant memory contention with the time being dominated by computations (FLOPS).

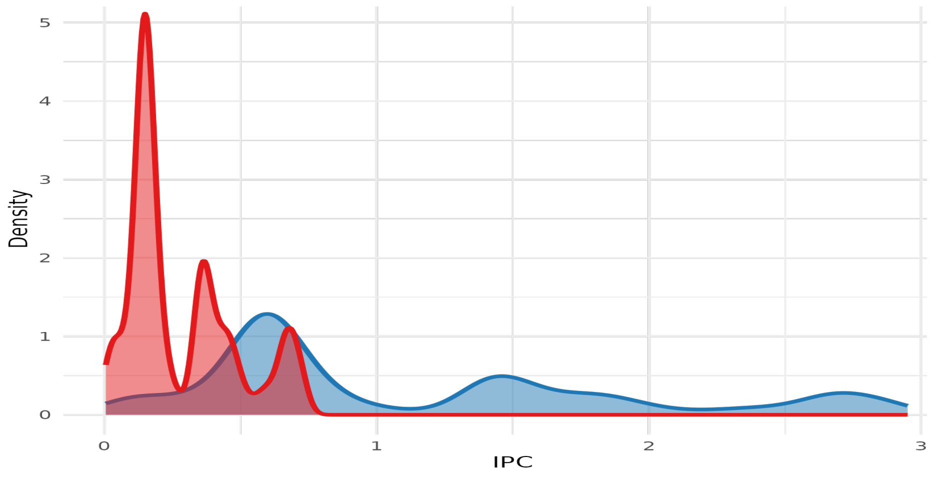
  Memory-Bound: The distribution would be higher (given longer execution times). In this instance, the total execution time would be dominated by memory access latency due to time spent waiting for data resulting in a broader and potentially heavier right tail or multi-modal appearance if there is significant non-uniform memory access (NUMA) effects or variability in memory contention and/or bandwidth saturation across runs with the time being dominated by memory stalls.

6.5.4 Metric Shift Results

As shown in Table 8, using the dataset with the parameters that simulate a memory-bound workload is identical to the previous bootstrapped model shown in Figure 14. This is because the SEM model expresses the underlying relationship between the parameters which intuitively should not change because of the workload as the data will flow through the same components and the relationships between those components should not change. This analysis further confirms the relationships between components and the structure of the model. Plots that confirm the predictions above are shown in Figures 15 to 18.

Table 8: Bootstrapped Memory-Bound Model Regressions and Variances

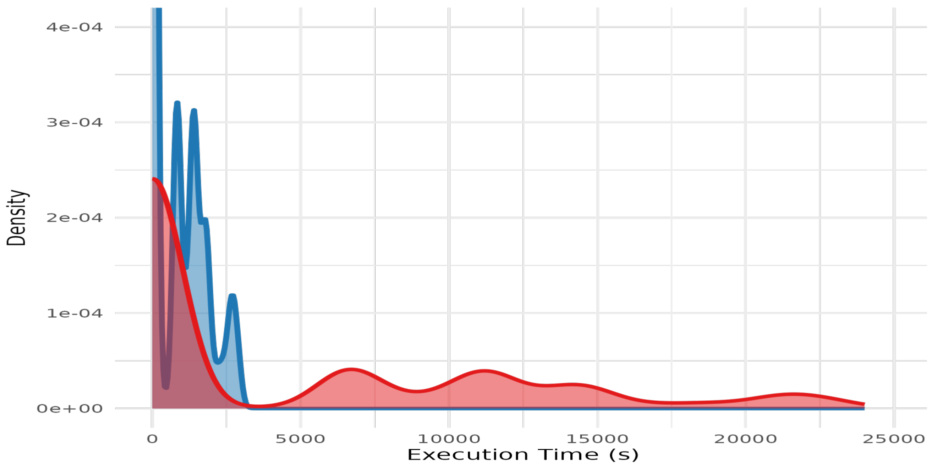
	Estimate	Std.Err	P(>z)
Regressions:			
Performance ∼			
Cache_Efficiency	-0.1667	0.0219	0.000
Cache_Efficiency ∼ cores	0.3368	0.0385	0.000
Cache_Efficiency ∼ llc_loads	0.4960	0.0167	0.000
Cache_Efficiency ∼ threads	0.2975	0.0377	0.000
Core_Performance	-0.9262	0.0137	0.000
Core_Performance ∼ ipc	0.7222	0.0110	0.000
Core_Performance ∼ l1_dcache_loads_rate	0.8045	0.0112	0.000
Core_Performance ∼ matrix_dimension	-0.4751	0.0150	0.000
island_numeric	0.6740	0.0087	0.000
parallel_time	0.5109	0.0119	0.000
Variances:			
Performance	0.9796	0.0204	0.000



(a) IPC Distribtution
Mean: Sampled (1.13) vs. Simulated (0.28)

🟦 Sampled_CPU_Bound
🟥 Simulated_Memory_Bound

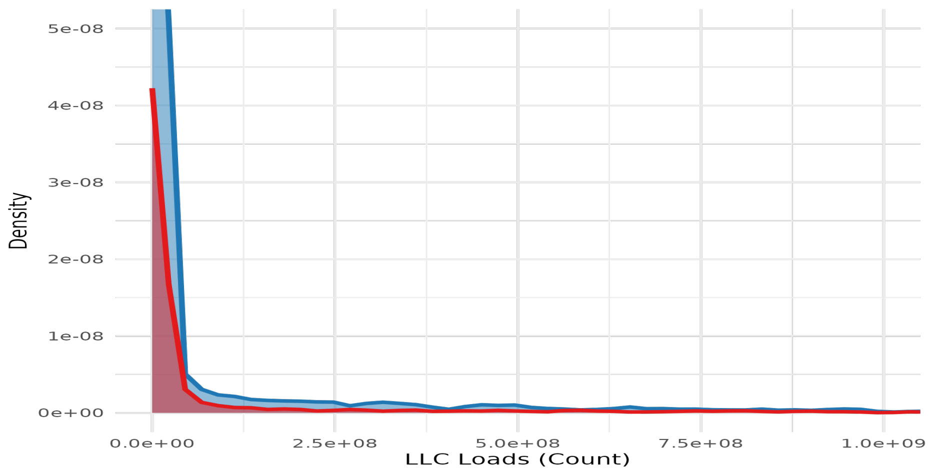
Figure 15: Sampled vs. Simulated - IPC



(a) L1 DCache Loads Rate Distribution
Mean (Seconds): Sampled (592.97) vs. Simulated (4,743.72)

🟦 Sampled_CPU_Bound
🟥 Simulated_Memory_Bound

Figure 16: Sampled vs. Simulated - L1 DCache Load Rates



(a) LLC Distribution
Mean (Count): Sampled (114,658,410.36) vs. Simulated (286,646,025.90)

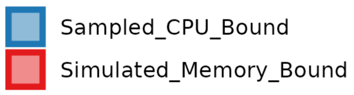
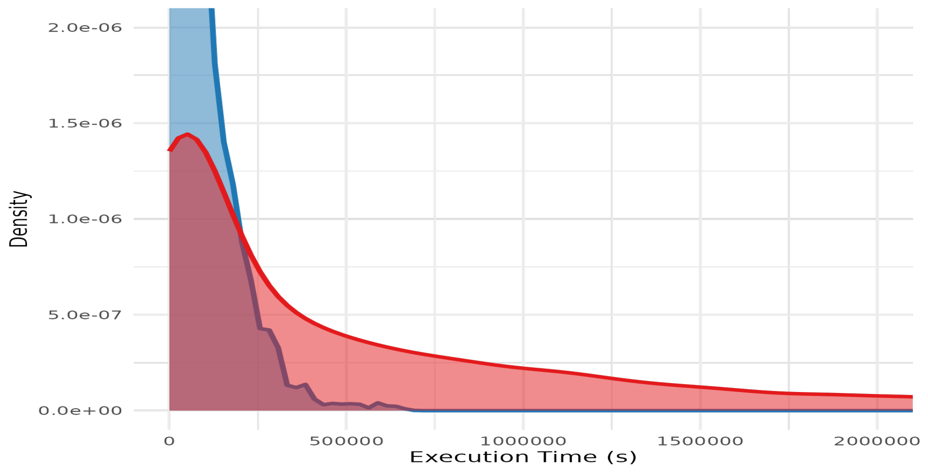


Figure 17: Sampled vs. Simulated - LLC



(a) Parallel Time Distribution
Mean (Seconds): Sampled (14,287.00) vs. Simulated (285,740.00)

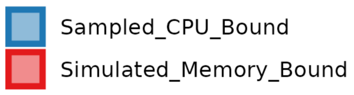


Figure 18: Sampled vs. Simulated - Parallel Time

7 Discussion

We organize the discussion of the models into two parts, first discussing the first principles model and following that with a discussion of the structural equation modeling.

7.1 First Principles Model

The coefficients on the workload expression, a_3, a_2, a_1, are likely proportional to factors that determine how efficiently the platform can perform arithmetic operations, handle memory access, and manage initialization and completion requirements. As the number of cores increases, these coefficients decrease as additional processing units allow for faster computation and improved data handling, reducing memory and lower-level overheads through additional parallelism.

Each coefficient corresponds to different stages of the computation, specifically:

1. The cubic term (a_3M^3) represents arithmetic operations.
2. The quadratic term (a_2M^2) corresponds to memory bandwidth required for storing and retrieving matrix elements.
3. The linear term (a_1M) accounts for both pre- and post-processing steps.

As the number of cores increases, these coefficients are influenced by the parallelism in computation, memory access, and initialization:

- a_3 decreases as more cores allow faster arithmetic operations.
- a_2 decreases as memory bandwidth improves with more cores.
- a_1 decreases due to improved task granularity and scheduling.

Therefore, increasing cores reduces each coefficient proportionally.

Overall, the model presented in Sections 4.3 and 5.6 does a good job both in matching empirical measurements (indicating correctness) and also providing insight into how and why certain behaviors exhibit themselves. Both of the above points are important in judging the quality and usefulness of the model.

The above supports the latter two hypotheses presented in Section 3. The forward model (actually, several forward models) has been shown and empirically validated, and all of the expressions are invertible.

7.2 Structural Equation Modeling

While SEM techniques can be used to develop models that directly predict outcomes given particular inputs, that is not how we are using them. Instead, we are using SEM techniques (both SEM measurement models and SEM structural models) to help us assess whether or not the assumptions made in the first principles model were appropriate.

As such, the initial use (measurement models) is quite conclusive, demonstrating the importance of the input parameters chosen in the first principles model, in addition to confirming several performance effects that we already strongly suspected.

The second use (structural models) is supportive of our assumptions. The bootstrapped model of Figure 14 shows a statistically significant relationship between the latent variables representing both core performance and cache efficiency, with the path ways connecting them to the performance latent variable being statistically significant as well.

The above implies that there is likely additional hidden structure in the data (i.e., additional latent variables) that we have not yet included in the SEM structural model. For example, the performance latent structure is associated with the vector $\overrightarrow{P}$ and the core and cache latent structures are associated with the vector $\overrightarrow{H}$. Potentially we could include in the model a latent structure associated with $\overrightarrow{A}$, and maybe even $\overrightarrow{S}$. Investigating alternative models that could expose this structure is left for future work.

8 Conclusions and Future Work

Hardware, software, and associated constraints augment the vast multi-objective performance space that must be navigated to reach performance goals. Computer performance modeling makes the performance space navigable. Using analytical models, equations, and formalisms, researchers are able to predict performance outcomes for many homogeneous platforms. The homogeneous region has been well-defined, annotated, and studied. This comprehensive performance knowledge is regularly leveraged for entire classes of homogeneous platforms.

However, heterogeneous platforms present a relatively unexplored region of the same space. We argue that the heterogeneous performance space, sharing commonalities with homogeneous regions, is knowable, structured, and exploitable. Furthermore, within that region, there are equivalent performance relationships and models, both generalizable and invertible, that characterize the platforms within that space as our experiments thus far demonstrate. We expect that future experiments will reveal further classes of platforms with similar characteristics.

The experiments in Section 5 reveal an underlying performance structure that we hypothesized should exist. In Section 5.1, by varying the island parameter we showed that the platform performance quite naturally clustered into similar performance groupings based on island selection. As we introduced additional parameters in Section 5.2, we found performance overlap between core configurations. Particularly, the performance of one single-threaded Big core overlapped execution time with three Little cores. The Welch two-sample t-test showed with statistical significance that the mean execution time for the configurations (Big[1,1] vs Little[3,3]) was equal, supported our hypothesis of equivalent relationships.

The stability of performance relationships allowed us to identify a smallest granularity compute unit (i.e., Little[1,1]). We then were able to normalize the platform performance by scaling performance to the smallest compute unit granularity. We then utilized a Big-to-Little ratio that enables performance prediction of the various platform configurations. Section 5.3 showed the predictive power of the Big-to-Little ratio.

We were able to predict any Little core configuration accurately, but the predictions diverged for some Big core configurations. We investigated the normalized execution ratios graphically and found that the ratio appeared parameter dependent. This was supported in Section 5.5 where we found that workload size was an important factor in parameter correlation. Armed with these observations, we re-parameterized the Big-to-Little ratio in Section 5.6 showing the prediction precision using the re-parameterized model which thus supported our hypotheses for the existence of a generalizable model and hinted at potential model invertability.

To assess the validity of the underlying assumptions made in developing the model, we turned to structural equation modeling, a set of data analysis techniques that have seen extensive use in many disciplines (e.g., social sciences, psychology, etc.). Using SEM measurement models, we confirmed the parameters selections made in our first principles model, and using SEM structural models, we showed evidence that our refinements to the first principles model were appropriate.

To summarize,

1. The ability to model Big performance as a scaled version of Little performance validates the first two hypotheses.

2. The close match between modeled performance and empirical measurements validates the third hypothesis.

3. The form of the model expressions being straightforward to invert validates the fourth hypothesis.

4. There is a trade-off available between model accuracy and model complexity.

Having shown the validity of our hypotheses, these experiments point to future work wherein we intend to discover other platforms and configurations occupying the performance space, in order to apply the characterization methods to new platforms. Both the application and the execution platform are individual examples within a much larger space. We intend to explore AMP usage in domains including:

- Autonomous Vehicles [24]: Real-time object detection and localization using matrix operations on local devices.
- Smart Factories [30]: IoT-enabled monitoring requiring matrix computations for predictive maintenance or quality control.
- Health Monitoring [17, 18]: Continuous health monitoring via wearable or implantable devices, involving matrix operations for signal processing.
- Agriculture IoT [26]: Edge-based crop monitoring using data from farm sensors.

The models developed in this paper relied on an extensive empirical data set. We would like to explore methods for developing and validating models of this general type using fewer execution runs. This might be facilitated using stochastic optimization techniques including hybrid sampling and one-armed bandit techniques [34]. The models presented here also can be expanded in a number of ways. Each of the vectors, $\overrightarrow{P}$, $\overrightarrow{A}$, and $\overrightarrow{H}$, can be extended to include more elements. Additional variations in the system software vector, $\overrightarrow{S}$, also may be incorporated into the model. For example, controlling how OpenMP maps matrix elements to cores is a scheduling option in $\overrightarrow{S}$ that would enable us to incorporate Dual island results into the current model. Of particular interest to us is the hardware vector $\overrightarrow{H}$. Given the proliferation of ARM devices on the market, we would like to compare performance, orchestration, and modeling techniques used on a wider range of platforms including e.g., the NVIDIA Jetson. Our preliminary experiments indicate the possibility of extension to many edge-based platforms. Finally, we would like to expand the use of SEM techniques to help us quantify the structural relationships that exist in data that relates applications ($\overrightarrow{A}$), hardware ($\overrightarrow{H}$), system software ($\overrightarrow{S}$), and performance ($\overrightarrow{P}$).

Acknowledgments

Support provided by NSF grant CNS-2229290 (CPS) and a gift from BECS Technology, Inc.

References

[1] Yoav Almog, Roni Rosner, Naftali Schwartz, and Ari Schmorak. Specialized dynamic optimizations for high-performance energy-efficient microarchitecture. In *Proc. of Int'l Symposium on Code Generation and Optimization*, pages 137–148. IEEE, 2004.

[2] Deepthi Amuru, Andleeb Zahra, Harsha V Vudumula, Pavan K Cherupally, Sushanth R Gurram, Amir Ahmad, and Zia Abbas. AI/ML algorithms and applications in VLSI design and technology. *Integration*, 93, 2023.

[3] Jonathan C. Beard and Roger D. Chamberlain. Analysis of a simple approach to modeling performance for streaming data applications. In *Proc. of Int'l Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 345–349. IEEE, August 2013.

[4] Anastasiia Butko, Florent Bruguier, Abdoulaye Gamatié, Gilles Sassatelli, David Novo, Lionel Torres, and Michel Robert. Full-system simulation of big.LITTLE multicore architecture for performance and energy exploration. In *Proc. of 10th Int'l Symposium on Embedded Multicore/Many-core Systems-on-Chip*, pages 201–208. IEEE, 2016.

[5] Yuan Chou, Brian Fahs, and Santosh Abraham. Microarchitecture optimizations for exploiting memory-level parallelism. *ACM SIGARCH Computer Architecture News*, 32(2):76, 2004.

[6] Shaohang Cui, Douglas Buchanan, and Ken Ferens. A knapsack scheduling algorithm for soft real-time multiprocessor system. In *Proc. of Int'l Conference on Embedded Systems, Cyber-physical Systems, and Applications*, 2013.

[7] Yi Ding, Ahsan Pervaiz, Michael Carbin, and Henry Hoffmann. Generalizable and interpretable learning for configuration extrapolation. In *Proc. of 29th Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering*, pages 728–740. ACM, 2021.

[8] Christophe Dubach, Timothy M Jones, and Michael FP O'Boyle. Exploring and predicting the effects of microarchitectural parameters and compiler optimizations on performance and energy. *ACM Transactions on Embedded Computing Systems*, 11(1):1–24, 2012.

[9] Clayton J. Faber and Roger D. Chamberlain. Application of network calculus models to heterogeneous streaming applications. *International Journal of Networking and Computing*, 15(1):51–63, January 2025.

[10] Clayton J. Faber, Tom Plano, Samatha Kodali, Zhili Xiao, Abhishek Dwaraki, Jeremy D. Buhler, Roger D. Chamberlain, and Anthony M. Cabrera. Platform agnostic streaming data application performance models. In *Proc. of IEEE/ACM Workshop on Redefining Scalability for Diversely Heterogeneous Architectures*, November 2021.

[11] J. B. Grace. *Structural Equation Modeling and Natural Systems.* Cambridge University Press, New York, 2006.

[12] Steven Harris, Roger D Chamberlain, and Christopher Gill. OpenCL Performance on the Intel Heterogeneous Architecture Research Platform. In *Proc. of High Performance Extreme Computing Conference.* IEEE, 2020.

[13] R. H. Hoyle. *Structural Equation Modeling: Concepts, Issues, and Applications.* Sage, Thousand Oaks, CA, 1995.

[14] R. H. Hoyle. *Structural Equation Modeling for Social and Personality Psychology.* Sage, London, 2011.

[15] Olumuyiwa Ibidunmoye, Francisco Hernández-Rodriguez, and Erik Elmroth. Performance anomaly detection and bottleneck identification. *ACM Computing Surveys*, 48(1):1–35, 2015.

[16] Engin Ïpek, Sally A McKee, Rich Caruana, Bronis R de Supinski, and Martin Schulz. Efficiently exploring architectural design spaces via predictive modeling. *ACM SIGOPS Operating Systems Review*, 40(5):195–206, 2006.

[17] SM Riazul Islam, Daehan Kwak, MD Humaun Kabir, Mahmud Hossain, and Kyung-Sup Kwak. The internet of things for health care: a comprehensive survey. *IEEE Access*, 3:678–708, 2015.

[18] Hazilah Mad Kaidi, Mohd Azri Mohd Izhar, Rudzidatul Akmam Dziyauddin, Nur Ezzati Shaiful, and Robiah Ahmad. A comprehensive review on wireless healthcare monitoring: System components. *IEEE Access*, 12:35008–35032, 2024.

[19] R. B. Kline. *Principles and Practice of Structural Equation Modeling.* Guilford Press, New York, 2010.

[20] Hugh Leather and Chris Cummins. Machine learning in compilers: Past, present and future. In *Proc. of Forum for Specification and Design Languages.* IEEE, 2020.

[21] Benjamin C. Lee and David M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proc. of 12th Int'l Conference on Architectural Support for Programming Languages and Operating Systems*, page 185–194. ACM, 2006.

[22] Xiaoyao Liang and David Brooks. Microarchitecture parameter selection to optimize system performance under process variation. In *Proc. of IEEE/ACM Int'l Conf. on Computer-Aided Design*, pages 429–436, 2006.

[23] Sebastian Litzinger, Jörg Keller, and Christoph Kessler. Scheduling moldable parallel streaming tasks on heterogeneous platforms with frequency scaling. In *Proc. of 27th European Signal Processing Conference*, pages 1–5, 2019.

[24] Liangkai Liu, Sidi Lu, Ren Zhong, Baofu Wu, Yongtao Yao, Qingyang Zhang, and Weisong Shi. Computing systems for autonomous driving: State of the art and challenges. *IEEE Internet of Things Journal*, 8(8):6469–6486, 2020.

[25] Yuchun Ma, Zhuoyuan Li, Jason Cong, Xianlong Hong, Glenn Reinman, Sheqin Dong, and Qiang Zhou. Micro-architecture pipelining optimization with throughput-aware floorplanning. In *Proc. of Asia and South Pacific Design Automation Conference*, pages 920–925. IEEE, 2007.

[26] Md Najmul Mowla, Neazmul Mowla, AFM Shahen Shah, Khaled M Rabie, and Thokozani Shongwe. Internet of Things and wireless sensor networks for smart agriculture applications: A survey. *IEEE Access*, 11:145813–145852, 2023.

[27] Marco Pagani, Alessio Balsini, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. A Linux-based support for developing real-time applications on heterogeneous platforms with dynamic FPGA reconfiguration. In *Proc. of 30th Int'l System-on-Chip Conference*, pages 96–101. IEEE, 2017.

[28] David V Schuehler, Benjamin C Brodie, Roger D Chamberlain, Ron K Cytron, Scott J Friedman, Jason Fritts, Phillip Jones, Praveen Krishnamurthy, John W Lockwood, Shobana Padmanabhan, and Huakai Zhang. Microarchitecture optimization for embedded systems. In *Proc. of 8th High Performance Embedded Computing Workshop*, 2004.

[29] N. Shirazi, A. Walters, and P. Athanas. Quantitative analysis of floating point arithmetic on FPGA based custom computing machines. In *Proc. of Symposium on FPGAs for Custom Computing Machines*, pages 155–162. IEEE, 1995.

[30] Mohsen Soori, Behrooz Arezoo, and Roza Dastres. Internet of things for smart factories in Industry 4.0, a review. *Internet of Things and Cyber-Physical Systems*, 3:192–204, 2023.

[31] Stephen M. Trimberger and Jason J. Moore. FPGA security: Motivations, features, and applications. *Proceedings of the IEEE*, 102(8):1248–1265, 2014.

[32] Siqi Wang, Gayathri Ananthanarayanan, Yifan Zeng, Neeraj Goel, Anuj Pathania, and Tulika Mitra. High-throughput CNN inference on embedded ARM big.LITTLE multicore processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2254–2267, 2019.

[33] Piotr Wojciechowski, K Subramani, Alvaro Velasquez, and Bugra Caskurlu. Priority-based bin packing with subset constraints. *Discrete Applied Mathematics*, 342:64–75, 2024.

[34] Bochun Wu, Tianyi Chen, Wei Ni, and Xin Wang. Multi-agent multi-armed bandit learning for online management of edge-assisted computing. *IEEE Transactions on Communications*, 69(12):8188–8199, 2021.

[35] Anna Yue and Sanyam Mehta. An application-oriented approach to designing hybrid CPU architectures. In *Proc. of Int'l Symposium on Performance Analysis of Systems and Software*, pages 92–102. IEEE, 2023.

[36] Yuhao Zhu and Vijay Janapa Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. In *Proc. of 19th Int'l Symp. on High Performance Computer Architecture*, pages 13–24. IEEE, 2013.