Machine Learning for Design Space Exploration and Optimization of Manycore Systems

Ryan Gary Kim  
Department of Electrical and Computer Engineering  
Colorado State University  
Fort Collins, CO, USA  
ryan.g.kim@colostate.edu

Janardhan Rao Doppa  
School of Electrical Engineering and Computer Science  
Washington State University  
Pullman, WA, USA  
jana.doppa@wsu.edu

Partha Pratim Pande  
School of Electrical Engineering and Computer Science  
Washington State University  
Pullman, WA, USA  
pande@wsu.edu

ABSTRACT
In the emerging data-driven science paradigm, computing systems ranging from IoT and mobile to manycores and datacenters play distinct roles. These systems need to be optimized for the objectives and constraints dictated by the needs of the application. In this paper, we describe how machine learning techniques can be leveraged to improve the computational-efficiency of hardware design optimization. This includes generic methodologies that are applicable for any hardware design space. As an example, we discuss a guided design space exploration framework to accelerate application-specific manycore systems design and advanced imutation learning techniques to improve on-chip resource management. We present some experimental results for application-specific manycore system design optimization and dynamic power management to demonstrate the efficacy of these methods over traditional EDA approaches.

CCS CONCEPTS
• Hardware → Electronic design automation, On-chip resource management, Application-specific VLSI designs;  
• Computing methodologies → Machine learning;  
• Computer systems organization

KEYWORDS

1 Introduction
In recent years, three fundamental developments have changed the landscape of computer systems design: technological changes, such as the slowing of Moore’s Law and the ending of Dennard Scaling; shifting computing paradigms, going from centralized PC/server to distributed or on-device mobile and IoT; and the rise of several application domains, such as deep learning and graph analytics. Together, these developments pose serious challenges in hardware design that cannot be solved through simple transistor scaling. These challenges include higher computing demands, constrained form factors, and renewed concerns in performance, energy, reliability, and system cooling.

As a result, application-specific manycore processors present a promising alternative over general-purpose systems to improve performance while meeting energy, temperature, and area constraints [1][2][3]. Unfortunately, there are several obstacles in the design of these application-specific systems. First, they need to fine-tune many design parameters and implement sophisticated run-time resource managers. Second, the intended application use-case greatly changes the deployment platform and design objectives. Lastly, the evaluation of these designs, like hardware simulations, are slow and expensive to execute. These factors slow down design space exploration and complicate run-time management. Hence, the design of these systems needs efficient modeling and optimization techniques to quickly uncover near-optimal solutions and satisfy multiple system objectives.

Fortunately, recent developments in machine learning present candidate solutions to design highly optimized hardware. In this paper, we will describe how machine learning techniques can be used to improve the computational-efficiency of manycore system design optimization. We examine high-level problems within the manycore domain such as large design spaces, multiple design objectives, and distributed run-time policies. We then present state-of-the-art machine learning solutions that can be applied to similar problems across the entire hardware design optimization landscape.

2 Manycore System Design and Optimization
Due to transistor scaling and various emerging integration and interconnection technologies, it is now possible to integrate a large number of cores onto a single chip. By doing so, we can
vastly improve the performance and energy-efficiency of highly parallel systems by avoiding expensive off-chip data transfers.

For these complicated manycore systems, there are two major design stages: design-time optimization and run-time optimization. During design-time optimization, the designer needs to choose hardware parameters that maximize and meet the objectives and constraints arising from the intended applications. For example, the combination and placement of CPUs, GPUs, memory, accelerators, and network connections need to be optimized for performance, energy, and thermal objectives. However, these optimization decisions have important implications on the complex interactions between memory and network subsystems. When considering design-time decisions across the entire computing platform, non-obvious implications of these interactions make design-time optimization very difficult.

During run-time, heterogeneous time-varying workloads result in non-uniform utilization of the system resources. It is the designer’s goal to optimize run-time policies that take advantage of this fact to dynamically allocate resources to improve system objectives. The main idea is to allocate precisely the right amount of resources to each task to achieve system objectives. Typically, over- or under-allocation lead to sub-optimal performance. For example, the system can reduce the operating frequency during low-intensity workloads to reduce power and maintain performance. Unfortunately, these run-time policies are difficult to design when we consider multiple subsystems and optimization objectives. In addition, the importance of each optimization objective depends on the application domain and use-case.

In the following sections, we present a couple of key issues in the design of emerging massive manycore systems. Specifically, we will look at the problems of design space exploration, a crucial technique for design-time optimization; and dynamic resource management, a method of run-time optimization. We discuss the limitations of traditional solutions for these problems in application-specific manycore systems and how machine learning has been successfully applied to these problems. We will also briefly consider the role of machine learning in the future of hardware design and its potential impact on computing systems.

3 Design Space Exploration of Manycore Systems

With the recent rise of diverse application domains (e.g., deep learning, graph analytics, autonomous systems) and computing paradigms (e.g., IoT and distributed computing), it is more important than ever to create systems that are optimized for these specific use-cases at design-time. This compels us to examine how we search through large design spaces and how to do this exploration more efficiently to find near-optimal designs. Here, we look at the goals and challenges that affect the design of manycore chips.

3.1 Main Challenges

As we improve our ability to create larger manycore systems with breakthrough technology, we must advance the associated design methods. These systems enable an immense number of design decisions, from selecting the combination and placement of processing elements (PEs), e.g., CPUs, GPUs, and accelerators, the network topology for the PEs and memory, to the integration of emerging technologies such as photonics and wireless. Since many of these design decisions have complex interactions with one another, e.g., the network topology and PE placement both factor into the nature of the traffic patterns, all of these decisions must be considered in a joint manner.

Furthermore, design objectives can have different priorities based on the application domain and the application use-case. For example, IoT edge processors running deep learning inference may prioritize energy and reliability over performance to achieve longer up-time and greater lifetime of the device. Since the application domain and use-case governs our design-space exploration, it is important that we can quickly search through the design-space and easily optimize the appropriate objectives.

3.2 Drawbacks of Traditional Methods

There are two prevalent approaches that are used to search the design space for suitable solutions: local search algorithms such as simulated annealing (SA) [4] and genetic algorithms (GA) such as NSGA-II [5]. For local search based methods, one of the key limitations is that the quality of the local search critically depends on the starting point of the search process. Although algorithms like SA try to mitigate this effect by incorporating some random exploration, they are still limited by the local nature of the search. If the search repeatedly begins near poor local minima, it is possible that the search will never find a high-quality solution. Hence, these types of algorithms can take considerable amounts of time to find high-quality solutions, especially as the design space and the number of objectives scales up. Due to this scalability limitation, although local search-based methods and genetic algorithms were sufficient in the past, large and complex design spaces will require sophisticated algorithms for design space exploration to optimize multiple correlated objectives.

3.3 Machine-Learning Guided Exploration

Machine-learning techniques can enable the problem-solver (a computational search procedure) to make intelligent search decisions to improve the computational efficiency of finding (near-) optimal solutions [23]. As discussed in Section 3.2, one of the major limitations of current local search based methods is that

![Figure 1: Overview of the STAGE algorithm.](image-url)
the quality of the solution critically depends on the starting location of the local search. Intuitively, if we can select starting locations that will lead the local search to high-quality local optima, it will enable the local search algorithms to find near-optimal solutions with significantly less restarts. Machine learning techniques can be utilized to learn useful search control knowledge based on past local search runs. The learned knowledge can identify parts of the design space where the local search should focus on. If this is successful, a lot of needless exploration can be eliminated.

Taking this general idea, STAGE [6] uses machine learning techniques to improve the accuracy and computational-efficiency of local search methods (Fig. 1). The first stage (Local search) performs a normal local search, guided by a cost function based on the designer’s goals. The second stage (Meta search) tries to use the search trajectories from previous local search runs to learn an evaluation function that can predict the outcome of a local search procedure from a given starting point.

Using this evaluation function, STAGE intelligently selects starting states with a high potential to lead to better quality solutions. This allows the algorithm to prune away bad starting states and reduce the number of local search calls needed to find (near-) optimal designs in the given design space. Unlike STAGE, other algorithms that use random restarts do not leverage any such knowledge and spend significant time searching from states that would otherwise be rejected by STAGE. Therefore, STAGE explicitly guides the search towards promising areas of the search space much faster than conventional search algorithms.

We can examine how STAGE performs in the context of designing 3D network-on-chip (NoC) architectures for manycore systems [7][8]. In Fig. 2(a) we demonstrate the quality of the best-found solution over time for SA, NSGA-II (GA), and STAGE for optimizing the communication cost of NoC design. We observe that STAGE finds a high-quality solution much faster than both SA and GA algorithms. In addition, the benefits of STAGE over SA improves as the system size, i.e., number of cores, increases (Fig. 2(b)). These results demonstrate the ability of STAGE to scale well to more challenging design space exploration problems.

Moving forward, the Pareto hypervolume (PHV) [9] metric can be used to extend STAGE to accommodate multiple objectives. The PHV metric corresponds to the hypervolume of the dominated portion of the objective space and is a measure for the quality of candidate Pareto sets [9]. PHV allows us to handle any number of objectives since PHV maps to a single output (cost). This is particularly useful for learning the evaluation function via a regression learning algorithm.

3.4 Open Problems in Design-Time Optimization

We list three open problems for design-time optimization: a) How can we optimize multiple objectives in the form of analytical functions with improved computational-efficiency and theoretical guarantees on the accuracy of solutions; b) Analytical models are not as accurate as simulations in evaluating designs. How can we perform efficient hybrid optimization guided by both analytical models and simulations; and c) How can we perform design optimization efficiently guided by simulations when an analytical model for an objective cannot be defined (e.g., system lifetime).

4 Dynamic Resource Management in Manycore Systems

During the execution of an application and the lifetime of the chip, certain scenarios influence different design variables that can cause design-time decisions to become sub-optimal. For instance, memory-bound applications may not need higher core frequencies or transistor aging may alter optimal voltage and frequency pairings. These scenarios emphasize the need for run-time policies that can control the available resource knobs to optimize for the current state of the manycore system.

There are currently a wide range of knobs available in a typical manycore system, which can be tuned as the workload of the application changes. These knobs include the voltage and frequency of each element (core, memory, etc.), task scheduler parameters, and adaptive data routing. In more advanced systems, additional knobs include the configuration of reconfigurable cores and the level of precision for workloads that allow approximate outputs. In the end, these run-time policies should tune these knobs to achieve the designer’s objectives and constraints. Some examples include adjusting the voltage and frequency to achieve power and performance goals or to distribute workloads to reduce the effects of aging and to improve the reliability of the system.
To accurately control these subsystem knobs to achieve system goals, many previous approaches take a control-theoretic approach by identifying the system and using a control loop to control the system response [10][11]. However, this can be especially difficult when there are many knobs, complex interactions between the subsystems, and the objectives have multi-scale time horizons (e.g., power and application execution time). All of these challenges arise in the problem of dynamic resource management in manycore systems. In the future, we believe that machine learning techniques can aid in this problem and allow us to automatically create better run-time policies for on-chip resource management. As a case-study, we consider dynamic power management in manycore systems and show the advantages of recent advances in machine learning.

4.1 Challenges in Dynamic Power Management

Let us consider the problem of power management in the context of dynamic resource management. Dynamic voltage and frequency scaling (DVFS) techniques are commonly employed for implementing dynamic power management. DVFS attempts to reduce the overall power by estimating the performance requirements of each core and adjusting the voltage and frequency (V/F) to match those requirements. For example, DVFS can lower the V/F to reduce power without sacrificing performance during low-intensity workloads. Ultimately, the goal of DVFS is to minimize the overall energy without reducing the performance of the system.

Although DVFS has seen success in the early days of multicore architectures, they operate on a core-by-core basis and employ core-level information to tailor the V/F values of each individual core. This reduces DVFS’s ability to scale with core count due to more area and energy overhead in larger systems. To deal with continually expanding core counts in manycore systems, voltage-frequency islands (VFIs) have emerged as an efficient and scalable power management strategy [12]. In these designs, VFIs attempt to group multiple cores that have similar performance requirements with the hope that they can share the same V/F without significant performance penalties.

Like DVFS, a policy can adjust the V/F of VFIs (dynamic VFI or DVFI) to take advantage of dynamic run-time requirements. Although this addresses the scalability problem in DVFS, it makes the performance estimation and control problem more difficult. Since multiple cores now share the same V/F level, the policy must consider the performance requirements of all cores in the VFI and decide on an appropriate V/F level. However, this is a non-trivial problem regardless of how well the cores are clustered into VFIs since VFIs still exhibit significant intra-VFI workload variations. As a straightforward approach, we can take techniques that have been widely employed in DVFS, including feedback control and reinforcement learning (RL), and extend them to DVFI.

4.2 Drawbacks of Existing DVFS Approaches

Feedback-based DVFS approaches were found to be very capable in controlling the levels of one or two performance indicators (e.g., CPU utilization and communication [11][13]). We could simply extend feedback control to DVFI by using average VFI state information (e.g., average utilization and communication of the cores within the VFI). However, this may not be able to accommodate every core within the VFI, especially for VFI’s with high intra-VFI workload variance. Although there are techniques to help alleviate this issue like introducing a bias [14], applications that change dramatically during run-time would still have significant problems.

On the other hand, RL-based approaches have been used to incorporate more information about the state of the core to achieve better energy-performance tradeoffs [15]. By using a trial-and-error approach, the RL learner interacts with the system and learns which actions lead to the best cumulative long-term reward for each state. RL can be directly applied to DVFI by considering the necessary VFI features (e.g., average and maximum utilization, and communication of the cores within the VFI). However, RL policies take significant amounts of data and time to learn a good policy because the state-action space needs to be sufficiently explored to be able to select good actions. In addition, designing an appropriate reward function for DVFI is particularly hard since there is no intuitive way to determine which VFI (or VFIs) were responsible for any system-level performance penalty.

4.3 DVFI Control via Imitation Learning

We consider recent advances in machine learning to overcome the above challenges. Imitation learning (IL) in particular has seen much success in recent years [16][17][18]. In traditional IL, expert demonstrations are provided as supervised training data (e.g., demonstrations of a human expert driving a car). Then the learner tries to imitate the behavior of the expert in a way that generalizes to similar tasks or situations. These characteristics make IL an exponentially better framework than RL for solving sequential decision-making policies. At a high level, the difference between IL and RL is the same as the difference between supervised learning and exploratory learning [17][18].

In a supervised setting, the learner is provided with the best action for a given state. Therefore, when it is possible to learn a good approximation of the expert, the amount of data and time that is required to learn the IL policy is polynomial (quadratic or less) with the number of decision steps [16]. Since exploratory learning needs to search for the best action for a given state, near-optimal RL is intractable for large state spaces [17][18] and may not scale very well with the number of cores.

The main caveat of IL is that it assumes the availability of a good Oracle (expert) policy to drive the learning process [19]. Typically, this is done by collecting a set of trajectories of the expert’s behavior on a set of training tasks. Then, supervised learning is used to find a policy that can replicate the decisions made on those trajectories. Often the supervised learning problem corresponds to learning a classifier or regressor to map states to actions. To improve the DVFI policy, an advanced IL technique called DAgger [16] can be employed to create more robust DVFI controllers. Due to the nature of sequential decision-making problems, if the control policy makes a mistake, it can cause the system to go down unintended trajectories and exhibit poor
behavior. Essentially, DAgger mitigates this problem by generating additional training data demonstrating how the Oracle recovers from potential mistakes. With these techniques we can create robust and high-quality DVFI policies efficiently using IL.

To illustrate the practical differences between these DVFI policies, we implemented each policy on a 4-VFI 64-core system running the PARSEC benchmark fluidanimate as an example. Each DVFI policy would periodically look at the VFIs and adjust the V/F of each VFI. One important aspect in analyzing the DVFI control policies is their potential achievable performance. To help estimate this, we show the execution time and energy dissipation trade-off in Fig. 3(a). Due to the nature of the RL reward function tuning process, a set of execution times and energy tradeoffs are generated. To provide a clear picture, we also generate feedback control (FB) and IL policies at various performance penalties along the performance penalty range seen by the RL set. It can be seen in Fig. 3(a) that IL is able to consistently outperform both RL and FB. IL favorably shifts the pareto frontier to the lower left, achieving lower energy at each performance penalty point. In addition, IL can achieve this level of performance at controller area overheads similar to FB and much less than RL (Fig. 3(b)).

We do acknowledge that FB may be desirable to some designers due to low computational and hardware overhead. However, the computational overhead of IL is incurred only once per application during the offline learning process and the hardware overhead of IL only accounts for 0.0229% of the chip area for a four-VFI 400 mm² system. Due to IL’s favorable qualities in incorporating more VFI information, less hardware overhead and computational overhead, and better execution time and energy dissipation, we offer IL as a promising alternative to create efficient DVFI controllers [20].

4.4 Learning for Dynamic Thermal Management

Besides power management, another significant resource management problem for manycore systems is to maintain reasonable temperatures. If temperatures are left unchecked, it could cause large cooling costs in datacenter environments or system malfunctions in mobile designs with limited cooling capacity. However, due to the larger time granularity of temperature and major external factors like airflow and external temperature, it may be difficult to accurately predict temperature violations before it is too late. For example, although several techniques respond well to impending temperature violations [10], they are not able to prevent them entirely.

Similar to what we saw for DVFI, we believe that advanced machine learning techniques such as IL could be a useful tool to predict temperature violations early enough to prevent them from occurring. This will become more important for upcoming 3D integrated circuits that have increased power density and thermal issues. These systems, especially 3D CPU-GPU heterogeneous architectures, should be designed so that power-hungry cores are placed near the heat sink without compromising performance. However, this adds a clear tradeoff between the thermal efficiency and performance of the system (cores closer to the sink are further away from the rest of the system). This adds another dimension and significant complexity to the existing problems in thermal management [25].

4.5 Open Problems in Run-Time Optimization

We list three open problems for run-time optimization: a) Constructing the oracle policy is not straightforward for a given run-time control problem. How can we design generic online data-efficient learning algorithms to construct accurate policies to control a given knob; b) How can we easily incorporate hardware domain knowledge into the learning and control; and c) How can we perform coordinated learning and control of all the knobs jointly to enable fully-adaptive manycore systems?

5 Conclusions and Future Directions: Machine-Learning’s Role in Manycore System Design

From the last two sections, we see a clear trend. Machine learning has the potential to significantly improve the state-of-the-art in terms of optimization speed, solution quality, and scalability of hardware optimization. By incorporating expert guidance or past experience, machine learning has the ability to dramatically
reduce the amount of blind exploration that traditional techniques require. This has the potential to change the way we design computing systems. For example, current solutions that employ heuristics to reactively adjust to run-time conditions, e.g., adaptive network routing or branch predictors, can be improved with machine learning techniques that proactively make decisions. This can be similarly done with solutions that consider relatively short time horizons, e.g., real-time scheduling, to extend it to consider longer time horizons, e.g., for battery-powered embedded real-time systems.

Most importantly, machine learning techniques will allow us to significantly raise the complexity of these design optimization problems. Due to the intricate interactions between many elements of computing systems, it has not been possible to consider many of the design variables simultaneously. However, machine learning presents a potentially fruitful direction towards holistically optimized and fully-adaptive (co-design from transistors to software) systems in the near future [22].

5.1 Toward Domain-Specific Architectures

We believe that machine learning techniques will help power the next generation of computing systems, namely, domain-specific architectures (DSA) [21]. As the opportunities to improve hardware systems through traditional methods (i.e., transistor scaling, parallelism, etc.) disappear, the need to exploit other opportunities grow. Naturally, this leads us to designs that exploit heterogeneity [24] and specialization to achieve maximum efficiency. By focusing on a specific application domain, DSAs greatly improve efficiency over general-purpose designs while allowing some flexibility over ASICs. Machine learning techniques are a part of the solution to explore what design options are more efficient for particular applications domains. Ultimately, we expect that future research in machine learning and EDA will culminate in enabling the design of holistically optimized DSAs [22].

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation (NSF) grants CNS-1564014 and CCF 1514269, and US Army Research Office grant W911NF1710483.

REFERENCES