iLORE: Discovering a Lineage of Microprocessors

Samuel Lewis Furman

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science & Applications

Kirk Cameron, Chair

Godmar Back

Margaret Ellis

May 24, 2021

Blacksburg, Virginia

Keywords: Computer history, systems, computer architecture, microprocessors

iLORE: Discovering a Lineage of Microprocessors

Samuel Lewis Furman

(ABSTRACT)

Researchers, benchmarking organizations, and hardware manufacturers maintain repositories of computer component and performance information. However, this data is split across many isolated sources and is stored in a form that is not conducive to analysis. A centralized repository of said data would arm stakeholders across industry and academia with a tool to more quantitatively understand the history of computing. We propose iLORE, a data model designed to represent intricate relationships between computer system benchmarks and computer components. We detail the methods we used to implement and populate the iLORE data model using data harvested from publicly available sources. Finally, we demonstrate the validity and utility of our iLORE implementation through an analysis of the characteristics and lineage of commercial microprocessors. We encourage the research community to interact with our data and visualizations at csgenome.org.

iLORE: Discovering a Lineage of Microprocessors

Samuel Lewis Furman

(GENERAL AUDIENCE ABSTRACT)

Researchers, benchmarking organizations, and hardware manufacturers maintain repositories of computer component and performance information. However, this data is split across many isolated sources and is stored in a form that is not conducive to analysis. A centralized repository of said data would arm stakeholders across industry and academia with a tool to more quantitatively understand the history of computing. We propose iLORE, a data model designed to represent intricate relationships between computer system benchmarks and computer components. We detail the methods we used to implement and populate the iLORE data model using data harvested from publicly available sources. Finally, we demonstrate the validity and utility of our iLORE implementation through an analysis of the characteristics and lineage of commercial microprocessors. We encourage the research community to interact with our data and visualizations at csgenome.org.

Dedication

To my loving parents who support me in my every venture.

Acknowledgments

I would like to thank my advisor, Dr. Kirk Cameron. As his student, I have gained a great appreciation for the scientific method and a lifelong love for research. I would like to thank Professor Margaret Ellis for fostering my fledgling interest in undergraduate research and continuing to support me throughout my masters. I would like to thank Dr. Godmar Back for inspiring me and guiding me with his enthusiasm and expertise for a wide variety of computer science fields. I would like to thank colleague and friend Nicolas Hardy for working alongside me through thick and thin in and outside the lab for these past three years. I would also like to thank fellow researchers Lalitha Kuppa and Tanvi Haldankar for their support, advice, and hard work. Without their efforts, this work would never have come this far. I have been fortunate enough to be a part of large teams working on the Computer Systems Genome Project and projects in the SCAPE Lab. I would like to thank all these team members and lab mates for helping me develop my skills as a student and researcher. Thank you! This material is based upon work supported in part by the National Science Foundation under Grant No. 1838271, 1565314, and 1939076.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the inception of electronic computing, researchers have been interested in the growth of computational performance. This rich history begins with the world's first electronic Turing complete system, ENIAC, and continues today with contemporary supercomputers whose calculating power enables the exploration of many scientific domains. Due to the explosive growth of computational power, the world enjoys significant advances in chemistry, medicine, and physics. Advances in computational power have induced sweeping changes in everyday life in the form of transformative technologies like smartphones and Internet of Things (IoT) devices.

Intel's introduction of the first single-chip microprocessor, the 4004, in the early 1970s [1] is an important milestone in the history of computing. In contrast to the earlier mainframe computers and minicomputers that required the combination of multiple integrated circuits, the 4004 was implemented on a single metal-oxide-semiconductor field-effect transistor (MOSFET) integrated circuit (IC). Microprocessors, as self-contained computational devices, quickly became the foundation upon which many technologies, including the personal computer, thrived. The descendants of these first microprocessors are used today to power the largest supercomputers and the smallest embedded devices.

We are interested in exploring the evolution of computer systems and microprocessor performance in more detail. A central hypothesis of both this work and the larger Computer Systems Genome (CSGenome) project is that most modern systems share a common ances-

try. By leveraging this common ancestry and tracking the dissemination of design ideas, we hope to classify and catalog the lineage of computer systems.

1.1 Research Challenges

Any structure representing a lineage of computer systems must be constructed using three types of data: computer system, benchmark, and computer component data. Computer system data refers to information describing the identity and characteristics of all kinds of computational devices. Benchmark information tracks the performance of computer systems. Computer component information provides details on the design of the individual technologies that make up computer systems.

Although numerous online repositories of such data exist [2, 3, 4, 5, 6, 7], individual sources are restricted in scope to either benchmark, system, or system component data. These sources occasionally cross data boundaries like in the case of TOP500's semiannual highlights [8], but no existing source makes a concerted effort to link multiple computer components to performance benchmarks. The current siloed state of computer systems performance data makes it difficult to mine trends in the evolution of computer systems.

Using a data-based approach to track the lineage of computer systems is difficult because critical data is often unobtainable from primary sources. One reason for missing data is a lack of incentives. Component manufacturers receive no rewards for maintaining comprehensive descriptions of their historical offerings. It is rare for manufacturers to provide information on products they no longer offer to consumers. Moreover, many manufacturers have either gone out of business or been acquired. Thus, it is challenging to find accurate information on discontinued products.

Third-party organizations fill the gap created by the lack of primary sources and provide repositories of useful data with caveats. Many of these sources expose none [6, 9] or only some [10] of their data for consumption by other data collection efforts. These sources provide the full extent of their data for browsing, but put a ban on programmatic collection. Therefore, these sources function as useful references for hobbyists but lack utility when it comes to discovering trends in the history of computing. Moreover, out of the sources that fully expose their data [2, 3, 4, 5], only Stanford's CPU DB [5] makes an effort to provide data in a format that is easily amenable to analysis. The remaining sources package their data in formats that require significant wrangling before the full utility of the provided information can be realized. Overall, the usability of all existing sources is lacking in one way or another.

Because each repository of data is maintained by a different organization with different goals, specialties, and backgrounds, reconciling the differences between two or more data repositories is difficult. Combining data within and across the three major categories necessitates an understanding of the policies and approaches used by these disparate sources. Successful integration of sources requires sorting out the unique labels, units, and formats used to represent overlapping data sets.

The data provided by the aforementioned sources fall into one of the three major categories. System, benchmark, and component data all fail to directly address or capture the evolution of computer system design efforts and innovations. Although it is possible to discover proxies for innovation in existing data sets by analyzing trends in computer component attributes, no existing sources have taken the effort to document the dissemination of ideas in computer system design. In other words, no known source explicitly codifies a lineage of either computer components or computer systems.

1.2 Research Considerations

After reviewing other efforts to catalog the history of computer system performance and exploring alternative sources of benchmark, system, and computer component data, we believe that there is a considerable need for a solution that exposes a unified set of data augmented with a lineage.

We consider the following design criteria:

- Our solution must integrate benchmark, system, and computer component data such that relationships between data sets are represented.
- Our solution must include data describing the evolution of computer systems.
- Our solution must be easily accessible so that we can attract the largest number of users and facilitate rapid analysis.

1.3 Research Contributions

In pursuit of the goals of this thesis, we make the following main contributions:

- We propose, design, and implement the iLORE data model for storing and relating computer system, computer component, and benchmark data.
- We design and implement a RESTful API that exposes our data to the general public.
- We synthesize information from primary and secondary sources to record a lineage of commercial microprocessors.

- We use the resulting database to extend previous analyses of microprocessors and benchmarks to corroborate and gain confidence in our data.

- We use the resulting database to pursue new analysis questions.

1.4 The CSGenome Project

Our work is conducted as part of the creation of the CSGenome project's key artifact iLORE. To our knowledge, iLORE is the first attempt at capturing a lineage of computer system performance over time. The contents of iLORE will allow us to better understand the role that computational innovations have played in shaping our way of life.

The CSGenome project aims to study the evolution of computer systems architecture and performance. Many undergraduate and graduate students contributed to the CSGenome project and therefore this work. I will indicate where this work relies on the extensive efforts of these students. This thesis and my work for the CSGenome project focus on the design and implementation of iLORE, collection and preparation of data, creation of a microprocessor lineage, and analysis of historic trends in microprocessors.

1.5 Organization

This thesis document is organized in the following manner. Chapter 2 describes other works that attempt to gather or analyze data on the history of computing. This review allows us to place iLORE in a proper context and distinguish ourselves from our predecessors. Chapter 3 outlines the techniques we use to design and populate iLORE. Chapter 4 provides analysis to validate and demonstrate the utility of our data repository. In Chapter 5, we reflect on

the contributions made in this thesis and detail steps for iLORE and CSG moving forward.

Chapter 2

Related Works

This thesis focuses on the construction and analysis of a data repository of microprocessor and benchmarking data. In this section, we provide an overview of past efforts pertaining to the collection and analysis of computer component and benchmarking data with a focus on microprocessors.

2.1 Computing Retrospectives

Many key computing innovations are the result of industry efforts to design commercially competitive products. The engineers and designers of these products often go on to document their experiences and contributions. While these works are often light on technical details, they help us understand the how and why of early innovations. Retrospectives from early microarchitects and system designers make up an important and rich corner of computer history.

Faggin describes the world of integrated circuits prior to the microprocessor and lays the foundation for understanding the magnitude of Intel's release of the first commercially available microprocessor, the 4004, and the supporting 4000 family of auxiliary chips [1]. Faggin also describes the technological innovations in semiconductors, mostly the result of work at Fairchild Semiconductor, that made microprocessors feasible. The author provides insight into Intel's early design process and entry into the market. Overall, this paper provides the

context to understand the rise of both microprocessors and Intel.

Noyce and Hoff recount Intel's early days and the factors that led to the microprocessor revolution [11]. The catalyst for the invention of the microprocessor was a contract between the Japanese calculator manufacturer Busicom and Intel. Hoff concluded that Busicom's designs were too complex and opted to increase memory requirements using Intel's innovations in low-cost MOS memories and simplify the instructions supported by the chips. As he simplified the design, Hoff realized that a simpler, more flexible processor would have applications beyond calculators. The result of this effort was the 4004. The authors go on to describe Intel's efforts through the next three generations of processors ending with the 32-bit iAPX 432.

Morse et al. [12] provide some background on the development of Intel's early microprocessor offerings. The authors briefly mention the 4004, but focus on subsequent designs from the 8008 to the 8086. This work provides a high level comparison of the specifications for each early microprocessor. Moreover, the authors place each design in a historical context. For example, the 8080 came about due to an advancement in fabrication technology. Intel found that it was not feasible to reuse the 8008's layout and instead designed its successor, the 8080. The 8085 and 8086 were additional enhancements of the 8080 enabled by further advances in fabrication. With these successors, Intel was able to achieve an order-of-magnitude improvement in throughput and pull critical components on-chip, such as the oscillator.

More recently, Stanley Mazor recorded his experiences working on both the Intel 8080 and 8086 [13, 14]. Mazor recounts that these early Intel microprocessors were designed by small teams of three to four people with little oversight from marketing and management. Mazor describes the development process of the Intel 8800 CPU in addition to the 8080 and 8086. Because these works were released far after the events they describe, Mazor describes the legacy of the 8086 and 8080 with hindsight. He makes note of key competitors and Intel

spin-offs like the Motorola 6800 and Zilog Z80 and traces the 8086's lineage through to the Intel Pentium chips.

Gross et al.'s retrospective on the original MIPS architecture is another example of a more recent perspective on a key microprocessor innovation [15]. The authors describe the design constraints that pushed them to produce the single-chip RISC MIPS processor. They also describe the CAD tools that were developed in support of the MIPS project. Similar to Mazor, the authors highlight the legacy of the MIPS processor as a successor to future RISC architectures produced by MIPS Inc. and DEC.

Departing from the microprocessor-centric viewpoint used in previous sources, a substantial body of work focuses on entire computer systems. Evans describes the market forces and conditions that pushed IBM to unify their mainframe computing system to the System/360 product line [16]. Mazor provides historical context for Intel's microprocessor offerings through his documentation of the first microcomputers [17]. In this work, Mazor also reflects on technology predictions of the time by comparing transistor counts of early Intel microprocessors to a predicted curve. Finally, Mazor outlines the market forces that drive Moore's law and the continued innovation in microprocessor design [18].

The work of this thesis enables the verification of the impact of these historical contributions to computer science. Through analysis of the microprocessor lineage, we can better understand how key designs evolved. Moreover, we can attempt to quantify the impact of such advances by means of integration with performance and computer system data.

2.2 Microarchitecture Case Studies

Similar to the aforementioned retrospectives, there is a large body of work that explores in detail both important microarchitectural techniques and the individual designs which implement said techniques. In contrast to the previous category of work, these studies forsake providing a historical context to focus entirely on the technical details.

A significant body of work explores improving microprocessor performance through exploitation of the instruction-level parallelism inherent to most workloads. Researchers have conducted efforts to optimize pipelining, an important technique for achieving instruction-level parallelism[19, 20, 21]. These works explore the trade-offs that constrain the pipeline depth of a microprocessor. Hartstein and Puzak take an empirical approach by studying the correlation between pipeline depth and SPEC CPU benchmark scores for in-order and out-of-order superscalar processors [19]. Dubey and Flynn propose a mathematical model for calculating the optimal pipeline depth [20]. Sprangle and Carmean explore the implications that deeply pipelined designs have on other system components including those that make up the memory hierarchy [21]. Sprangle and Carmean also investigate the relationship between pipelining and other microarchitectural techniques such as branch prediction.

Branch prediction is another critical technique for exploiting instruction-level parallelism that has been studied extensively [22, 23, 24]. Smith provides a foundational overview of branch prediction strategies with the goal of maximizing prediction accuracy. Two other works describe more sophisticated branch prediction methods and explore the hardware cost to prediction accuracy trade-offs for different implementations of these methods. McFarling suggests combining several history-based branch prediction methods in an ensemble to capture the advantages of each individual predictor [24]. Yeh and Patt introduce an adaptive branch prediction method that makes decisions using two levels of historical branch outcomes

[23].

Another body of work examines techniques for achieving the relatively coarse grained thread-level and process-level parallelism. Tullsen et al. compare simultaneous multithreading (SMT) to traditional multithreaded processors and single chip superscalar processors [25]. The authors find that SMT enables greater utilization of functional units while necessitating more complex microprocessor designs. Hammand et al. compare chip multi-processors (CMP) and SMT and argue that CMP is better because implementing CMP requires less complicated and more easily optimizable hardware [26]. In reality, both of these techniques, often in combination, are utilized in modern microprocessors.

An alternative class of sources explores the implementation details of microarchitectural techniques in individual processors. Papworth describes the design process of the Pentium Pro microarchitecture with a focus on how the end product design greatly differs from the initial design goals [27]. Papworth describes trade-offs in area versus performance and clock speed versus design effort. The Pentium Pro leans heavily on pipelining to meet performance goals. Tendler et al. give a detailed report on the POWER4 microprocessor and system microarchitecture [28]. The IBM POWER4 uses superscalar execution and branch prediction to provide the maximum possible performance in both technical and commercial applications. Koufaty and Marr describe how Intel brought SMT technology to both the server and consumer market in the Xeon and Pentium 4 processors, respectively [29].

In the same fashion as the retrospectives mentioned in Section 2.1, the work of this thesis provides the tools for researchers to better understand the impact and proliferation of key developments in microarchitecture design. Performance and lineage data provide an additional lens through which we can view both individual chips and important design features.

2.3 Trends in Microprocessor Design

All of the works discussed up to this point are focused on a single topic or incident. Many studies take a much broader perspective on microprocessor design often in an attempt to steer further research in the field in a new and potentially more fruitful direction. These studies attempt to present and describe some observed trend in microprocessor design and/or make predictions about the future of microprocessors.

Microprocessor design has always been a multi-objective optimization problem. However, this set of objectives has shifted as microprocessors have evolved. Patt describes the evolution of microprocessor design at a high level [30]. Patt suggests that all advances in microprocessors including pipelining, on-chip caching, branch prediction, integrated floating point units, out-of-order processing, etc. are the result of one of three forces: shifting design requirements, bottlenecks or challenges, and good fortune.

A similar but more forward-facing batch of sources attempt to examine and understand trends in microprocessor design to better inform future design efforts [31, 32, 33, 34]. Kozyrakis and Patterson suggest that microarchitects could be more successful if they change their perspective and design for personal mobile computing instead of incrementally improving desktop and server applications [31]. Ronen et al. indicate that power concerns are likely to halt the technology shrinking that fueled microprocessor improvements up to the turn of the millennium [32]. Borkar corroborates Ronen et al.'s findings by suggesting that power will become a bottleneck as performance and density continue to scale [33]. A more recent source from Borkar and Chien suggest an increase in core counts and inclusion of heterogeneous cores as avenues to gain performance while contending with the power wall [34].

Microprocessor designers lean on multiprocessing to circumvent many of the ceilings discussed in the aforementioned works. A more recent body of work discusses the problems

that threaten to put a limit on single-chip process-level parallelism [35, 36]. Esmaeilzadeh et al. [35] indicate that mass adoption of multi-core designs is not a long term solution to the power wall and the failure of Dennard scaling [37]. The authors suggest that economics may halt multi-core designs well before manufacturing capabilities. Esmaeilzadeh et al. suggest that multi-core alone is not sufficient to carry performance gains while Moore's law breaks down [36]. The authors present pessimistic and optimistic perspectives and conclude that disruptive micro-architectural innovations and specialized hardware are the best bets for continued performance improvements.

Analyses using the data found in iLORE can be used to corroborate or invalidate predictions made in these kinds of works. Because of the relatively large time scales at which we have been able to collect data, iLORE is particularly useful when it comes to investigating how scaling trends play out over long periods of time.

2.4 Existing Repositories of Benchmark and Microprocessor Data

This work is certainly not the first effort to collect computer system, benchmark, or component data. Stakeholders including hardware manufacturers, benchmarking groups, other researchers, and hobbyists have come before us and created repositories containing these data types. The scope, focus, and accessibility of each of these existing repositories vary according to the goals of the party responsible for its creation.

The TOP500 supercomputing list [3] was launched in 1993. It ranks the 500 most powerful supercomputers twice a year, once in June and once in November. The TOP500 project aims to provide a consistent source of data capable of revealing patterns and enabling a deeper

understanding of developments in the HPC community.

A system's TOP500 ranking is determined by its performance running the LINPACK Benchmark [38]. The LINPACK Benchmark measures a computer's ability to solve a dense system of linear equations. LINPACK scores are highly dependent on floating point performance and are not at all universal measures of computational ability across all problem domains. The TOP500 authors acknowledge this shortcoming and provide an attribute that tracks which domains each entrant system specializes in.

TOP500 provides several metrics that represent entrants' LINPACK performance. R_{max} refers to the maximal LINPACK performance achieved in an experimental environment and R_{peak} refers to the theoretical peak performance [39]. Supercomputers are ranked by their R_{max} scores and ties are broken using R_{peak} scores. The TOP500 authors also provide the N_{max} and $N_{1/2}$ attributes that refer to the problem size at which the system achieved its R_{max} and half of its R_{max} respectively.

In addition to performance metrics, TOP500 provides several attributes that describe both the hardware specifications and the installation details of each of its entrants. For example, the "Installation Site," "Location," and "Field of Application" attributes allow consumers of the data to track who uses the most powerful supercomputers for what purposes. The "Memory," "Processor," "Accelerator/Co-Processor," and "Architecture" attributes, among others, describe a system's hardware specifications. In more recent TOP500 lists, the authors include the "Power" attribute as a nod to the rising importance of energy efficiency as a design criteria for HPC systems.

In an effort to encourage hardware vendors and system architects to abandon the "performance-at-any-cost" design paradigm that was common in the HPC community in the years following the turn of the millennium, the Green500 list[4] was introduced in November of 2006. The

Green500 list is a reordering of TOP500 systems using energy efficiency instead of raw performance as its primary criteria. Therefore, the biannual release of the Green500 list closely follows the release of each TOP500 list.

The Green500 list uses FLOPS/Watt as the metric for measuring power efficiency, but, in June 2013, the list adopted a more structured set of rules to govern the methodology of power measurement. This rule set [40] was published by the Energy Efficient High Performance Computing Working Group (EEHPC WG) in collaboration with the TOP500 and Green500 lists. Under these rules, power measurements are sorted into three levels of increasing accuracy and precision. All three levels are accepted as valid entries to the Green500 list.

The Standard Performance Evaluation Corporation (SPEC) [2] is a non-profit corporation that creates standardized benchmarks and maintains an open repository of results submitted by its member organizations. SPEC intends for its benchmarks to provide trustworthy, meaningful, and reproducible data upon which consumers can evaluate and differentiate candidate systems and components. SPEC manages benchmarks in many domains from storage devices to mail servers. Of particular interest to us is the family of CPU benchmarks [2] that measure processor, memory, and compiler performance.

The first SPEC CPU benchmark, SPEC CPU89, was introduced in 1989. The SPEC CPU suite has received occasional updates since its inception that displace earlier benchmarks in the family. SPEC CPU92, SPEC CPU95, SPEC CPU2000, SPEC CPU2006, and SPEC CPU2017 all replace their predecessors. Moreover, it is not uncommon for a particular benchmark to evolve within its own lifetime to respond to technical developments and the rising performance of computer systems.

In contrast to TOP500, the SPEC CPU family targets commodity hardware that is more

widely available to consumers. Moreover, instead of ranking its entrants using a single benchmark, the SPEC CPU family consists of a suite of many benchmarks designed to imitate the execution patterns of real applications. The structure and rules vary from version to version, but each SPEC CPU benchmark consists of a suite of integer benchmarks and a suite of floating point benchmarks. SPEC calculates a single score representative of either integer or floating point performance using a system's performance across the individual benchmarks that make up each suite. The more recent SPEC CPU benchmarks support both SPECspeed and SPECrate suites [41]. SPECspeed runs only one copy of each benchmark and higher scores are awarded for completing a benchmark in less time. SPECrate runs multiple copies of each benchmark. SPEC awards higher scores for greater throughput, or work accomplished, in a given unit of time. Some SPEC CPU benchmarks also allow for "base" and "peak" scores. Base scores require a uniform set of compiler optimizations for each microbenchmark. Peak, or "aggressive compilation," scores allow different, microbenchmark-specific compilation options.

Similar to TOP500, the SPEC CPU benchmarks provide information describing the hardware specifications and software environments of its entrants. SPEC CPU benchmarks additionally provide information on the compilation methods used to build the benchmarks. In contrast to TOP500, SPEC entries do not include data describing the installation locations of the systems being evaluated. Therefore, it is not feasible to mine SPEC records for geographic trends. Far more than 1000 submissions are made to SPEC every year. Therefore the volume of data from SPEC CPU benchmarks greatly exceeds that of data available from the TOP500 and Green500 lists.

The Stanford CPU DB is another source of microprocessor data that is invaluable for mining historic trends in microprocessor design up to 2014 [5]. The CPU DB provides performance characteristics like thermal design power (TDP) and clock frequency and detailed information

on the microarchitectures and technologies used to implement each processor. The CPU DB is the foundation for our processor attribute data.

The Stanford CPU DB distinguishes itself from other available sources in the breadth, depth, and organization of the data it contains. In terms of breadth, the CPU DB hosts information from a wide variety of manufacturers including DEC, IBM, Sun, and HP. Moreover, the CPU DB contains information regarding microprocessors from as far back as the early 1970s compared to manufacturer-maintained sources that rarely host information on products more than two decades ago. In terms of depth, the CPU DB contains detailed information on the process technology used to fabricate microprocessors including attributes like FO4 delay and metal layer counts. Finally, whereas other sources offer flat records, the CPU DB's schema exposes the relationships between different categories of processor attributes.

Danowitz et al. use the Stanford CPU DB to conduct analysis on the history of microprocessor trends [42]. This work provides insights into the changing nature of power density, performance, and architecture of microprocessors over several decades. The authors focus on comparing the impact of both fabrication and microarchitectural improvements on performance. To this end, Danowitz and others present a model to port chronologically distant chips to the same lithography for a fair comparison of microarchitectural quality.

Each of the previous attempts at collecting computer system, benchmark, and component data are lacking in two critical areas. First, no previous effort attempts to integrate data from multiple computer components with system and benchmark information. Second, no previous repository provides data in a form that is immediately amenable to analysis. Our work offers greater potential for analysis to a larger population of researchers via the integration of data sources and the convenience of a RESTful API.

2.5 Applications of the SPEC CPU Benchmark

As an important benchmark in the microprocessor industry, many efforts have been made to better understand the SPEC CPU benchmarks [2]. Of particular interest is the influence of various hardware components on performance outcomes. As the main metric for performance in this study, it is important that we understand the history of the SPEC CPU benchmarks in more detail.

Because each SPEC CPU suite of benchmarks is considerably different from the last, an important body of work explores and attempts to understand these differences. Dujmovic and Dujmovic [43] investigate the differences between three early generations of SPEC benchmarks: SPEC89, SPEC92, and SPEC95. According to this work, the earliest suite, SPEC89, only included 10 benchmarks. In subsequent generations, microbenchmarks were split into an integer and floating point suite. The authors find that the microbenchmarks that make up the later generations are both less redundant and are more complete in their evaluation of computational power. Henning picks up where Dujmovic and Dujmovic left off. Henning covers the factors that led to the expansion of the SPEC CPU suites from the first generation SPEC89 through SPEC2006 [44]. The SPEC CPU subcommittee maintains that more microbenchmarks enable the representation of more programming styles and application areas and that this greater representation outweighs the increased maintenance costs. Other forces influencing SPEC's growth, including changes in implementation languages and the adoption of more open source applications, are less pertinent to thoughtful benchmark design. Finally, Panda et al. highlight differences between SPEC2006 and SPEC2017 using principal component analysis (PCA) [45].

While Panda et al. use PCA to compare and contrast two SPEC suites, many works instead use PCA to evaluate the efficacy of a single suite [46, 47, 48, 49]. This body of work is

particularly germane to microarchitects because the SPEC suites are often used in simulation settings for developing new microarchitectures. If a subset of microbenchmarks are predictive of the overall suite, microarchitects can more quickly iterate by only simulating that crucial subset. The hardware industry also has a vested interest in ensuring that SPEC suites provide good coverage and truthfully capture the computational capabilities of the evaluated components. PCA indicates that, while suites are improving with each generation, SPEC2000 is still highly redundant [48, 49]. Moreover, it is possible to use roughly half of the SPEC2006 benchmarks to predict the results of the full suite, but doing so may result in the loss of some information [46, 47].

The SPEC CPU suites measure the performance of memory hierarchies and compilers in addition to processors. There is a body of work that characterizes the interplay between the SPEC benchmarks and the memory system. Gee et al. [50] collected data for cache misses across the SPEC92 benchmark suite. The authors find that instruction and data misses for integer benchmarks are below other reported figures. However floating point misses are relatively higher and are more consistent with other reports. Charney and Puzak [51] and Sair and Charney [52] find that only a few microbenchmarks place a considerable load on the memory hierarchy in the SPEC95 and SPEC2000 suites respectively. Jaleel et al. [53] use instrumentation-driver simulation to characterize the memory behavior of the SPEC2006 benchmark suite. The authors find that, relative to SPEC2000, SPEC2006 uses an order-of-magnitude greater number of instructions and imposes a larger memory load. Singh and Awasthi [54] show that SPEC2017 instruction counts are an order-of-magnitude higher than those of SPEC2006 and that most SPEC2017 workloads have a working set larger than 32 MB, a reasonable last level cache size, but smaller than 5 GB.

In contrast to other works that attempt to make architecture-independent observations about SPEC benchmark suites, some works provide great detail on the SPEC performance charac-

teristics of individual microarchitectures [55, 56, 57]. These sources use performance counters to discover the interplay between SPEC CPU benchmarks and important microarchitectural features like branch prediction techniques, cache sizes, out-of-order execution, and super scalar execution. In pitting AMD Ryzen 5 processors against eighth generation Intel Core i7 processors, Oi [56] finds that the two processor groups trade blows in power and performance across floating point and integer benchmarks.

The iLORE repository enables further and more comprehensive studies of these kinds. By integrating computer component and SPEC data, we can more clearly analyze the relationships between SPEC CPU suites and computer components. Overall, the combination of data found in our database enables a mutual improvement in our understanding of both the SPEC CPU benchmark and microprocessors.

2.6 Understanding the LINPACK Benchmark

Whereas the SPEC CPU benchmarks serve as fundamental metrics for microprocessor designers, the TOP500 or High Performance LINPACK (HPL) benchmark [38] is the most widely utilized metric for ranking HPC systems.

The LINPACK User's Guide [58] informally introduces the LINPACK benchmark in the 1970s. The appendix of the user guide includes performance data across 23 machines for solving a fixed-size linear equation using the LINPACK software package. The LINPACK authors provided this data to give users a method to estimate LINPACK's performance on their own machines. Because of the somewhat accidental introduction of the LINPACK benchmark, Dongarra followed up with a work that clarifies the structure and meaning of benchmark results [59]. The benchmark proved to be useful throughout the 1980s and into the early 1990s [60]. The authors of the TOP500 selected the LINPACK benchmark

because of its widespread use [38]. Instead of the fixed-size problems used by previous implementations of the benchmark, the authors of the TOP500 opted to use a version of the benchmark that supports arbitrary problem sizes and parallel execution across multiple hardware devices [61].

Despite its longevity and eminence in the HPC community, the TOP500 list has come under scrutiny for its use of the LINPACK benchmark. Some alternatives have been suggested to address the common criticism that the benchmark has become increasingly divorced from real application performance [62, 63, 64]. Dongarra et al. [62] propose the HPC Challenge (HPCC) benchmark as an alternative to LINPACK. HPCC requires varied access patterns designed to expose inefficiencies in the memory subsystems of HPC systems. Heroux and Dongarra propose the high performance conjugate gradient (HPCG) benchmark as another alternative [63, 64]. HPCG is designed to mimic the computation and data access patterns of real scientific applications.

Although we do not conduct analysis using the LINPACK benchmark in this thesis, iLORE's integration of component data and LINPACK data enables a deeper exploration of the supercomputing benchmark. The combination of LINPACK with other benchmark data sets is one promising avenue of research that we mention in Section 3.6.

2.7 Summary

The works reviewed here all fall short in accomplishing one of the following: integrating multiple kinds of data, openly providing access to their data, or incorporating a direct measurement of the lineage of design ideas in computing.

We identify three critical categories of historic computer data: component, system, and

benchmark data. The discussed references incorporate at most two of these three categories in their analysis. Failure to properly integrate these bodies of information causes existing sources to miss out on the insights that can be gained from analysing the complex performance relationships between different parts of a computer system.

The quantitative analyses conducted in many of the reviewed works [31, 32, 33, 34] are either difficult or impossible to reproduce because the data used to conduct said analyses is not provided to the research community.

The CPU DB [42], is a notable exception in that the authors provide their raw data in the form of CSV files. However, processing these CSV files requires an understanding of a reasonably complex schema and legwork to build out a series of tools that consume the custom schema. Because we collect computer system and component information from 2015 up to the present, our information is both broader in scope and more complete than that found in the CPU DB.

Each of these sources describe some piece of computer history but all fail to precisely describe either the ancestors or descendants of the systems and microprocessors that were subjected to analysis. The inclusion of a direct lineage would greatly augment many of the reviewed references.

Chapter 3

Methods

In this chapter, we describe the techniques we use to collect and prepare data, design and populate the database, and perform analyses using the database.

3.1 Requirements and Specifications

In service of the fundamental goals of the CSGenome project, we considered the following criteria when designing our data model:

- The data model must be able to represent rich lineage and taxonomy relationships between and among microprocessors.

- The data model must be able to represent upgrades to a system's hardware configurations between benchmarks.

- The data model must be able to represent the hardware configurations of now commonplace heterogeneous computer systems that leverage accelerators in addition to general purpose microprocessors.

3.2 Data Collection

We use this section to describe the tools and techniques that we use to collect data for the iLORE database. Figure 3.1 outlines the process that we use to collect and prepare our data.

In line with our goal to capture and represent the relationship between computer systems and their constituent components, we collect data of several types using common scraping and extraction techniques in concert with manual efforts. We collect computer system, benchmark, and processor attribute data using Python scripts that leverage the popular Beautiful Soup and Pandas libraries [65, 66]. We lean on manual efforts to discover both processor taxonomy and lineage data. This section describes both the scraping and manual lineage synthesis steps pictured in Figure 3.1. Data collection occurs in parallel with the data preparation and schema design efforts detailed in Sections 3.3 and 3.4.

3.2.1 Computer System and Benchmark Data

Computer system and benchmark data are records that combine system and component hardware specifications with some measure of computer performance. Repositories of computer system and benchmark data are maintained by different organizations and each target a particular tranche of computer systems and utilize their own methodology for evaluating entrants. Some of these repositories expose their data for manual review and browsing but forbid programmatic collection. We gather data from several open repositories of computer system and benchmark data. See Section 2.4 for information on the design of these benchmarks.

TOP500 TOP500 publishes its biannual lists on its website and in the form of downloadable XML and EXCEL files. We download the EXCEL file corresponding to every list from

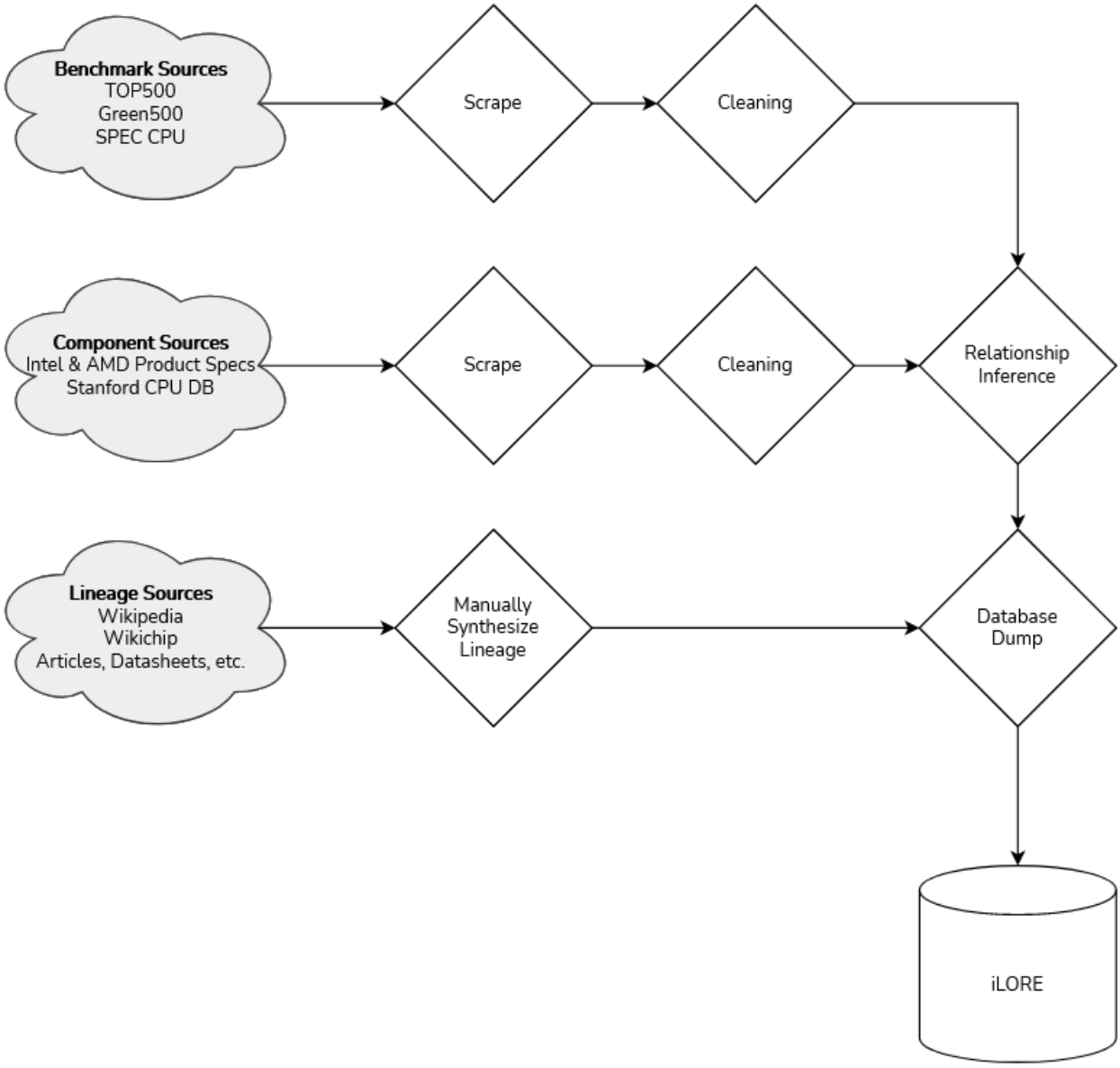

Figure 3.1: The high-level cleaning pipeline. We pull from benchmark, component, and lineage sources. Data from all three sources is used to populate the iLORE database.

June 1993 to the present day. Using the xlrd and pandas libraries [66, 67], we convert the given EXCEL files into more easily manipulated CSV files. CSV files are used due to their conceptual similarity to the pandas dataframe objects we use to manipulate data in memory. It would be equally valid to store intermediate TOP500 data files in another form that does not have a rigid schema and is easily processed like a JSON file. Overall, TOP500 is a high-quality source of data that requires few data cleaning operations. Notable operations are described in detail in Section 3.4.

Green500 Similar to TOP500, Green500 data is available for download as XML and EXCEL files. Identically to TOP500, we download files containing every Green500 list dating from its inception to the present day to disk before extracting and storing the data in the more easily manipulated CSV file format. Notable operations are described in detail in Section 3.4.

SPEC The Performance Database Server provides partial records of submitted SPEC CPU92 results [68]. SPEC provides full access to all results for suites since SPEC CPU95. This data is available in many forms including HTML, CSV, and text files. We download all available HTML files to disk and extract valuable information and store it in intermediate CSV files using the Beautiful Soup library [65]. The SPEC CPU benchmarks are reasonably high-quality data sources that require some cleaning operations. Details are provided in Section 3.4.

3.2.2 Processor Attribute Data

Processor attribute data describes the performance and design characteristics of general-purpose processors. Processor attribute data is open to the public in several different forms

from both processor manufacturers and independent curators.

Unfortunately, many independent curators of data disallow the programmatic or manual collection of their data. CPU-World [6] is one notable collection of data that is not available for our purposes. CPU-World was initially a platform used to display the processors owned by a single CPU collector. In recent years, CPU-World's information has grown in detail and scope. Independent chip collectors contribute most of this new information.

We gather processor attribute data from three key sources. The first two are online repositories of product specifications maintained by the microprocessor manufacturers Intel and AMD [69, 70]. The final source is a previous effort made by Danowitz et al. to construct a database of microprocessors that is useful for investigating trends in the history of microprocessors [5]. In terms of processor attribute data, the iLORE database goes well beyond the efforts made by Danowitz et al.

Stanford CPU DB The contents of the CPU DB are available for download as a collection of CSV files. Each CSV file corresponds to a traditional SQL table in the CPU DB. To join the Stanford CPU DB with other sources of processor and benchmark information, we fully join the downloaded CSV files into a single flat CSV file. Based on the contents of the input CSV files, we infer and reconstruct the CPU DB SQL schema. We augment this schema with an additional destination table populated with the resulting flat records.

We dump each CSV file into its corresponding table in the SQLite database. We use select, join, and insert operations to construct and populate the additional destination table. Each flat record in the destination table is a self-contained description of a processor stitched together from the CPU DB's tables. Finally, we save the contents of the destination table in a CSV file for ease of processing further down the data pipeline. Overall, the CPU DB is an incredibly high-quality source of data that, in some cases, cannot be obtained anywhere

else. Please refer to Section 3.4 for details on cleaning the CPU DB.

Intel Ark Intel provides detailed specifications on a wide variety of their more modern hardware offerings. Intel organizes its product specifications in a hierarchical structure. At the top level, Intel separates its hardware products into broad categories. Processor information is further segmented first by families like Core and Xeon and second by generation or series. Each processor has a page that contains traditional attribute information like clock frequencies and core counts and supplemental information like integrated graphics details. We are primarily interested in general-purpose processors and co-processors.

The collection of Intel specification data occurs in two steps. Starting from the root page, we crawl Intel's product specifications. First, we download the HTML source file of each processor page to disk. Second, using BeautifulSoup and Pandas [65, 66], we extract relevant attribute information from each HTML file for storage in a CSV file.

AMD Product Specifications AMD provides attribute information for their modern microprocessor products. Relative to Intel, AMD's records are limited to more recent processors. Whereas Intel provides information on processors dating back more than ten years, AMD only provides data from around the last five years. Fortunately, AMD offers their data in bulk in the form of a CSV file. Therefore, we can avoid crawling and scraping and instead download the provided CSV file. Relative to Intel, AMD's specifications are often less detailed and lack analytically valuable attributes like transistor counts.

3.2.3 Processor Lineage Data

The CSGenome project hypothesizes that modern computer systems share a common ancestry. In support of this hypothesis, we catalog the lineage of microprocessors dating back

to their commercial introduction in the mid-twentieth century.

Preliminary Lineage Discovery Our approach to processor lineage creation is based on the intuition that direct processor descendants are most often siloed by designers. In other words, cross-pollination of design ideas is rarer, but not absent entirely across different design shops. We set out to make an initial attempt to discover the parent and child relationships that exist between processors.

Initially seeking a designer with only a small number of released microprocessors, we selected Hewlett-Packard and their PA-RISC line of chips. Hewlett-Packard released around a dozen chips in this line over 14 years from 1991 to 2005 [71]. Using information gathered from Wikipedia and individual processor data sheets, we pieced together a linear ancestry from the PA-7000 to the PA-8900. This ancestry is shown in Figure 3.2.

During this preliminary phase, we modeled the ancestry of microprocessors as a collection of rooted trees. Under the common graph theory definition, a tree is a connected, undirected, and acyclic graph. Nodes in the tree represent processors and edges represent relationships between the incident nodes.

During this exploratory design phase, we manually assembled the lineage structure in the diagramming tool Lucidchart [72]. We used boxes to represent individual processors and arrows to indicate parentage between related processors. At this point, we established two practices that we carried forward to the discovery of the final lineage. First, as we conduct manual research and slowly piece together parent-child relationships, we record the sources of information used to inform our decisions. Second, we make small notes that summarize the nature of the relationship between two processors.

Satisfied with the basic methodology we established in our work with the PA-RISC chips,

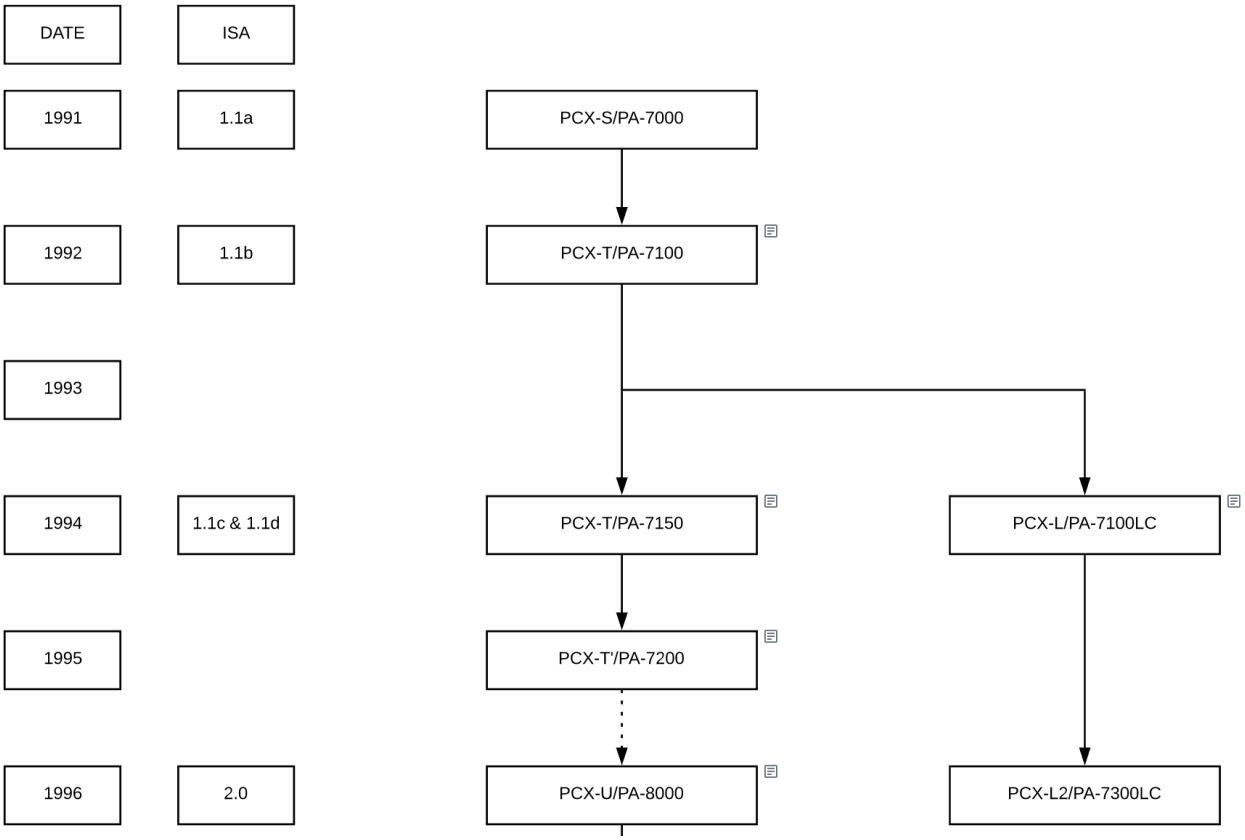

Figure 3.2: A manually created tree of PA-RISC microprocessors. We track instruction set architecture and release date on the left. The dotted line between the PA-7200 and the PA-8000 indicates a lack of confidence in that relationship.

we moved to discover the lineage of the more voluminous SPARC and POWER chips. Once again, Wikipedia presented a reasonably comprehensive framework to understand the history of both SPARC and POWER processor offerings. Armed with a general knowledge of these processor families, we used datasheets, internal manufacturer presentations, and other supplementary sources to fill in the gaps and discover the processor lineage. This effort proceeded relatively unencumbered. However, especially with the POWER chips, manually maintaining a visual diagram of the lineage structure was becoming cumbersome. Each update would often require the manipulation and reorganization of several entities. Early SPARC and Power processor trees are presented in Figures 3.3 and 3.4 respectively.

At this point, we realized that continuing to work in Lucidchart was unsustainable. Maintaining a visual diagram was cumbersome, allowed errors to hide, and prevented programmatic analysis. We opted to translate the diagrams we had constructed into CSV files. Each row in a file described a single parent-child relationship. To verify our work, we used the graph visualization software GraphViz to occasionally render and review a set of parent-child relationships as a figure [73]. This change in workflow allowed us to iterate more quickly and represents a step towards the final design of the data model described in Section 3.3.

The lineage of POWER chips also represents the first occurrence of a processor inheriting design ideas from multiple predecessors. IBM's POWER3 processor descends from both the POWER2 Super Chip (P2SC) and the PowerPC 620. IBM released the P2SC in 1996 as the next installment in its flagship line of processors. The P2SC packs the eight chip design of its ancestor, the POWER2, on a single die [74]. In partnership with Apple and Motorola, IBM released the PowerPC 620 in 1995. The PowerPC 620 is the first workstation product in the PowerPC family. The PowerPC 620 maintains backwards compatibility with the Power PC 601, 603, and 604 while supporting both 32-bit and 64-bit applications. The POWER3 combines the PowerPC architecture with the floating point processing capabilities of the

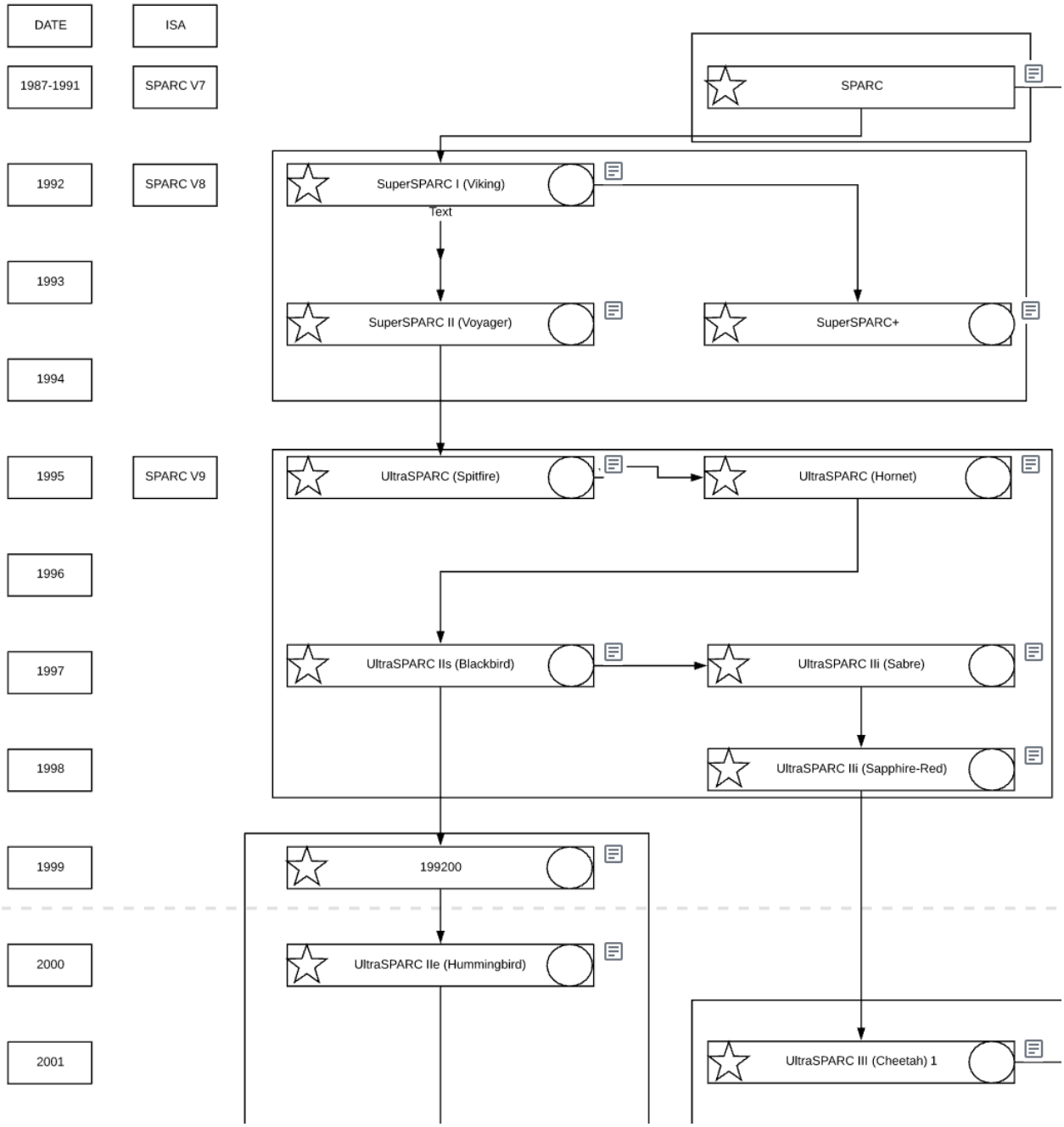

Figure 3.3: Manually created tree of Sun Microsystems SPARC Processors. Boxes around processors are an early attempt at grouping processors into a taxonomy. We used shapes to indicate the designer of each processor (e.g. stars were used for Sun Microsystems).

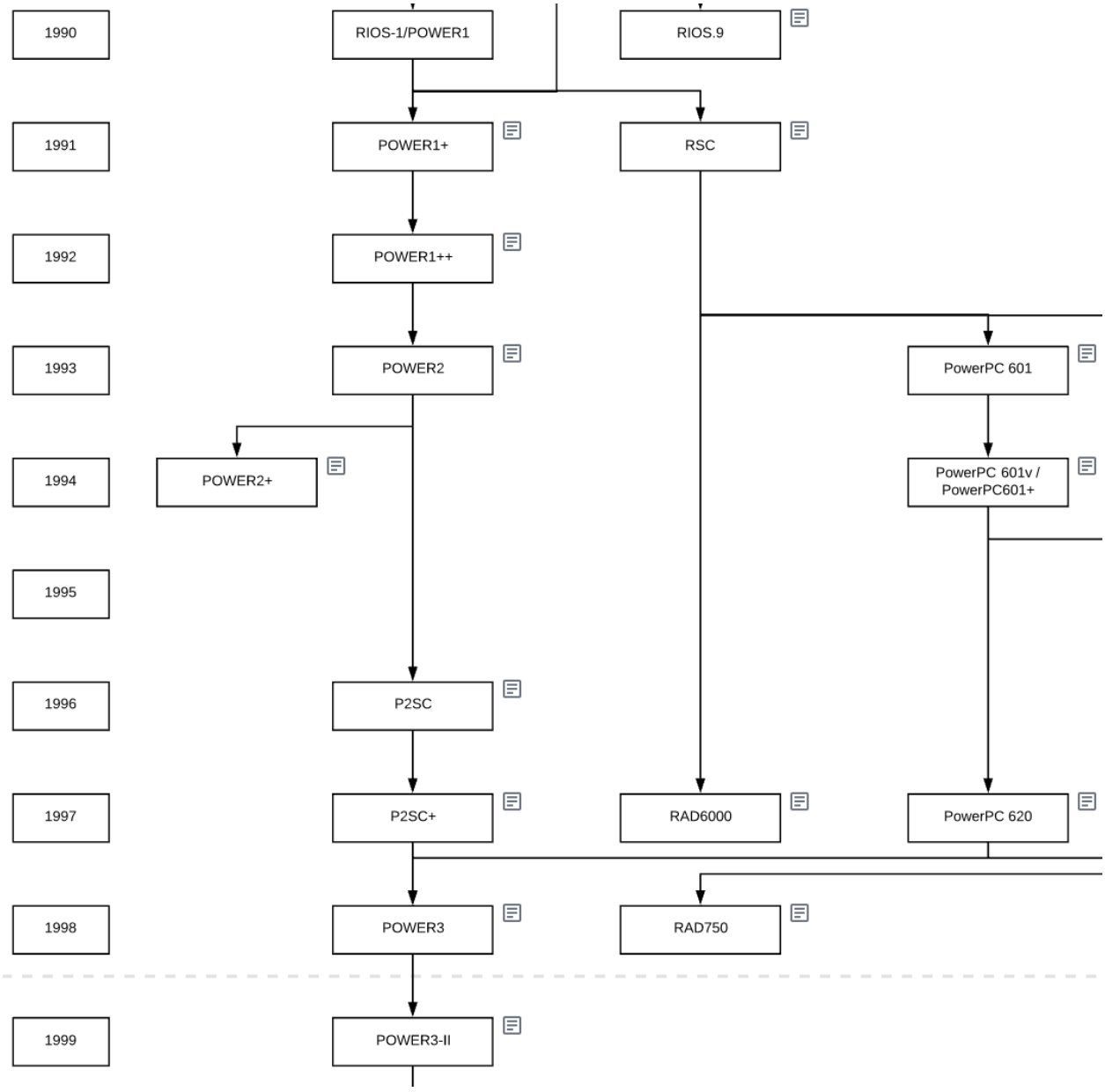

Figure 3.4: A manually created tree of IBM POWER processors. The processors show represent only a subset of POWER processors. The leftmost column of processors represents the lineage of IBM's flagship line.

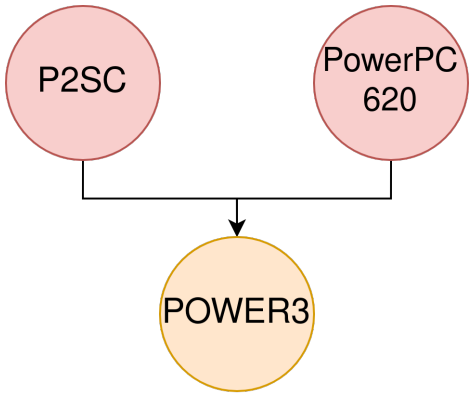

Figure 3.5: A multiple inheritance relationship in the IBM POWER tree of processors.

P2SC [75]. This instance of multiple inheritance is demonstrated in Figure 3.5.

Whereas we once thought of processor lineage as a tree, exploration of the POWER processors revealed that microprocessor history is more accurately modeled with a collection of directed acyclic graphs (DAG). Compared to a tree, the directed nature of a DAG allows for multiple inheritance. This aspect of the processor lineage complicates the design of our data model and necessitates additional care when it comes to visualization.

Dealing With Complexity Using a Taxonomy Coming off the completion of the POWER line, we decided to catalog the relatively large history of AMD and Intel processors. The volume of offerings from both AMD and Intel dwarfs the previously cataloged lines. However, we started discovering lineage relationships among these manufacturers in much the same way. Wikipedia still proves to be a solid jumping-off point to get a good grasp on historic Intel and AMD products. Moreover, data sheets on Intel and AMD products are more readily available than datasheets for now-defunct manufacturers. These data sheets, in addition to alternative secondary sources, prove useful in deciphering the complex relationships between processors.

Intel and AMD are the first lines to contradict our early assumption of isolation in design

ideas across manufacturers. Early in Intel's and AMD's histories, AMD released the AmX86 line of processors as a second-sourced product from Intel [76]. This crossover starts with the AMD Am186 and Intel 8086 chips and ends with the 8086 being the ancestry of all modern x86 processors.

Our goal in cataloging the lineage of microprocessors is to create a written record of the inheritance of key microprocessor design innovations. In our work leading up to Intel and AMD, we found that documenting relationships between individual microprocessors served this goal well. Each of the previous companies put out easily distinguishable products that often represented reasonable departures in design. On the other hand, Intel and AMD have released many processors that are, from a design perspective, not noticeably different. The sheer number of relatively similar processor models motivates the grouping of individual processors into larger groups.

We initially reached for common schemes used to group processors including: families, microarchitectures and instruction set architectures (ISAs).

Manufacturers use processor families to group products in a similar market segment together. For example, Intel's long-running Xeon family is primarily intended for use in servers. The designs of Xeon processors released in the early 2000s are far different than the designs of modern Xeon offerings. Because processor families provide little information on a processor's design, they do meet our criteria for a grouping in our taxonomy.

An ISA is a specification that describes the structure and behavior of machine code for an abstract computer. ISAs are commonly used to group and understand processors. Individual processor manufacturers tend to develop and update only a single small collection of ISAs. While Intel has made several commercially unsuccessful attempts to branch out with ISAs like iAPX 432 and i860 [77, 78], the vast majority of their microprocessor offerings continue

to implement x86 even after four decades. ARM represents an alternative business model. ARM licenses its ISAs across manufacturers.

Because designers use the same ISAs for many design cycles, they are often updated or augmented with modular, and sometimes specialized, bundles of new instructions referred to as extensions. To illustrate, Intel developed a series of Streaming SIMD Extensions (SSE) to add single instruction, multiple data (SIMD) operations to x86 [79]. Moreover, because the same ISA is used for long periods of time, implementations of the same ISA have the potential to vary wildly. Therefore, grouping individual processors directly into models results in the loss of critical information.

ISAs prescribe no hardware design requirements and offer no implementation details. Microarchitectures are the actual hardware implementation of ISAs. The same ISA can be implemented using significantly different techniques with meaningful consequences for performance and power. For example, Intel's Nehalem and Westmere designs are two microarchitectures that implement nearly identical sets of instructions. Nehalem and Westmere form a pair in the Tick-Tock production model that Intel used from 2007 to 2016 [80]. Under the Tick-Tock production model, Intel alternated new microarchitectures and process shrinks. The Westmere microarchitecture is primarily a die-shrink of Nehalem. Westmere represents a significant design effort and a microarchitectural departure relative to Nehalem.

Because processors within the same microarchitecture feature similar designs, grouping processors into microarchitectures is appealing at first glance. However, Intel and AMD make updates to each microarchitecture in the form of core generations and core steppings [81, 82]. Core generations are small updates to an existing architecture that are made to improve or optimize performance characteristics. Core steppings are more minute changes designed to remedy errata in earlier processor batches. Athlon:Argon and Athlon:Pluto are two core generations that use the same microarchitecture. Athlon:Pluto is the core generation following

Athlon:Argon. Therefore, grouping processors directly into microarchitectures still results in an imprecise tracking of the lineage of design decisions.

Because common methods for grouping processors are ill-suited to our task, we propose a new method for grouping processors. Leaning on the taxonomic classification system used in biology, we name the layer above processors genus. In biology, a genus is the second most granular taxon, sitting directly above species. In our taxonomy, microprocessors function as the species that are grouped into genera. The genus level captures the minor differences between processors that occur during core generations, core steppings, and binning.

By design, each genus can be placed into a microarchitecture. This feature allows us to extend the two most granular layers of the taxonomy with the commonly used microarchitecture and ISA schema. The design of our final taxonomy features four levels from least to most granular: instruction set architecture (ISA), microarchitecture, genus, and individual processor model. An example of a microprocessor being classified using our taxonomy is provided in Figure 3.6. Intel's i7-3770 is grouped into the Core i7:Ivy Bridge genus to distinguish it from differently binned models. The processor implements the Ivy Bridge microarchitecture which in turn implements the x86-64 instruction set.

Determining the lineage at processor genus level and above instead of at the model level allows us to mitigate the complexity of Intel and AMD's large portfolios while capturing the significant design innovations.

3.3 iLORE Schema Design

iLORE Schema design occurs in parallel with the data collection and data preparation efforts detailed in Sections 3.2 and 3.4. Each step informs the others. We craft our data model to be

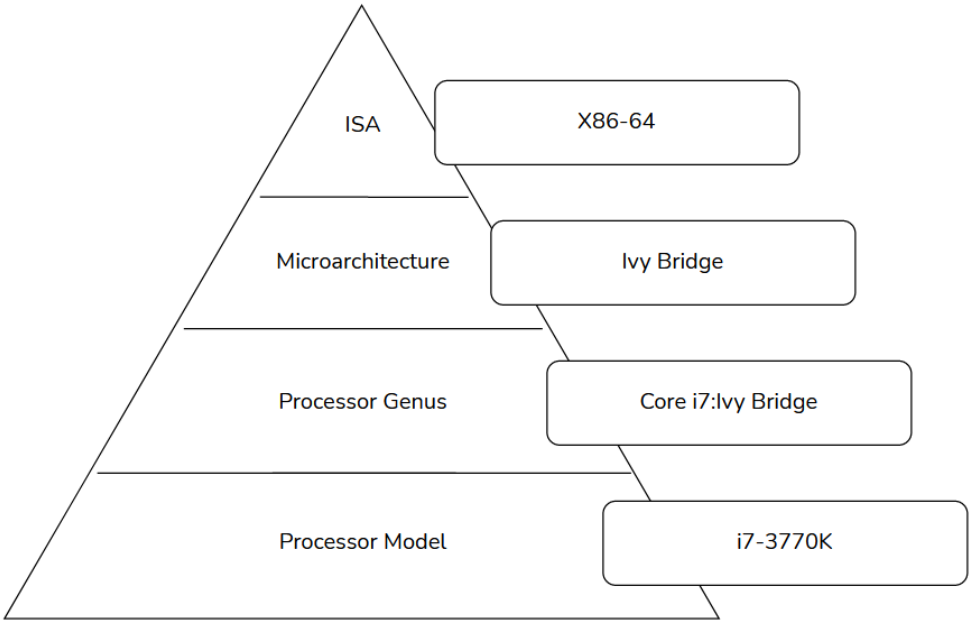

Figure 3.6: An example of placing a microprocessor in our taxonomy.

both rigid enough to store the data that is available to us and flexible enough to accurately model both future data from our current sources and data from untapped sources.

3.3.1 First Iteration

The original data model was relatively limited in its design. We designed the original data model with the sole intent of storing data from our early sources [2, 3, 4, 5]. A simplified version of this model is presented in Figure 3.7.

In this first schema, a system was minimally defined by a unique identifier and its model name. Optionally, system records included monikers, system families, an operating system configuration, and a flag to check if the record in question represented a cluster.

Systems could run one or more benchmarks that are described in benchmark-specific tables. Although the details of these benchmark tables cannot be observed in Figure 3.7, each

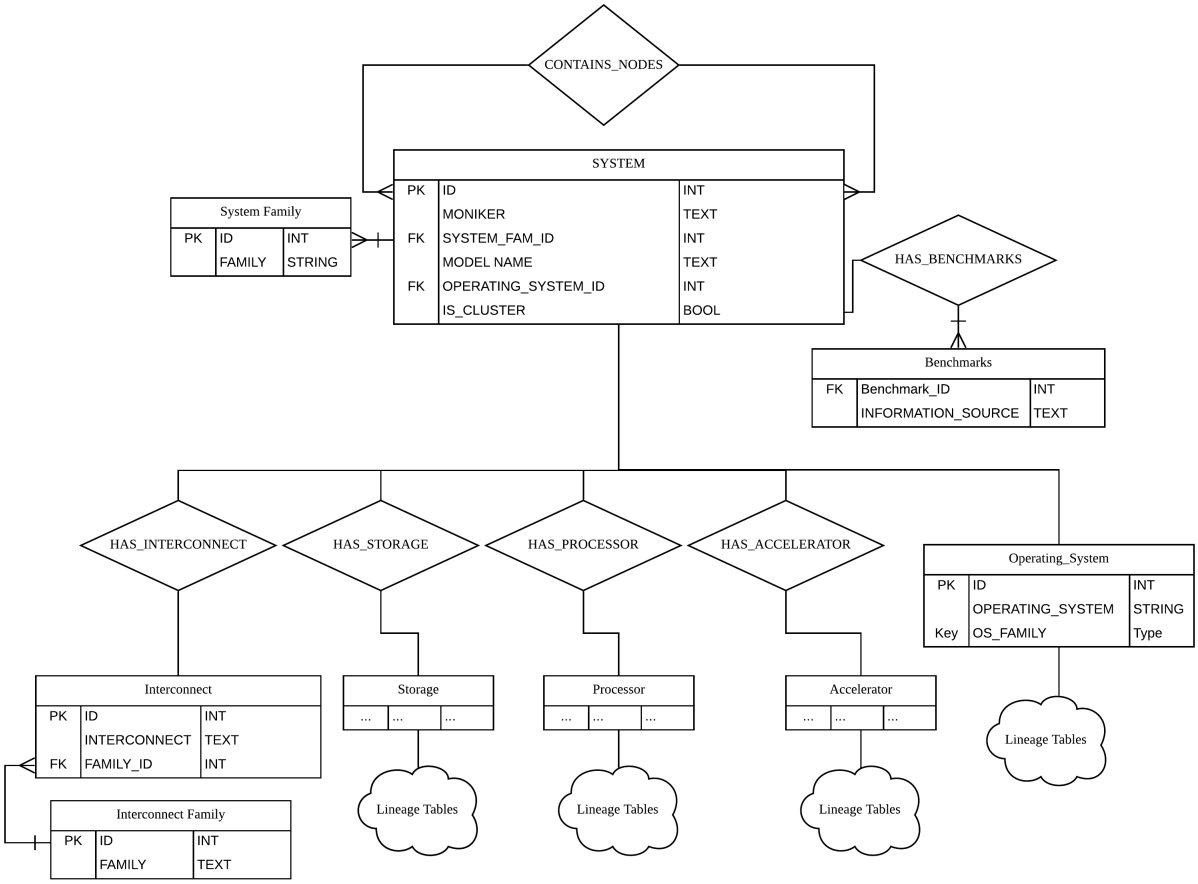

Figure 3.7: The first iteration of the iLORE data model. The system table functions as the core of this design. Operating systems are directly incorporated into the system table. Benchmark and hardware configuration tables branch off of the system table.

benchmark table minimally included a date and a performance score.

Hardware configurations were represented using many-to-many relationships with the system table. Each component was represented using its own sub-schema that includes lineage tables. The format of each lineage table varied according to the component.

This model was promising in that it could flexibly represent heterogeneous hardware configurations and enable the combination of individual component lineage data with benchmarking data. However this model failed to properly represent systems that run benchmarks with different hardware configurations. Hardware configurations were not baked into the description of the system, i.e., the schema legally allowed for the same system to be linked with multiple hardware configurations. This oversight introduced problems when it came to joining benchmark and component data. Benchmark records associated with a modified system would be joined with both the old and updated hardware configurations. In other words, a system that swaps out its processors was modeled identically to a system that strictly adds processors.

Despite its shortcomings, some characteristics of the first model persist in our current model. Because the redesign was primarily motivated by a need to represent more complicated relationships between computer systems and their components, the designs of individual component tables remain relatively unaffected. Therefore, processor tables in their current form are nearly identical to the processor tables of this first model. Moreover, the tables for individual benchmarks are also largely the same.

3.3.2 Final Schema Design

Based on the observed data, the same computer system can have multiple hardware configurations over its lifespan. The final iLORE schema is motivated by the following question:

if system maintainers upgrade or replace some of the components in a given system, are the old and updated systems fundamentally different entities? The iLORE data model maintains that, as long as two systems share a name, owner, and location, those two systems are identical and should be modeled as such. The final schema is provided in Figure 3.8.

In line with this thinking, the focal point of the iLORE data model is the ternary relationship between systems, system configurations, and benchmarks. A system may use one or more different system configurations to run one or more benchmarks during its lifetime. This flexible ternary relationship addresses the first iteration's failure to properly model systems with changing hardware. Under the final schema, hardware configurations are independently stored and connected to benchmarking data. Providing a greater separation between a system's identity and its hardware configuration enables the accurate representation of a system at different points in its lifetime.

The system table shown on the left of Figure 3.8 contains identifying information outside of hardware component specifications. Several attributes describe the purpose or application domain of the system. Others describe the system's location and designer. The "moniker" and "computer" attributes function as descriptive pet names for systems that have them like "Fugaku."

The system configuration table shown in the center of Figure 3.8 is a tuple of the hardware and software components utilized by the system: processor, memory, accelerators, and operating system. In support of the flexible representation of heterogeneous computing systems, each member of the system configuration tuple is itself an object of configuration information specific to each component. For example, the id stored in the processor field of system configuration represents an arbitrary number of arbitrarily many different processors.

The benchmark table pictured to the right in Figure 3.8 contains basic metadata indicat-

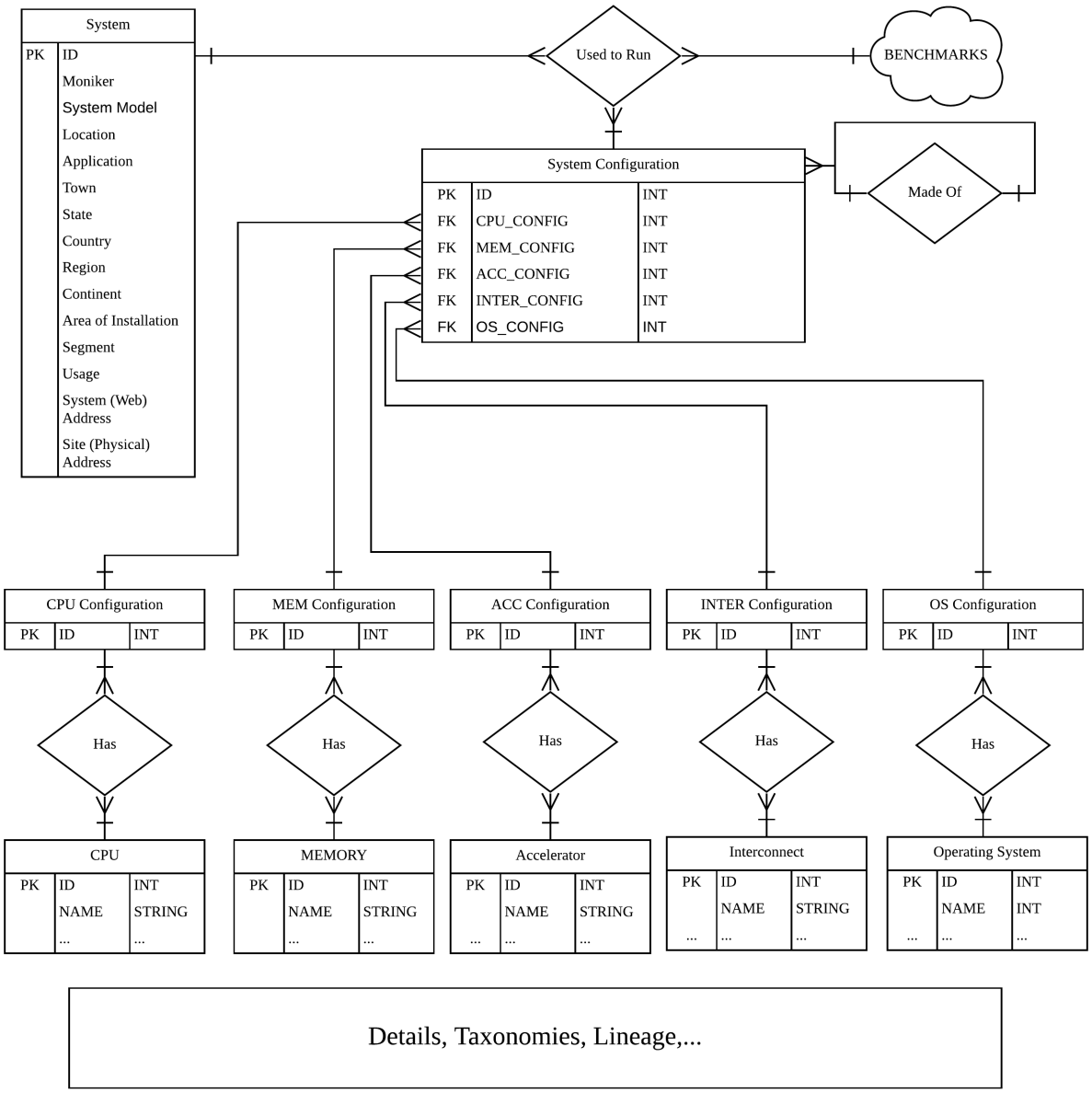

Figure 3.8: The final iteration of the iLORE data model. The ternary relationship between computer systems, system configurations, and benchmarks serves as the core of this design. Systems use one or more system configurations to run one or more benchmarks. We structure system configurations as a tuple of individual component configurations. We treat operating systems like major hardware components.

ing the source and type of the benchmark information. We use separate tables to store benchmark-specific information. To illustrate, we store LINPACK scores in a TOP500 table

and SPEC CPU scores in SPEC tables.

This schema is easily extensible in that new component information can be added without modifying the central structure. Because component information is loosely coupled with the rest of the schema, the sub-schemas used to represent individual components can be changed quickly in response to new data sources. Moreover, if system architecture changes in a way that necessitates the collection of alternative classes of components, we can easily extend the system configuration tuple.

3.3.3 Representing Processor Attribute Data

A representative overview of the processor schema is provided in figure 3.9. Our model for representing processor attribute data takes inspiration from the model used in the CPU DB [5], but provides additions to enable the integration of lineage and computer system data.

The "processor" table functions as the core of the processor schema. This table stores descriptive characteristics like "clock_speed," "model," and "tdp". We factor simple categorical attributes like "codename," "manufacturer," and "source" out into tables for reference. We augment the processor table with sources to aid future improvements and revisions.

Borrowing from the CPU DB, records in the processor table contain a link to an entry in the "technology" table. The technology table stores information like "feature_size" and "num_metal_layers" that describe fabrication characteristics. The processor table contains two links to the "cache" table. One link is for the processor's "on-chip cache" and the second is for the processor's "off-chip cache". Because cache configurations are often reused across processor models, isolating cache configurations in a separate table saves space and ensures that updates to a single cache record affect all relevant processors.

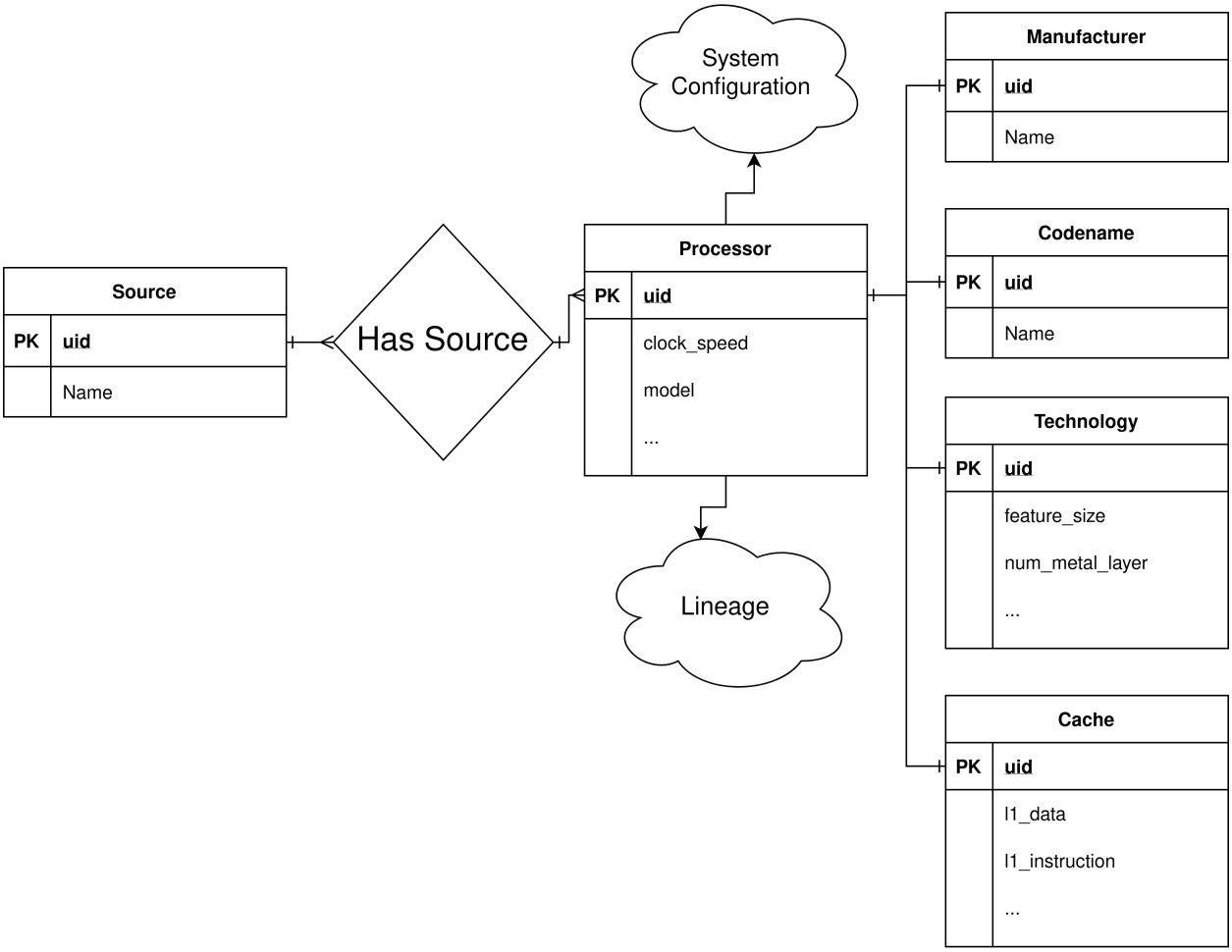

Figure 3.9: An overview of the processor schema. The foundation of this design comes from the Stanford CPU DB [5]. The system configuration schema is shown in detail in Figure 3.8. The lineage schema is shown in detail in Figure 3.11.

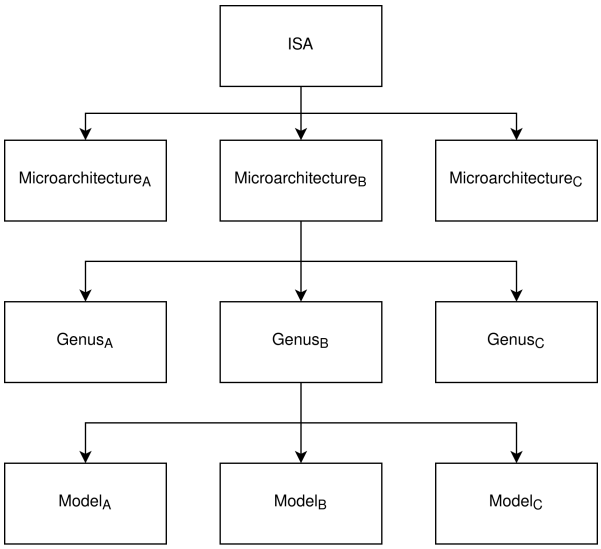

Figure 3.10: The structure of the microprocessor taxonomy.

3.3.4 Integrating Lineage and Taxonomy Data

Unlike in the case of processor attribute data, we are unable to lean on the CPU DB to design a schema that accurately models processor lineage and taxonomy data.

A visual representation of our taxonomy is provided in Figure 3.10. Layers of the taxonomy follow a simple one-to-many relationship in which the larger taxon may be linked to many members of the smaller taxon. A single processor genus will contain one or more processors, but each processor only resides within a single genus. The same applies with a genus being a member of only a single microarchitecture and a microarchitecture being a member of only a single ISA. We use an id placed in the smaller taxa to represent each of these relationships. For example, the processor table contains an identifier linking the processor to its respective genus.

An overview of the data model for lineage storage is given in Figure 3.11. As mentioned in Section 3.2.3, due to the rare presence of multiple inheritance, the lineage of microprocessors must be modeled as a directed acyclic graph (DAG) and not a tree. Parents and children

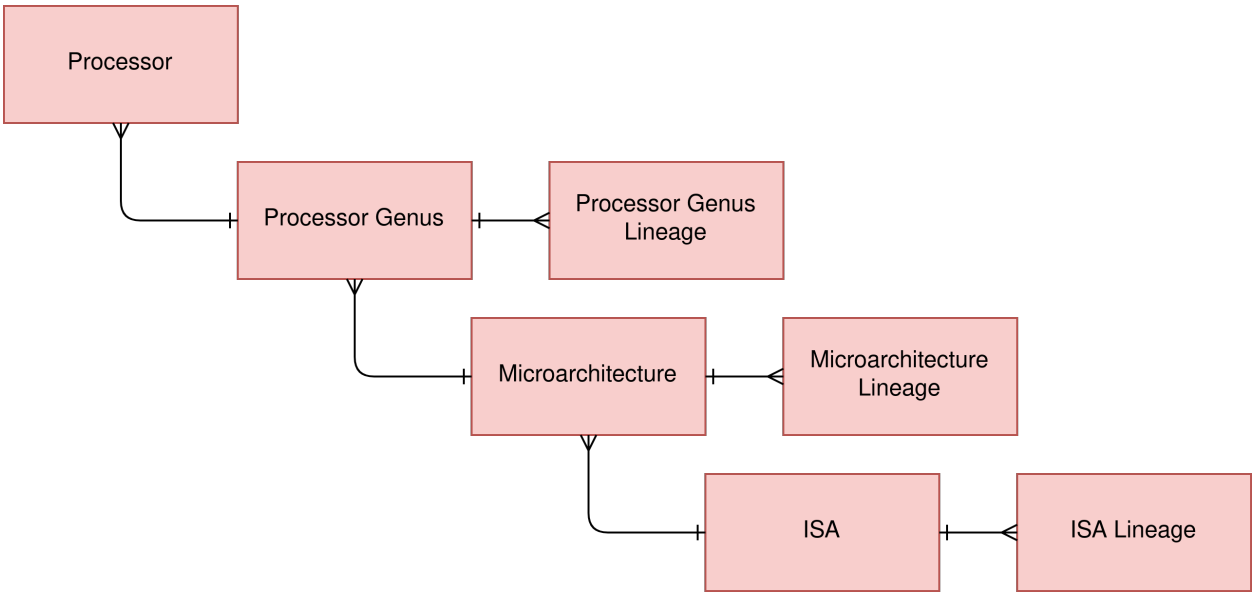

Figure 3.11: Storing the lineage in the iLORE data model.

follow a many-to-many relationship. Our data model contains a lineage table for each level of the taxonomy above the processor level. Each record contained in these lineage tables is comprised of two identifiers and a note. One identifier indicates the parent in the relationship and the other indicates the child. The note contains a human-readable description of the relationship found in that record. Similar to processor attribute data, we want to facilitate auditing of our lineage relationships. Therefore, we link each record of the lineage tables with the one or more sources that describe the given relationship.

3.3.5 Implementing the Schema

We implement our data model in the form of an SQLite schema described using SQLAlchemy [83]. The architecture of this implementation is provided in Figure 3.12. SQLAlchemy is an object-relational mapping (ORM) that allows us to describe the tables and relationships of our schema as Python classes. SQLAlchemy also facilitates querying and analyzing data in the database. Through SQLAlchemy we can process query results and update the database

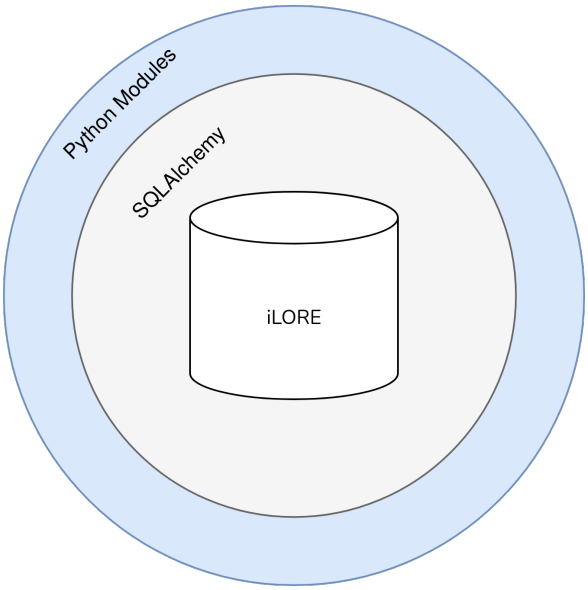

Figure 3.12: The architecture of our SQLAlchemy implementation of the schema.

through Python Objects. This pattern of interacting with the data is more flexible than raw SQL queries. Depending on how our code handles the Python objects returned by the ORM layer, we occasionally pay a performance penalty for this flexibility.

We implement our schema in two key Python modules. The first Python module contains definitions for temporary tables intended for use during the initial construction phase of the database. During the initial build, this module contained only definitions for tables to dump processor and computer system benchmark data. It now includes a memory dump table and will likely continue to expand as we incorporate more components into our database. The second Python module contains definitions for the permanent tables that make up our data model. This module has also expanded as we flesh out the data model for components like memory, accelerators, and operating systems.

3.4 Data Preparation

One of the fundamental goals of the CSGenome project is the combination of computer component, system, and system benchmark data. We must overcome several common data cleaning problems before combining these disparate data sets. All of our modifications can be classified using the system presented in Rahm and Do [84]. Data preparation occurs in parallel with the data collection and schema design efforts detailed in Sections 3.2 and 3.3. This section covers the cleaning, relationship inference, and database dump steps shown in Figure 3.1.

3.4.1 General Architecture

We use a pipe and filter architecture to implement our cleaning methods. Our architecture is comprised of three major phases: data collection, data preparation, and database population.

The data collection phase refers to either programmatic or manual collection of CSV, EXCEL, and HTML files onto a local storage device. We use the data preparation phase to make any modifications to the form or content of the data necessary to make said data amenable to insertion into our data model. The database population phase refers to the actual insertion of data from a CSV file format into the SQLite database.

3.4.2 Preliminary Data Examination

The first efforts made on the CSGenome project took the form of exploratory analysis conducted by a small group of undergraduates, including myself. Initially, we split up into pairs to identify and study potential sources of data. This discovery process yielded the sources of data that became the foundation of the current database. We also found several sources that

were either unsuitable for the CSG project like UserBenchmark [10] or not worth prioritizing like TPC [85]. During this time period, we also created several ad hoc cleaning scripts and made our first attempts at merging disparate sources of data.

From these experiences, we constructed our standardization guidelines. These guidelines took the form of a shared document that provided canonical aliases, allowed forms, known problems, etc. for each attribute. The standardization guidelines were an attempt at distilling the knowledge we gained through manual exploration into a structured and condensed form that would inform the policies implemented by our cleaning scripts. These guidelines would eventually be translated into a key component of our data cleaning pipeline.

This period of preliminary data examination yielded several valuable directives that we utilize in our final pipeline. First, we store data in a more rich format than flat CSV files. Our early experiences indicated that the payoff in the analysis justifies the cost of organizing the data into a structured format. Second, to facilitate rapid development and deployment, we prioritize the high-quality sources of benchmark data like TOP500 and SPEC. Third, we use a software architecture more sophisticated than ad hoc scripts to maximize code reuse and improve the extensibility of our efforts.

3.4.3 Single-Source Data Problems

Single-source data problems refer to errors in data isolated to an individual data source. In other words, these problems do not arise from the integration of multiple sources.

The first single-source data problem handled by the pipeline is the extraction of embedded values. Every SPEC CPU benchmark reports processor cache information as a string. To accurately link benchmark records with their corresponding processors later in the pipeline, we extract information from these cache strings and populate more granular fields for cache

sizes at different levels of the cache hierarchy. To illustrate, we transform the string "32 KB I + 32 KB D on chip per core" into the fields "l1_data" and "l1_instruction" both populated with integer value 32. Note that cache capacities come in several different formats. We use a collection of regular expressions to parse information from each of these formats. We use a similar technique to extract data from unstructured strings and populate fields like "num_cores" and "num_threads".

Characters that provide no valuable information or are unnecessary are scattered throughout the source data. Copyright and trademark characters, among others, are of little value in our efforts and often make other data operations more difficult. Therefore we remove these early in our cleaning pipeline. Redundant phrases like "CPU" or "Processor" in strings under the "processor" attribute are some other examples of disposable information that we remove early on. Other pieces of information like unit labels are valuable but should not be stored in every field. For instance, making sure to convert to uniform units as discussed in Section 3.4.4, we remove the strings "MHz" and "GHz" from processor frequencies. Removal of these strings allows us to treat many fields as integers or floats and reap the benefits of type checking throughout the remainder of our pipeline.

Date formatting is the third class of single-source data problems. The first date-related problem is the variety of forms in which dates are stored in the input data. We use a collection of regular expressions and the Python DateTime library to convert all dates to a single form. Early in the life of the CSGenome project, we stored dates in a custom float format. This format served us well for early analysis but became a detriment when storing our data in a relational database. Therefore, we switched to the more commonly supported YYYY-MM-DD format. Another problem is the varying levels of granularity at which dates are provided. Of the two main types of dates we handle, benchmark run dates and hardware release dates, hardware release dates are often given in much coarser terms.

Whereas benchmarks are dated to the day, hardware is typically only dated to the release month or quarter. In the case where only a month or quarter is given, we map that date to the earliest possible day in the given period. To illustrate, a processor released in July of 2017 and a processor released in quarter three of 2017 would both be given the date 2017-07-01.

A final class of trivial single-source data problems is the fixing of whitespace irregularities. Often as artifacts of other cleaning and extracting steps, additional whitespace characters get introduced between and on both ends of data entries. We strip this extraneous whitespace.

3.4.4 Multi-Source Data Problems

Multi-source data problems refer to errors that arise when attempting to combine disparate sources of data.

The first multi-source data problem we deal with is naming conflicts of certain attributes. To illustrate, SPEC CPU 2000 uses "CPU" to refer to the system's processor while TOP500 uses "Processor". Several other instances of this schema mismatch occur across sources. We handle this class of problems using a global attribute map that translates attribute labels into a single canonical form. For example, "NHalf," "nhalf," "n-half," "Nhalf," "n_half" and are all translated to the canonical "n_half". By convention, downstream from this aliasing step, all of our attribute names are lower case with words separated by underscores.

Even if two sources of data use the same label for an attribute, that attribute is often stored in different units. Our data set is especially prone to this problem because it spans multiple decades and many metrics in computer science grow in a superlinear fashion. Naturally, curators of the sources from which we pull will update their units to better match the values that get recorded over time. One instance of this is the shift from using Mhz to Ghz

to measure processor clock frequencies. For each instance of this problem, we establish a canonical unit. In the case of clock frequencies, we use the presence of the strings "MHz" and "GHz" to determine the input unit and convert everything to MHz. Failing the presence of such strings it is possible to employ a simple heuristic that assumes that single-digit clock speeds are in GHz and converts them to MHz. This heuristic is effective because single-digit clock speeds measured in MHz have not been commonplace since the 1970s and none of our benchmark sources date back that far.

Entity resolution issues constitute the final type of multi-source data problems. Broadly, an entity resolution problem refers to a situation in which two aliases refer to the same semantic object [86]. Especially because our data's destination is a relational database, we must recognize and canonicalize aliases among categorical variables. Some instances of this problem are not difficult to solve. To illustrate, among processor manufacturers both "Intel" and "intel" are used to refer to the canonical manufacturer Intel. While the case of manufacturers is easy to solve by hand, the complexity of the problem escalates both with the complexity of information stored and with the number of possible options.

3.4.5 Connecting Processors and System Configurations

Integrating microprocessor component data and system benchmark data is both one of the key goals of the CSGenome project and a difficult entity resolution problem. This matching problem is complicated from several perspectives. First, the quantity of unique values is large. A benchmark record can be matched up to any one of several thousand processor records. Second, the data has multiple dimensions. A processor is described by a collection of attributes instead of a single attribute.

The two sides of this matching problem are a benchmark record which contains an incomplete

set of processor information including at a minimum a processor model and clock speed and a more complete processor record. The goal is to accurately identify the full processor record that matches with each incomplete benchmark record. The problem is further compounded by the presence of conflicting information between attributes on the benchmark and processor side.

One reason for conflicting information in valid matches is the aliasing of processor models. Aliasing is particularly common among the DEC Alpha chips. Some benchmark records referring to these processors provide full models like 21264A. Other records use only codenames like EV67. Making matches of this variety requires some domain expertise because the strings that describe the full model and codename are nothing alike.

Another reason for conflicting information is a difference between a processor's capabilities and what is utilized for a particular benchmark record. One example of this is the SPEC CPU int benchmarks. These benchmarks do not use more than a single core and often disable simultaneous multithreading. Therefore there are many SPEC CPU records where the reported number of cores does not match with the available number of cores. Another case of this is the reporting of clock speeds with a different amount of significant figures. There are some cases where benchmark records report more significant figures than the processor record derived from a manufacturer specification (e.g. 2667 MHz vs 2666.7 MHz).

A final reason for conflicting information is human error. While curators of both the TOP500 list and SPEC CPU benchmarks work hard to ensure the validity of their data, the benchmark records are ultimately submitted by individuals that sometimes make data entry mistakes. There are a relatively small number of cases where attributes like clock speed are blatantly wrong on the benchmark side.

Not taking into account cases of conflicting information, finding a match between benchmarks

and systems can still be difficult due to the ambiguous naming of processor models. This problem is particularly rampant among manufacturers like Intel and AMD who release many processors and have long-running processor lines like Pentium, Xeon, and Opteron. Early processors in these lines often lack unique model numbers or have no model numbers at all. It is often difficult or impossible to differentiate among these ambiguous processors based on the information available from the benchmark records.

Taking these challenges into account, we first attempted to develop a programmatic solution that would accurately map processors to benchmark records. This tool was called the "fuzzy matcher" or the "autocompleter". The autocompleter tool used a heuristic based on the tokens that made up a processor's model.

In practice, the autocompleter failed to produce satisfactory results due to the aforementioned challenges. The autocompleter was only truly effective at making matches against processors that have unique model numbers like Intel Xeon E5-2680. However, records of this type do comprise the majority of our data, and the autocompleter was useful for getting an approximate match. However, its lackluster accuracy was not enough to justify the cost of maintaining a tree of tokens and we decided not to rely on the autocompleter to completely solve this problem.

After understanding the shortcomings of the autocompleter, we moved to utilize the dedupe library [87]. Dedupe is a Python library that uses machine learning to perform entity resolution. Dedupe takes as input two sets of data, in this case, attributes on the benchmark side and attributes on the processor side, and repeatedly presents pairs of records from the two sets of data to a user on a command-line interface. For each pair, the user decides if the records are a match, not a match, or cannot be determined either way. Through this iterative process, dedupe discovers which attributes are most important for determining a match. After sufficient training, the output of dedupe can be used to match the entirety of

the two data sets. Unfortunately, we were unable to get dedupe to accurately make matches and abandoned the tool.

Fortunately, after analyzing data from the four SPEC CPU suites, TOP500, and Green500, we found that roughly 1500 unique processor tuples covered 95% of all benchmark records. In light of the above challenges and the manageable size of the data set, we use the autocompleter to make the best guess at matching unique processor tuples to a processor record and manually review and replace or supplement autocompleter's guess manually. Benchmarks that fall outside of this majority of tuples are matched with what we call a processor "stub". These stub records contain processor information extracted from the benchmarks in question but are marked as incomplete. Stub records are either duplicates of existing canonical records or new processors for which we do not have comprehensive information. Future efforts should be conducted to eliminate or reduce these stub records.

3.4.6 Populating the Database

After preparing the component and benchmark data and establishing our data model, we populate the database. To ensure the presence of the processor records to which benchmark data were manually linked, we build the processor tables before building the systems and benchmark tables. Tables in the database are populated via a bootstrapping process we refer to as the "dumptable" method.

The population of the processor tables begins by dumping the flat processor CSV file counting processor attribute, processor taxonomy, and processor lineage data to the temporary processor dump table. The contents of lookup tables like source, manufacturer, codename, and cache are extracted from the dump table using SQL queries issued through SQLAlchemy. Higher-level tables like processor, technology, and microarchitecture are constructed by

querying against both the processor dump table to get raw information and the previously populated lookup tables to translate raw information into corresponding identifiers. By this same procedure, higher levels of the processor taxonomy are populated. To complete the population of the processor tables, we query the dump table and corresponding levels of the taxonomy to populate the taxonomy and taxonomy source tables.

Filling the database with system and benchmark data also starts by moving a flat CSV file into a dump table. From the systems dump table, we use SQLAlchemy to normalize categorical variables like country, application, and interconnect, to their look-up tables. Next, because they have few dependencies, we extract information for individual benchmarks. Common data like information sources are aggregated in a join benchmark table that links to individual benchmarks. Third, we construct the system table using both the system dump and look-up tables. Finally, we link together each benchmark with a system and system or hardware configuration.

3.4.7 Exposing Data through an API

Other repositories of this data demand an understanding of their data model to conduct meaningful analysis. Even the CPU DB [5] only provides raw CSV files that require parsing. To provide our data to the largest number of users, we expose the contents of the iLORE database to the public through a RESTful API. Researchers may use HTTP requests to receive and process JSON responses that can be processed across a wide variety of programming models and statistical tools. This API layer serves as a single contract that defines interactions between the iLORE database and collaborative research efforts.

To support the iLORE API, we employ several layers on top of the database. The majority of our API architecture is constructed by Nicolas Hardy, another graduate student supporting

the CSGenome project.

As previously mentioned, we use SQLAlchemy to implement our schema. SQLAlchemy is useful here for its ability to return the results of SQL queries as collections of Python objects. Python objects are a flexible and often preferred form of our data for local analysis, however, they are not suitable for transmission. Therefore, after extracting data from the database, we use marshmallow [88] to marshal our objects into the JSON format that the end users' applications receive. The outermost layer is the interface that users interact with. We use the Flask web framework [89] to define the endpoints to which users send their requests.

3.4.8 Supplementary Data Preparation

In the course of reviewing our work and conducting analysis, we find it necessary to revise our data after the initial database build process. After weighing our options, we perform these modifications in place instead of cleaning the input files and performing another bootstrapped build. Although in-place modifications are more performant than the costly full build, this decision comes with several trade-offs. First, in-place modification removes the ability to go from raw data to a fully updated database. In other words, in place modification introduces the possibility of losing work. Second, because changes occur in a small, incremental fashion instead of large, sweeping builds, it is easy to lose track of which changes have been made. To mitigate this effect, we have added fields to each of the database tables to track the author of a record, the date of the record's creation, and the date of the record's most recent modification.

Abandoning the full rebuild process or "locking" the database allows us to become more confident in both the structure of our schema and the quality of our data. This locking process is an early step towards analysis and future data improvements.

These post-build updates fall into two classes: data migrations and schema migrations. Data migrations are changes that require modifications to the contents of the database but leave its form untouched. Schema migrations instead require modifications to the actual data model of the database. Schema migrations often require an auxiliary data modification to be valid.

To illustrate, consider adding a set of unique constraints to a table. Before applying the unique constraint, we would have to perform data migration to ensure that none of the existing records violate the new constraint. We perform data migrations through ad hoc scripts that leverage SQLAlchemy. We perform schema migrations using a tool called Alembic [90] that maintains a history of schema migrations that enables upgrading and downgrading the schema of a given database to an arbitrary point in time.

Because we are using SQLite which is file-based, we regularly back up our database files to a shared server. These files are each tagged with a timestamp and a git commit hash. If a revision is found to introduce errors, we can return to the database before that misstep and lose minimal progress. Because multiple developers are working simultaneously to update the iLORE database, special care is taken to make sure that we serialize both data and schema migrations.

All future revisions and extensions to the data are planned to occur through the API described in Section 3.4.7 after the completion of the POST and PUT endpoints. The API will provide a single interface through which we perform data migrations. Both manual modifications and ad hoc scripts would be conducted at the API layer instead of the SQLAlchemy Layer.

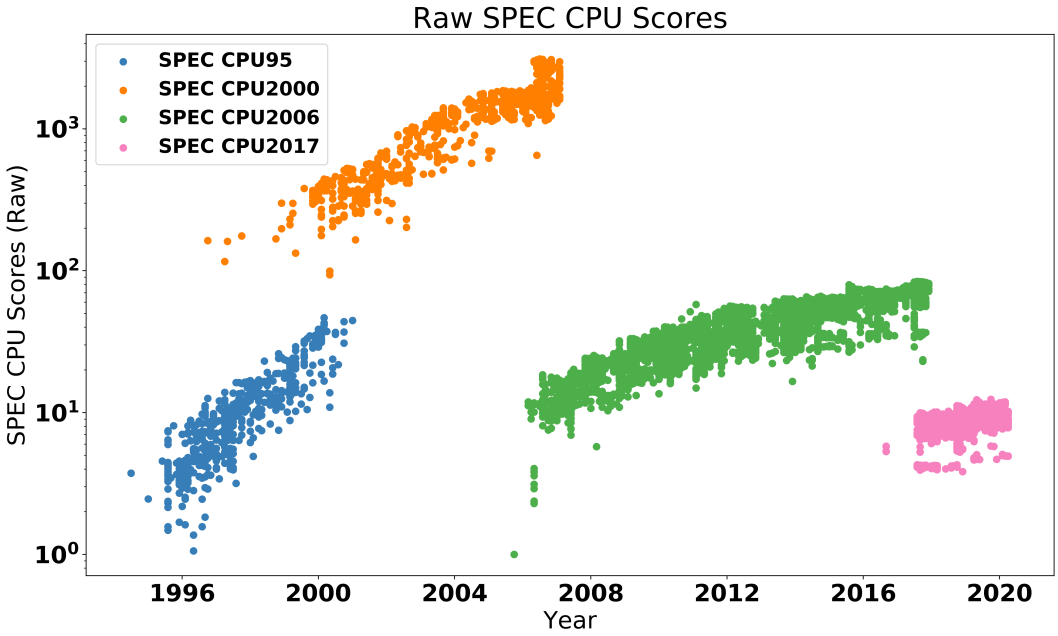

Figure 3.13: Raw SPEC CPU single-core integer benchmarks

3.5 Creating a Comparable Performance Metric

Our primary performance metric is SPEC CPU benchmark scores. However, because SPEC changes their benchmark suites periodically as described Section 2.5 raw scores are not comparable across different suite iterations.

Figure 3.13 presents unaltered single-core integer speed scores across SPEC CPU95, CPU2000, CPU2006, and CPU2017. Each suite is contained within its colored cluster. Because we know that computational power has increased significantly over time, Figure 3.13 is a clear demonstration that comparisons of raw scores across SPEC suites are not valuable. The SPEC organization recognizes the value of historical comparisons and suggests using systems benchmarked across multiple suites as bridges [91]. However, SPEC indicates that such conversions are approximate at best.

SPEC CPU Conversion Factors Summary Statistics				
Factor	Size of Overlap Set	Unique Processors	R2	Conversion Factor
$\text{factor}_{2006-2017}$	31858	71	0.825	0.118
$\text{factor}_{2000-2006}$	3527	65	0.695	0.006
$\text{factor}_{1995-2000}$	199	28	0.763	10.822

Table 3.1: Summary statistics for SPEC CPU conversion factors.

Conversion from one SPEC suite to another is accomplished by applying a conversion factor to each SPEC suite. This is a common methodology employed in previous studies [42, 92]. Robert Munafo calculates his conversion factor by comparing the performance of a single reference machine across two SPEC suites. Danowitz et al. instead find an overlapping set of benchmark records across two SPEC suites that use the same processor. The CPU DB's conversion factor is the geometric mean of all the performance ratios in this overlapping set.

We repeat the methodology of Danowitz et al. here. We calculate a conversion factor for each pair of consecutive SPEC suites using the base integer speed score. To convert between non-adjacent SPEC suites, we multiply by multiple conversion factors.

$$\text{score}_{2017} = \text{score}_{2006} \times \text{factor}_{2006-2017}$$

$$\text{score}_{2017} = \text{score}_{2000} \times \text{factor}_{2000-2006} \times \text{factor}_{2006-2017}$$

$$\text{score}_{2017} = \text{score}_{1995} \times \text{factor}_{1995-2000} \times \text{factor}_{2000-2006} \times \text{factor}_{2006-2017}$$

Danowitz et al. indicate that more accurate alternatives to this single conversion factor method may exist. Because their work was released in 2014, Danowitz et al. only provide conversion factors up to SPEC CPU2006. Therefore, we validate our work using shuffled, iterated k-fold cross-validation with five iterations and five folds per iteration. The results of this verification, found in Table 3.1, corroborate the findings of Danowitz et al. The

	Conversion Factor R2	Regression R2
$\text{factor}_{2006-2017}$	0.788394	0.795925
$\text{factor}_{2000-2006}$	0.787513	0.815651
$\text{factor}_{1995-2000}$	0.788394	0.842915

Table 3.2: Comparing conversion factor and regression methods for SPEC score conversion.

single conversion factor method produces reasonable predictions but offers some room for improvement.

Because the authors of the Stanford CPU DB indicate that more promising methods may exist and we were concerned that the small number of unique processors in the overlap set were not representative of all SPEC CPU results, Yueyao Wang, another student supporting the CSGenome project, developed more sophisticated regression models based on processor characteristics. These models are found in Equation (3.5).

$$\log(\frac{\text{score}_{2000}}{\text{score}_{1995}}) = \beta_0 + \beta_1 \log(\text{score}_{1995})$$
$$\log(\frac{\text{score}_{2006}}{\text{score}_{2000}}) = \beta_0 + \beta_1 \log(\text{score}_{2000}) + \beta_2 \text{threads\_per\_core}$$
$$\log(\frac{\text{score}_{2017}}{\text{score}_{2006}}) = \beta_0 + \beta_1 \log(\text{score}_{2006}) + \beta_2 \text{threads\_per\_core} + \beta_3 \text{number\_cores}$$

In collaboration with Wang, we evaluate our single conversion factor method against the regression models using k-fold cross validation with 5 folds and find that the regression models provide reasonable but minor improvements to accuracy. Results are shown in Table 3.2. Because we can achieve relatively high accuracy using a simple method, we use the single conversion factor technique throughout the remainder of our analyses.

We use data pulled through the API to construct the regression models and conversion factors. Abbreviated results used for analysis can be found in Figure 3.14. Rapid development

```
{
  "data": [
    {
      "clock_speed": 250,
      "l1_data": 32,
      "l1_instruction": 32,
      "l1_shared": null,
      "l2_shared": 1024,
      "l3_shared": null,
      "max_clock_speed": null,
      "new_bmark_id": 7636,
      "new_information_source": "https://www.spec.org/cpu2000/results/res2000q3/cpu2000-20000612-00124.html",
      "new_score": 99.4,
      "number_cores": 1,
      "old_bmark_id": 1206,
      "old_information_source": "https://www.spec.org/cpu95/results/res2000q2/cpu95-20000508-04233.html",
      "old_score": 10.9,
      "proc_id": 1010,
      "tdp": null,
      "threads_per_core": 1
    },
    ...
  ]
```


Figure 3.14: Abbreviated results returned as a response to a GET request on our SPEC overlap endpoint located at https://ilore.cs.vt.edu/devapi/benchmarks/specoverlap/1995,2000?operation_type=integer&score_type=speed&compiler_type=base

of such models would be impossible using existing data sets and is an example of the utility of the iLORE database and accompanying API. We hope that users of the API will be able to conduct similar analyses moving forward.

Unfortunately, we only have SPEC CPU scores dating back to 1995. Microprocessors released before 1995 are not included in performance analyses but are present when looking strictly at other characteristics. The listed conversion factors are only applicable to the base integer speed SPEC CPU scores. Each type of SPEC CPU score requires a separate calculation of a conversion factor. We calculate the conversion factors for each of the eight kinds of SPEC CPU scores using the conversion factor methodology. Relative to the converted base integer speed scores, the other types of converted scores are far less accurate. Danowitz et al. [42] only utilize the base integer speed scores in their performance analysis. Although they do not explicitly justify this selection, we hypothesize that the conversion method is particularly well suited to base integer speed scores. Therefore, we also use base integer speed as our performance metric.

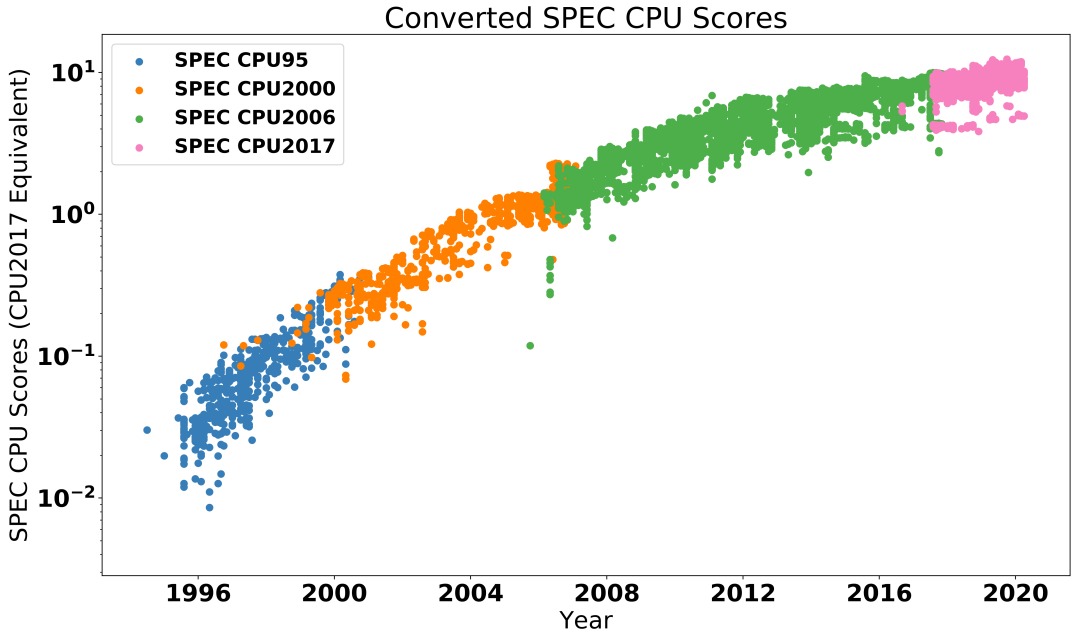

Figure 3.15: Adjusted single-core integer speed scores across 4 SPEC CPU suites

In Figure 3.15, we display the results of applying the conversion factors to each of the SPEC suites. The once disparate points are now connected in a single, sub-exponential trend. The details of this trend will be explored in more detail in Section 4.1.

3.5.1 Summary of Database Contents

Before we dive into analysis, it is important to understand the contents of the database at a high level. The analysis found in this thesis pulls from the computer component and benchmark tables of the iLORE database. A list of attributes relevant to analysis can be seen in Table 3.3. Each benchmark record is linked with a processor record using the structure described in Section 3.3.

Please refer to Table 3.4 to get an idea of the scope of the iLORE database. The number of SPEC Benchmarks is 2.05x the number of TOP500 benchmarks because the TOP500

Processor Attributes		**Benchmark Attributes**
Information Source	Transistor Count	Information Source
Release Date	Manufacturer	Benchmark Date
Clock Speed	Processor Genus	Benchmark Score
Max Clock Speed	Die Size	
Number of Cores	Cache Sizes	
Threads Per Core	Model Number	
TDP	Process Size	

Table 3.3: Important attributes for analysis.

Entity	Count
SPEC Benchmarks	76264
TOP500 Benchmarks	36995
Processors	4454
Processors (non-stub)	3692
Processor Genera	779
Microarchitectures	287
ISAs	79

Table 3.4: Counts for records in key tables.

list is limited to 1000 entrants per year. Moreover, we present a count for both stub and non-stub processor records. As discussed in Section 3.4.5, stub records are incomplete and are not linked into the lineage or considered in the analyses presented in section 4. Finally, the number of ISAs may seem inflated at first glance. Different versions of an ISA such as PA-RISC 1.0 and PA-RISC 2.0 are treated as unique records.

3.6 Limitations

In this section, we discuss the current state and limitations of the iLORE database.

3.6.1 Data Collection and Preparation

Duplicate Processors and Systems Our approaches to cleaning both processor and system data leave the possibility for duplicates in the iLORE database. Duplicate system records are the result of our heuristic approach for checking the equality of two systems. We determine the equality of system records by comparing a relevant subset of system attributes. Some members of this relevant subset may not be properly canonicalized, leading to multiple semantically identical systems in the final database. Duplicate processor records may exist for a similar reason and are the result of the relationship inference process. Failure to make a match between a system and a processor results in the creation of an inferred or stub processor record. Due to manual error, it is possible that this inferred record is one that already exists in the database. To combat these duplicates, we make a manual pass of the processor records and remove any duplicates found. No such effort has been made for system records at this point.

These effects have the potential to inflate the number of systems and processors stored in the iLORE database. Analyses that investigate individual systems changing their hardware configurations are especially sensitive to these issues.

Missing Processors A substantial minority of our stub processor records have no existing canonical record in the database. These records include proprietary and less common processors such as IBM's POWER9. In contrast to Intel and AMD, manufacturers of these rarer processors do not maintain large, open repositories of information. Information on these processors cannot be gathered using our usual scraping methods. Therefore, we should lean on crowd sourcing or manual efforts to find these missing processors.

Linking TOP500 and SPEC Integration of disparate data sets is one of the iLORE database's fundamental goals. In its current state, iLORE does not support the integration of TOP500 and SPEC benchmarks. Unfortunately, there are no system configurations used across the two data sets. This is not surprising considering TOP500 is a list of the world's fastest supercomputers and SPEC tends to target smaller machines. We are in the process of integrating these data sets by examining processors used in both TOP500 and SPEC, but several hurdles must be overcome before meaningful analysis is possible. The first hurdle is to identify a SPEC microbenchmark that approximates the LINPACK [38] benchmark used by TOP500. The second hurdle is to develop a methodology for normalizing the size of the machines used across the two benchmarks. As this research matures, we hope to improve the integration of our data.

3.6.2 Sustainability of the Data Pipeline

Updating the Database Publicly available data sources of benchmark and component data are constantly being updated with new and valuable information. TOP500 and Green500 [3, 4] release lists twice a year. SPEC releases results from their latest benchmark suite CPU2017 [41] at least once per quarter. AMD and Intel [69, 70] update their product specifications as new processors get released. It is evident that iLORE will require regular updates to maintain its utility.

Currently, only processors and memory data are integrated with benchmark and system data. Further work is underway to collect and integrate other components including accelerators and operating systems. Therefore, it is key that we have procedures to add entirely new sets of information to the database.

Current State of the Data Pipeline The first part of the data pipeline is scraping. Current scraping modules should continue to be effective with only minor updates to account for outliers. Therefore, we should have no problem continuing to collect data from existing sources. However, new sources will require their own scraping modules.

After scraping, data is cleaned using a collection of filters. With some modification to their form, these filters can be used to clean new data sets. New data sets will likely come with their own set of challenges that require the creation of new filters. Moreover, manually conducted portions of the initial cleaning process will need to be repeated on newly ingested data. For example, the manual matching of system and processor records will need to be replicated with any updated data sets.

Modules used to populate the database are not reusable at present because manual modifications described in Section 3.4.8 have been made to the database after its initial dump. New population methods should modify the database in place.

Chapter 4

Analysis and Results

We substantially modify existing data sets to construct the iLORE database. Therefore, to gain confidence in our methods, we mine our database for previously published, peer reviewed, and well-understood trends. After gaining confidence in our data, we move to demonstrate the utility of the iLORE database by answering analysis questions that leverage the integration of microprocessor and benchmark data. We conduct this analysis using data queried from either the database using SQLAlchemy or the API.

4.1 Validating the Data

Here we pull from past works and well-understood scaling trends to verify the validity of our data. We retrieve the data for these analyses using SQLAlchemy queries. These analyses are used to identify and patch issues in the iLORE database.

4.1.1 Scaling Trends

Karl Rupp is a computer scientist whose research interests include multi-core and many-core architectures. He has authored a series of blog posts exploring the evolution of fundamental microprocessor trends over the past six years [93, 94, 95]. The first blog post [93] extends data originally collected by Horowitz and others. Rupp augments this original data set with

data from AMD Opteron, Intel Xeon, Power7+, and Power8 processors. Rupp also includes the many-core Intel Xeon Phi processors in his plots. In his subsequent blog posts, Rupp continues to collect more data and upgrade these plots.

Rupp's most recent blog post [95] discusses a single figure which features data up to 2020. The plot tracks five quantities: transistor counts, single-thread integer performance, frequency, power, and the number of cores. The quantities are plotted on a log scale and the performance scores are multiplied by a constant factor to make the graph more readable. Because SPEC CPU scores are only available after 1988, processors plotted before this point lack a performance score.

Transistor counts are the first quantity tracked by Rupp's figure. Transistor counts are a particularly popular microprocessor metric because of Moore's law [96]. Moore's law describes the empirical observation that the number of transistors placed on a dense integrated circuit (IC) double every N months. In his original 1965 publication [96], Moore forecasted that this doubling would occur every year or $N = 12$. The law has since been revised towards a higher N of 18 to 24 months. By Rupp's plot, it is clear that the exponential growth in transistor counts, regardless of the rate, has continued until 2020.

Single-threaded performance is the next metric displayed by Rupp's figure. Here, there is a clear exponential trend until around 2010. In the last ten years, there has been a noticeable slowdown in improvements to single-threaded performance. This trend is directly tied to the leveling off of processor clock speeds around the same time frame. Instructions per cycle (IPC) must be improved to achieve greater single-thread performance under equivalent clock speeds. Maintaining clock speeds and improving IPC has proved a difficult challenge in the last decade leading to this slowing.

As mentioned previously, the third quantity, clock speeds, have stagnated since around 2005.

For many decades following the introduction of the microprocessor, microarchitects relied on Dennard scaling [37] to keep the power density of their chips under control. The Dennard scaling law states that the power density of transistors remains constant as they shrink. Bohr recognized the breakdown in Dennard scaling in 2007 [97]. At this point, it was no longer feasible for manufacturers to meet power budgets while maintaining exponential increases in clock speeds. As a result, clock speeds have hovered between three and four gigahertz for the past decade.

Similar to the past two trends, power grew roughly exponentially until it leveled off. Microprocessor power stopped increasing a little bit earlier than frequency and single-thread performance around 2000. Many works around this time [32, 33, 98] condemn the once common practice of trading power for performance as unsustainable in both mobile and server applications. Figure 6 from Ronen et al. [32] indicates that the processors of this time were exponentially increasing in power density despite already exceeding the power density of a typical hotplate. Microarchitects have been dealing with this "power wall" ever since.

To achieve performance gains despite the breakdowns in clock speeds and single-thread performance, microprocessor designers have been investing their transistor budgets in increasing the number of cores on a single die. Many references advocate for this approach of improving performance [26, 99]. Rupp's figure demonstrates an exponential increase in the number of logical cores starting around 2005.

We recreate Rupp's 48 years of microprocessor trend data using the iLORE database in Figure 4.1. We fetched the same five quantities and plotted them on a log scale against time. Similar to Rupp's figure, Figure 4.1 does not display a performance metric for early data points. Our performance measurements start in the early 1990s as opposed to the late 1980s of Rupp's plot because our earliest benchmark is the SPEC CPU95 suite instead of the CPU89 suite. All processors plotted after 1995 use a SPEC CPU2017 equivalent score

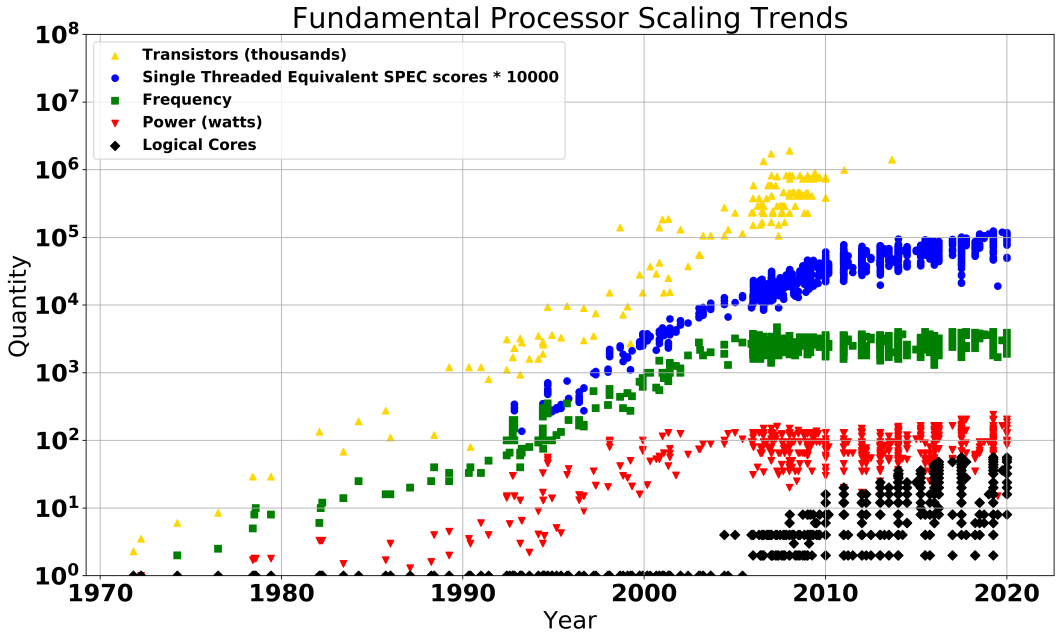

Figure 4.1: Using the iLORE database to recreate the fundamental trends curves shown in Rupp's figure.

calculated using the methods outlined in Section 3.5.

Using our data, we independently observe an exponential increase in transistor counts until 2014 after which we lack data. Unfortunately, for unknown reasons, both AMD and Intel opt to not publish transistor counts for their most recent products [69, 70].

Our methods for creating a comparable single-thread performance score line up favorably with Rupp's plot. Due to differences in normalization techniques, Rupp applies a constant multiplier of 1000 and we apply a constant multiplier of 10000 to achieve roughly the same curve.

Clock speeds in both plots follow the same fundamental trend of exponential growth until 2005 followed by stagnation up to the present.

The variance in power measurements displayed in Figure 4.1 is noticeably higher than that

found in Rupp's figure. This effect is the result of our inclusion of mobile processors intended for use in devices like laptops and tablets in the database. Rupp's data set contains only flagship processors that tend to always max out their power budget. Despite these differences, the top of the power trend found in Figure 4.1 approximates the trend shown in Rupp's figure.

Similar to power, Figure 4.1 showcases a greater variance in logical core counts than Rupp's figure. This variance in logical core counts can once again be attributed to Rupp's use of only flagship processors that tend to include the max core counts of their time. The curves are similar when comparing the top of the logical core trend found in Figure 4.1 to the trend in Rupp's figure.

To gain more confidence in the quality of our data, we also compare against analyses conducted by Danowitz et al. [42]. Successfully recreating analyses conducted by Danowitz et al. using the original CPU DB indicates that we accurately integrate the CPU DB.

Figure 4.2 is generated from the iLORE database in an attempt to recreate Figure 7 from Danowitz et al. [42]. Both figures plot processor frequency against processor release dates. From 1985 to 2011, the plots are nearly identical. We use the CPU DB to fill any holes in our data, the iLORE database preserves many of the eccentricities found in the CPU DB. Figure 7 from Danowitz et al. [42] and Figure 4.2 both display several vertical stacks of DEC points in the mid 1990s. Similarly, both graphs contain a vertical cluster of particularly fast IBM processors around 2007.

Figure 4.2 provides data well after Figure 7 from Danowitz et al. [42] stops. This tail end of the curve maintains the stagnating pattern we begin to see in both figures in the late 2000s. This period reveals the gaps in our data for AMD processors. Our source for modern AMD processors [69] only provides information on products released from 2016 to present day. Therefore we see a gap between the end of data from the CPU DB and the start of

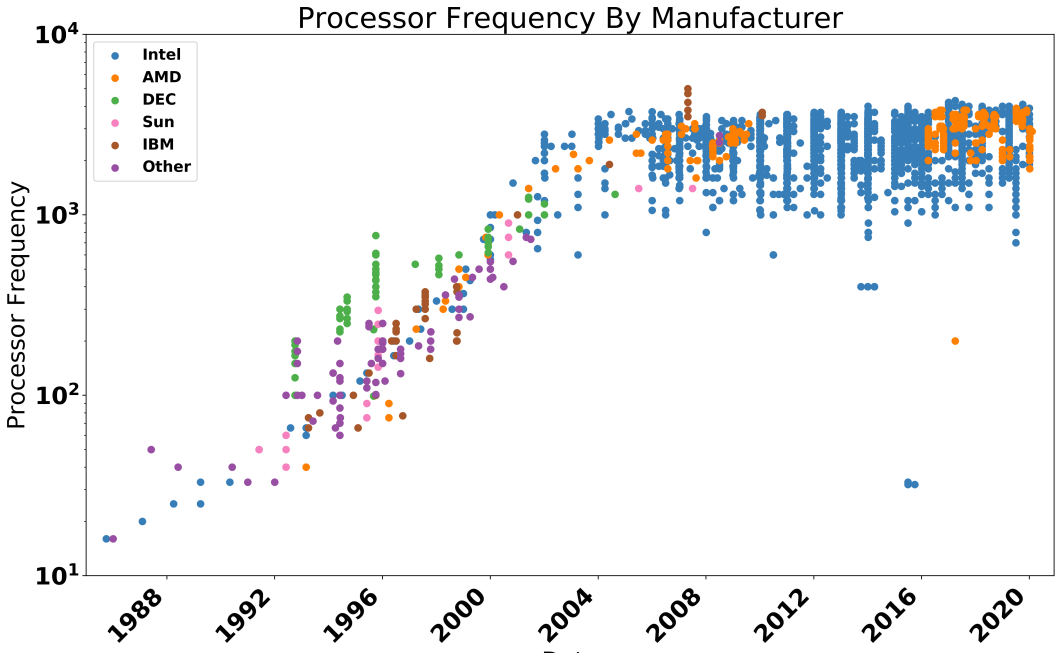

Figure 4.2: Processor frequency over time generated from the iLORE database.

data from the AMD product specifications.

Figure 4.2 contains a cluster of outliers with surprisingly low clock speeds in 2016. Although they constitute a small minority of our data, we include some embedded processors in the iLORE database. Low clock speed outliers correspond to these microcontroller records [100].

Because the differences between the Rupp's figure and Figure 7 from Danowitz et al. [42] and our recreation Figures 4.1 and 4.2, are either minimal or have a defensible explanation, we believe the iLORE database can support exploratory analysis of microprocessor and SPEC performance data.

4.1.2 TOP500 Performance Over Time

To verify that the database model is accurate for TOP500 data, we lean on the work of Nicolas Hardy, another graduate student supporting the CSGenome project. Hardy generated Figure 4.3 using the iLORE API in an attempt to recreate Figure 2 from TOP500's performance development statistics [101]. [1]

Both graphs plot LINPACK performance against the publication date of the TOP500 lists. The blue and yellow points track the performance of the weakest and strongest system in each list respectively. The green points track the sum of the performance of all systems within a given list. Both graphs include a line of best fit. We generate lines of best fit using linear regression on the log of the data and achieve R^2 values of 0.9930, 0.9878, and 0.9896 for the sum of performance, strongest performance, and weakest performance respectively. The reference plot provides no methodology for its lines of fit and no R^2 values.

Figure 2 from TOP500's performance development statistics [101] and Figure 4.3 are nearly identical. Each of the three series follow a clear linear trend on the logarithmic axis that tapers off slightly in more recent years. Individual fluctuations in the data are preserved like the "stair steps" in the yellow points.

Because the visual differences between the reference Figure 2 from TOP500's performance development statistics [101] and our recreation Figure 4.3 are minimal, we conclude that the TOP500 performance data stored in iLORE is accurate..

[1] Hardy provides more details on Figure 4.3 and Figure 2 from TOP500's performance development statistics [101] in his MS thesis, iLORE: A Data Schema for Aggregating Disparate Sources of Computer System and Benchmark Information (Hardy, 2021)

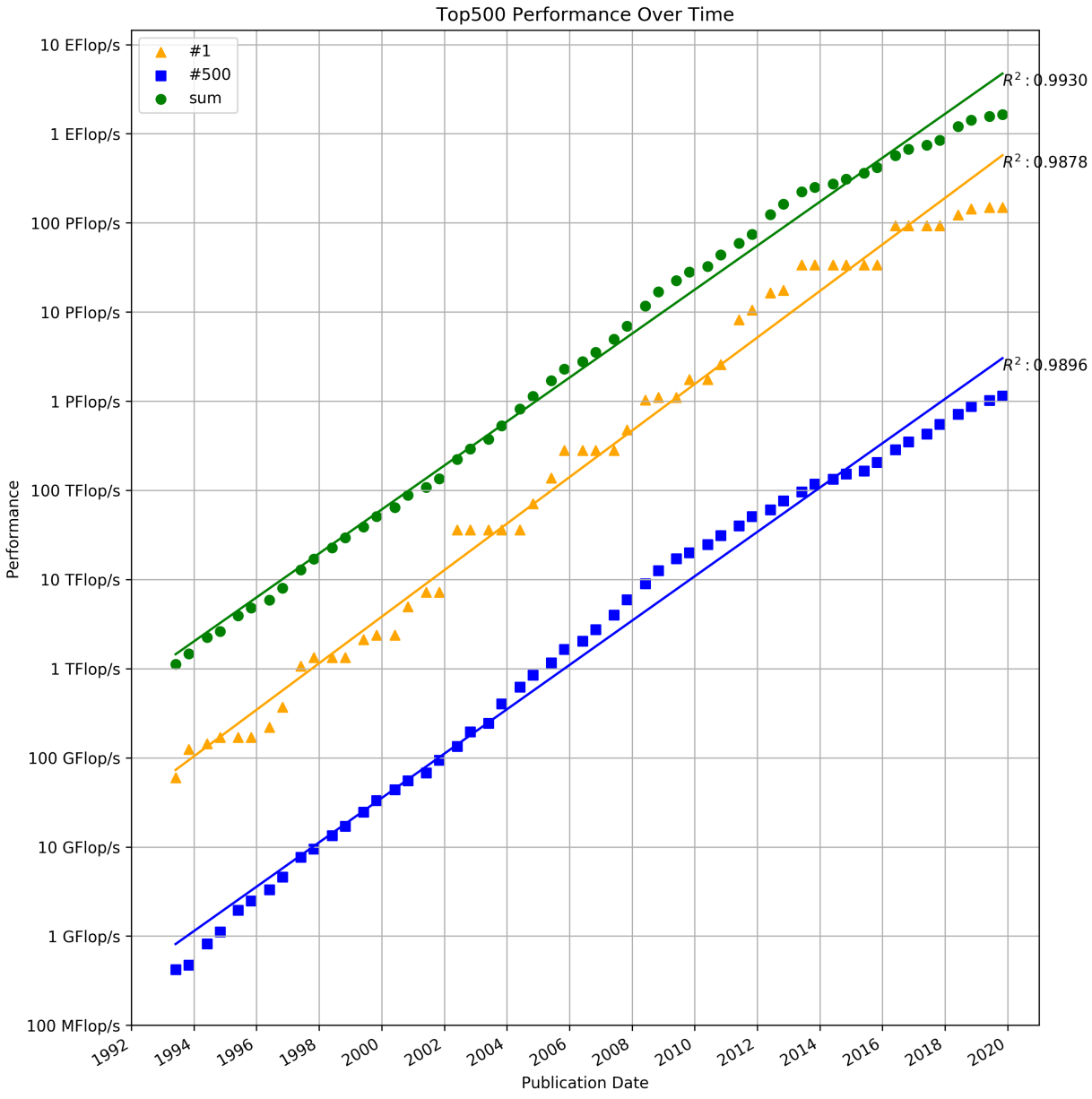

Figure 4.3: TOP500 performance over time generated from the iLORE database.

4.2 Exploratory Analysis

Here we demonstrate the value of the iLORE database in its current state by exploring unanswered analysis questions. We retrieve the data for these analyses directly from the iLORE database using SQLAlchemy queries.

4.2.1 SPEC Market Share

TOP500 frequently publishes visualizations and charts that summarize the historical trends in TOP500 lists [102]. These visualizations track both system configurations and performance characteristics of the world's fastest supercomputers. Of particular interest are visualizations displaying the vendor market share broken down by both the number of systems and the share of total performance. These visualizations provide members of the HPC community with a high-level understanding of the large players in the supercomputing market.

Figure 1 from the TOP500 statistics over time page [103] displays the market share of the TOP500 from the first list in 1993 up to June 2020. Large, long-standing corporations like Hewlett Packard Enterprise (HPE), Cray/HPE, and IBM make up a large share of every TOP500 list. However, there is a fair amount of heterogeneity across the entire lifespan of the TOP500 with at least nine vendors participating in every list. The "Others" category has grown in recent years indicating that variety may even be on the rise.

Because the SPEC CPU suites target machines that are much smaller than what appears on the TOP500 list, we believe that examining the SPEC market share will provide more insights into trends in personal computing. Using data from the iLORE database, we generated Figure 4.4. We plot the market share of the processor manufacturers used in SPEC benchmarks against the availability date of each processor. Like in Figure 1 from the TOP500

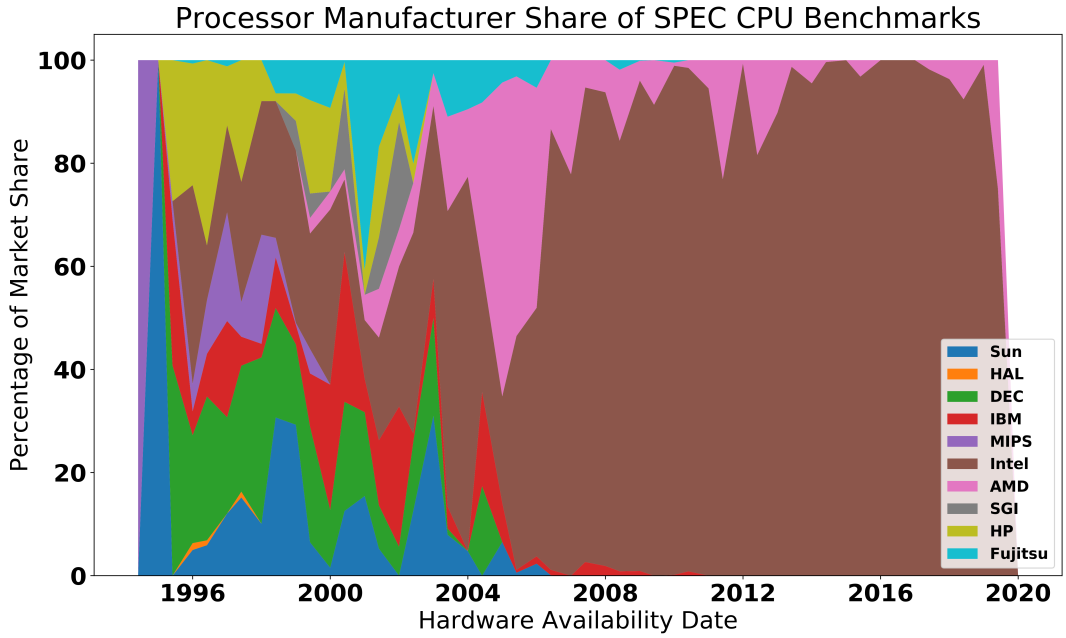

Figure 4.4: Processor market share for SPEC CPU95, CPU2000, CPU2006, and CPU2017.

statistics over time page [103], we bin the raw dates in six-month intervals to decrease noise and get two data points per year.

Relative to the TOP500, heterogeneity in SPEC vendors is severely lacking. The AMD and Intel duopoly dominate modern SPEC suites. Many companies including DEC, MIPS, HP exited the SPEC data in the early 2000s. Vendors like IBM and Fujitsu, which still have strong showings on recent TOP500 lists, only held on to their meager SPEC presence until around 2010.

We draw particular attention to the decline of Silicon Graphics, Inc. (SGI). SGI was a manufacturer that specialized in three-dimensional graphics workstations. The co-founder of Ars Technica, Jon Stokes, attributes the downfall of SGI to three factors: the closing of the gap between CISC and RISC performance, the disappearance of SGI workstations' bus bandwidth advantage, and increased specialization in hardware companies [104]. Up until

the mid-1990s, RISC chips held an advantage over CISC chips because of their lightweight decoding logic. Stokes indicates that, at this point, this advantage disappeared because P6 processors spent only a small portion of their transistor budget on decoding the x86 ISA. Moreover, SGI workstations once held an advantage over commodity machines because proprietary machines can implement custom wide busses with significantly greater bandwidth. This advantage lessened as microprocessors started to implement larger and layered caches that reduced the need for high-bandwidth busses. Lastly, towards the end of SGI's prime, hardware components designed by vendors of individual components combined into machines using industry-standard protocols proved to be more cost-effective. Therefore, SGI's market was encroached upon by personal computing machines.

Figure 4.4 demonstrates the capabilities of the iLORE database with respect to investigating trends at the intersection of component and benchmark data across industries. We can track and analyze interesting developments in the microprocessor industry and supercomputers using this data.

4.2.2 Exploring The Processor Lineage

Here, we demonstrate the utility of the microprocessor lineage through both qualitative and quantitative analyses.

The Lineage of Video Game Consoles

One goal of the CSGenome project is a deeper understanding of culturally significant technologies. We select video games as a category of technologies that drive innovations in computation and are used by the public. To illustrate, Sony's PlayStation 3 console was often used to construct clusters like in the case of Virginia Tech's RidgeRunner Cluster

[105].

We first investigate the lineage of the best-selling video game console of all time, Sony's PlayStation 2 [106]. We select the PlayStation 2 because, by its sales figures, Sony's second console is the most influential.

The short ancestry of the Emotion Engine processor that powers Sony's PlayStation 2 is displayed in Figure 4.5. The oldest ancestor, the MIPS R5000, was designed by Quantum Effect Design and licensed by MIPS Technologies, Inc. in 1995 [107]. The MIPS R5000 has relatively large level one caches designed to run modern desktop computing applications with a focus on 3D graphics. It was designed as an upgrade path for users of the existing R4600 and R4700 processors.

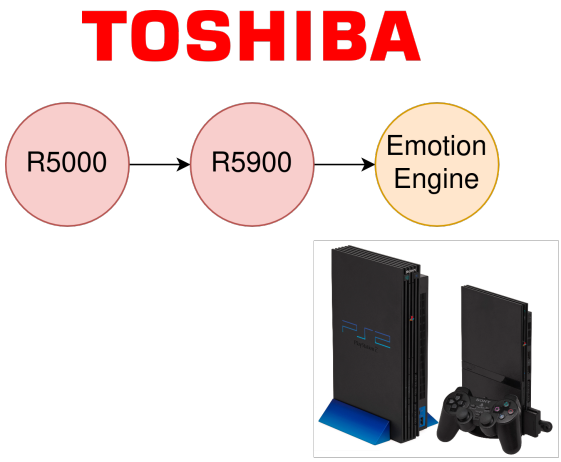

Figure 4.5: A close look at the short lineage of the Emotion Engine processor that powers the Sony Play Station 2.

The R5900, an extension of the R5000, was designed by Sony and Toshiba in 1998 exclusively for use in the PlayStation 2 [108]. The R5900 includes 107 proprietary, 128-bit single instruction multiple data (SIMD) instructions. These instructions allow the R5900 to further specialize in the 3D graphics rendering workloads required of the PlayStation 2. The Emotion Engine is comprised of the R5900, two vector processing units, and a graphics interface unit.

Next, we investigate the Nintendo 64, Gamecube, Wii, and Wii U to both demonstrate contrasting patterns in the lineage and present another aspect of our data that reveals the fall of SGI.

The lineage of the Nintendo 64's Reality Immersion Technology is provided in Figure 4.6. The earliest member of this lineage, the MIPS R4200, was designed as a low-cost RISC processor for embedded and mobile computing in 1992 [109]. The R4200 implements the classic 5-stage MIPS pipeline. The R4300i is an extension of the R4200 intended for low-power digital media manipulation released in 1994 [110]. To achieve reduced size and power, the R4300i includes two power-saving modes and a 32-bit external bus. The R4300i, licensed as VR4300 by NEC, is combined with SGI's Reality Coprocessor to create the Reality Immersion Technology [111].

Figure 4.6: A close look at the short lineage of the Reality Immersion Technology processor that powers the Nintendo 64.

The lineage of Nintendo's Gamecube, Wii, and Wii U consoles is provided in Figure 4.7. In contrast to Sony's PlayStation 2 and the Nintendo 64, the lineage of these consoles is long and unbroken. Moreover, Nintendo's transition from SGI to IBM in the lineage is another marker of the fall of SGI. Because SGI had moved away from graphics specialization in the

early 2000s when Nintendo was designing the Gamecube, Nintendo turned to IBM for a processor.

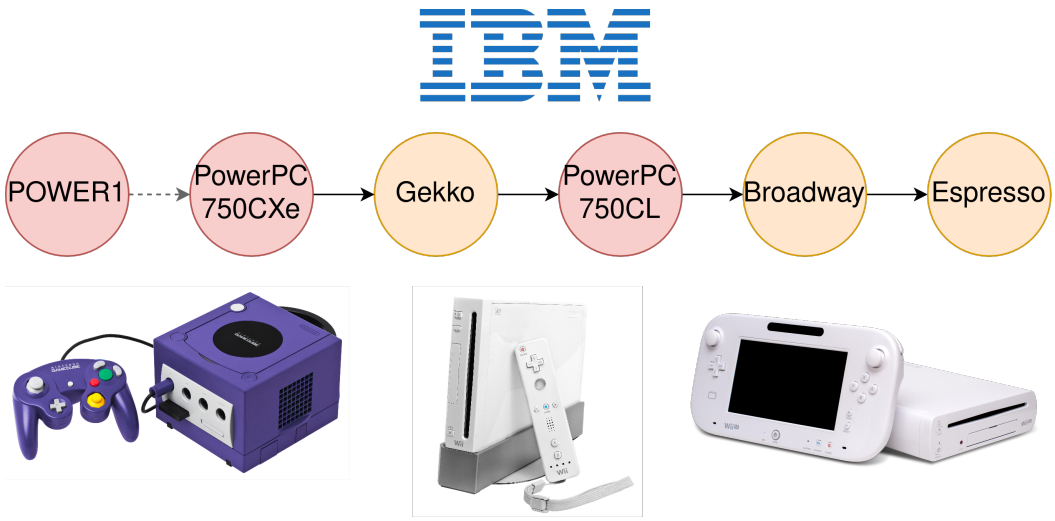

Figure 4.7: A close look at the long, unbroken lineage of Nintendo's Gamecube, Wii, and Wii U consoles.

The most immediate ancestor in this IBM lineage is the PowerPC 750CXe released in 2000. The PowerPC 7xx line of the processors was designed by IBM and Motorola for embedded applications like printers, routers, storage devices, and video game consoles [112]. Similar to the R5900, the Gekko processor, released in 2000, is an enhancement of its parent that includes a collection of SIMD instructions for graphics processing [113]. The subsequent PowerPC 750CL is a die-shrink of the Gekko designed in 2005 for general-purpose use [114]. The Broadway processor of the Nintendo Wii is a clocked-up version of the PowerPC 750CL that was also released in 2005 [115]. Unfortunately, IBM and Nintendo have declined to release more details about the Wii U's 2011 Espresso processor to the public. However, secondary sources reveal that the Espresso is a three-core enhancement to the Broadway processor [116].

The grey dashed line in Figure 4.7 indicates that IBM's POWER1 processor is a distant ancestor of the chips that power modern Nintendo consoles. The POWER1 processor is

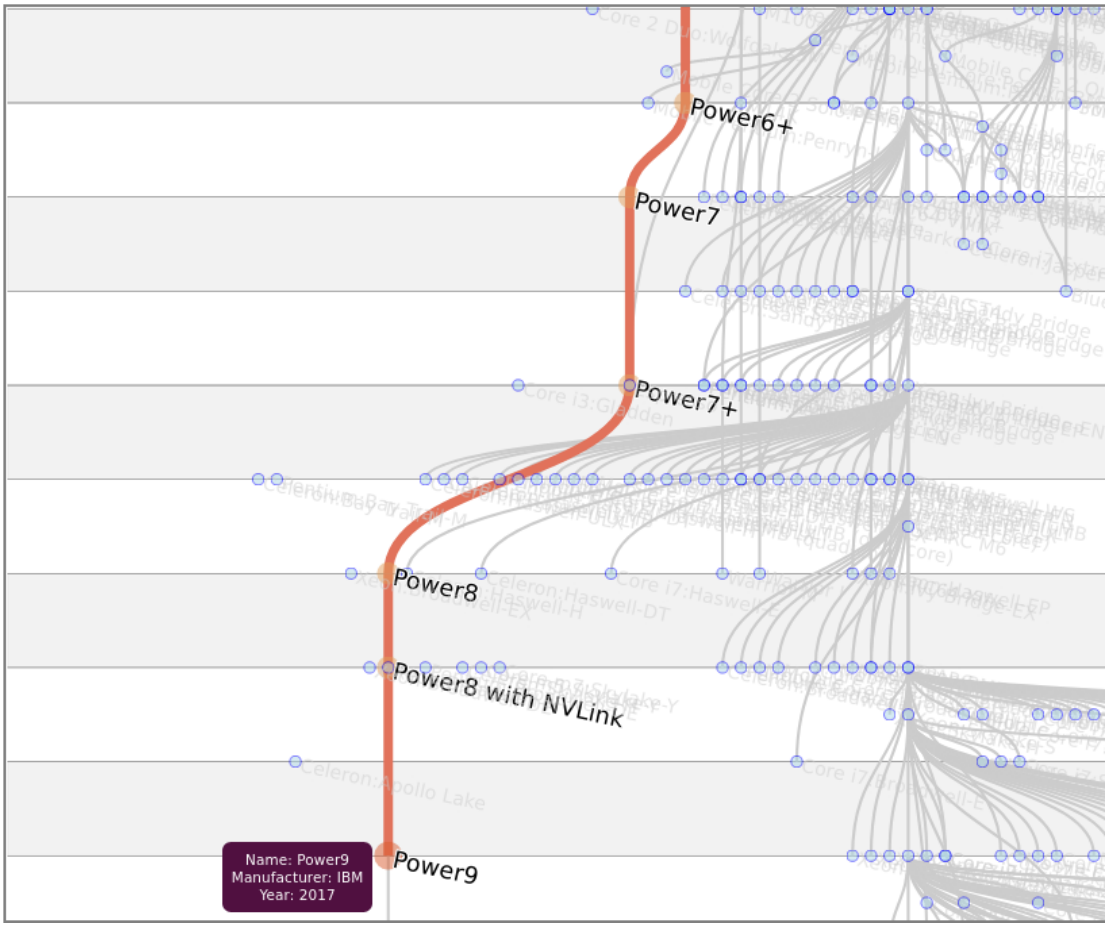

Figure 4.8: Distant relatives of the Gekko, Espresso, and Broadway processors. More details and the rest of the processor lineage can be found at https://ilore.cs.vt.edu/lineage.

also a distant ancestor of the POWER9 processor used to power Oak Ridge National Lab's (ORNL) Summit supercomputer [117]. Figure 4.8 shows the lineage of the distant relations of the processors that power these Nintendo consoles.

Despite occupying different branches of the lineage, the processors that power these video game consoles share some characteristics and design features. For example, both the R5000 and R4200 lines of processors adhere to the simple MIPS design philosophy. Because video game consoles are designed to specialize in 3D graphics rendering, it is common for console CPUs to include proprietary SIMD instructions.

This analysis demonstrates the ability of the microprocessor lineage to augment works like the case studies [1, 12, 13, 14, 15] presented in Section 2.1.

Extending Brainiacs vs. Speed demons

The processor performance equation [118] is a model that captures the influence of microprocessor design tradeoffs on performance. Equation (4.1) is a common formulation of the law provided by Hennessy and Patterson. The first two factors are controlled by the ISA and the last factor is controlled by the microarchitecture.

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{ClockCycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{ClockCycles}} \tag{4.1}$$

Most approaches in microprocessor design, including the RISC and CISC approaches [119, 120], argue for optimizing different factors of the processor performance equation. CISC architectures, with their powerful but sometimes unwieldy instructions, lower the first factor but raise the second. RISC architectures, with their simple instructions, take the opposite approach. Brainiacs and Speed demons are another set of dueling approaches to microarchitecture design [32]. Brainiac designs prefer to maximize instructions per clock cycle (IPC). Speed demons prefer to minimize time per clock cycle.

Figure 4 from Ronen et al. [32] substantiates the viability of both brainiac and speed demon designs. Figure 4 from Ronen et al. [32] plots SPEC CPU92 single-thread integer performance overclock speed against clock speed. The downward sweeping bands mark positions of equivalent performance. Performance can either be achieved by increasing IPC and moving up on the plot, or by increasing clock speed and moving right on the plot. Figure

4 from Ronen et al. [32] features three processor families that represent both sides of these arguments.

The Alpha family of processors represents the speed demon philosophy of design. They achieve performance primarily through clock speed. IBM's PowerPC family and Intel's Pentium family are closer to the Brainiac camp.

We use our microprocessor lineage to observe how Brainiacs vs. Speed demons plays out over the last couple of decades. To extend Figure 4 from Ronen et al. [32] which uses SPEC CPU92, we plot data from CPU95, CPU2000, CPU2006, and CPU2017. We convert all scores to a CPU2017 equivalent score using the methods outlined in Section 3.5.

In Figure 4.9, we plot SPEC CPU scores over clock speed against clock speed for representative processor genera pulled from the lineage. We have selected one line for each major manufacturer featured in the iLORE database. Because modern AMD processors are a significant departure from previous designs, we plot one line for more classic AMD offerings and one line for recent products. The points plotted in Figure 4.9 represent the highest performing benchmark records that use processors that belong to the target genera. In line with Figure 4.4, only Intel and AMD appear after a certain point.

Figure 4.9 suggests that vendors favored the Speed demon approach beyond 1995 with all lines of the lineage achieving performance primarily through frequency increases. All early lines show a preference from the Speed demon approach. Clock speeds consistently increase until designs hit the power wall at around 2700 MHz. Increases in clock speed are rarer after this point as designers turn to improve IPC to bolster single-thread performance.

Gwennap corroborates the dominance of the Speed demon approach in his 1999 article [121]. He indicates that Brainiac and Speed demon approaches had multiple representatives in 1993. IBM's Power, Sun's SuperSparc, and Motorola's 88110 were high-IPC Brainiac

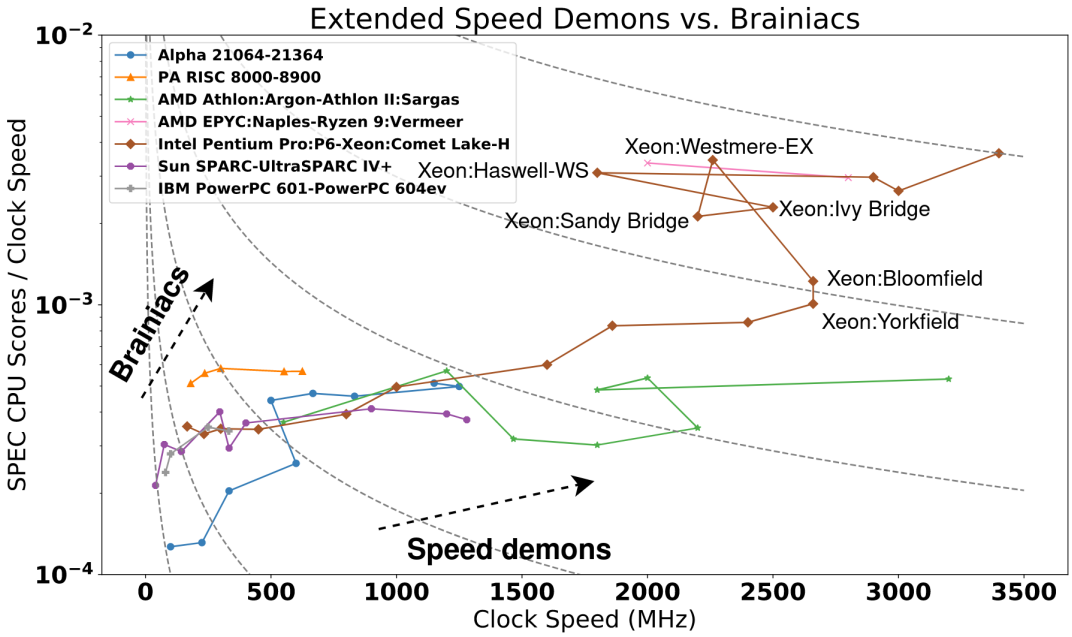

Figure 4.9: Extending Figure 4 from Ronen et al. [32] using the microprocessor lineage and performance data from SPEC CPU95, CPU2000, CPU2006, and CPU2017.

designs. DEC's Alpha, HP's PA-RISC, and MIPS R4000 were Speed demons that aimed for high clock speeds. However, throughout the 1990s, Speed demon approach began to dominate. Gwennap indicates the clock speed gap between the slowest and fastest processor families shrunk from a factor of 5 to a factor of 2.5 from 1993 to 1999.

We draw attention to the back and forth pattern in the latter portion of the Intel lineage. We hypothesize that this pattern is the result of Intel's Tick-Tock production model [80]. Under this model Intel alternated process shrinks, "Ticks," and new microarchitectures "Tocks". We label six genera that follow this production model [122, 123, 124, 125, 126, 127]. The Xeon:Yorkfield model represents a Tick step and subsequent designs follow the alternating pattern. The Xeon:Westmere-EX is a an outlier because it is a server chip whereas the rest of the genera in the Intel line are workstation processors. Ignoring this outlier, Ticks, or process shrinks, seem to provide faster clock speeds and Tocks, or new microarchitectures,

seem to provide improved IPC.

Although we use SPEC integer scores as a measure of single-core performance, many recent benchmarks are conducted with the "Auto Parallel" flag toggled on. Danowitz et al. discuss how the Auto Parallel flag causes these single core benchmark scores to increase due to multicore effects [42]. Future efforts should be undertaken to quantify the sensitivity of single-score SPEC benchmarks to core counts. Moreover, a similar analysis should be conducted using rate scores. These analyses may reveal a more modern approach in the multicore era that challenges both the Speed demon and Brainiac approach.

These analyses demonstrate the validity and utility of the iLORE database in its current state. In Section 5.1 we discuss integrating additional data sets that would enable further analysis.

Chapter 5

Conclusions and Future Work

5.1 Future Work

The work accomplished in service of this thesis serves primarily as a foundation upon which many further studies may be conducted.

The first class of opportunities for growth are improvements to the underlying data and schema of the iLORE repository. There are several known weaknesses of the data that represent low-hanging fruit in near future work.

Although we track which ISA each microarchitecture implements, just recording the ISA does not give a clear picture of the instructions supported by a particular piece of hardware. Particularly egregious offenders include long running ISAs such as x86. Since its introduction over 40 years ago, x86 has received numerous updates and enhancements in the form of ISA extensions. This information could help us understand how single instruction multiple data (SIMD) extensions impacted performance across CPU benchmarks. Efforts should be made to integrate this information into the iLORE repository.

As it stands, the earliest performance metrics in our database come from the first pair of TOP500 lists released in 1993. To better understand how performance evolved during the early stages of microprocessor development, we should incorporate the earlier SPEC CPU92 and other early performance metrics.

Another notable area of weakness is the lack of ARM microprocessors in the database. Over the past 40 years, ARM's presence in the HPC benchmarks that serve as our primary measures of performance has been extremely limited until the June 2020 TOP500 list [128]. The ARM-powered super computer Fugaku debuted at the number one spot on this list. Because it is likely that ARM will have an increasingly large presence in the HPC world, it is clear that efforts should be made to incorporate these processors in iLORE.

The final noteworthy weakness of our data model concerns the storage of internally heterogeneous microprocessors. While we account for heterogeneity at a system level, the current database schema is not expressive enough to accurately model a processor whose individual cores are significantly different. Considering the recent release of the Apple M1 [129], these heterogeneous chips are likely to become commonplace.

One important vehicle for accomplishing these improvements is help from the community. As our API becomes more sophisticated, we plan to create tools that would allow our users to provide us feedback and suggest revisions for our data. This is particularly helpful for lineage data which is manually created based on a set of guiding principles but is ultimately subjective. CPU-World's CWID program is one instance of successful crowd sourcing in a previous repository [130]. CWID is a program that users download and run on their local machines. The program runs the cpuid instruction on the users machine and sends the results to CPU-World's database. This program allows CPU-World to collect very detailed information on cache topologies and ISA extensions among other attributes. Providing a similar set of instructions to our users could unlock further analysis opportunities.

Moreover, we can work to further augment our component attribute data with taxonomic information. The current layers of the microprocessor taxonomy aid us in a number of goals. First, grouping many processors under a single genus enables us to document the meaningful adaptations in microprocessor design without handling the full complexity associated with

the vast number of products shipped each year. Second, grouping many genera under a single microarchitecture provides us with yet larger groups at which it is appropriate to compare performance and other metrics. Third, grouping many microarchitectures under a single ISA enables us to track how different implementation strategies are used to implement the same instructions over time. Further work should be conducted to explore the possibility of adding additional layers to the taxonomy. Processors and accelerators could potentially be unified at one of these higher layers. With a more generalized taxonomy, we might gain a better understanding of how the different pieces of a computer system fit together.

The overarching goal of the CSGenome project is to provide a deeper understanding of how computer systems and their performance have matured over time. The lineage of microprocessors presented here is but one piece among many when it comes to fully capturing the evolution of computer systems. Moving forward, we will continue to integrate data on additional computer components. As we approach a more comprehensive collection of computer component attributes and lineage data, we can start to think about combining the lineage of different components to create some aggregate measure of evolution between two systems.

5.2 Conclusion

This thesis presents the techniques and methods that were used to construct and analyze the iLORE data model. iLORE endeavors to represent the rich relationships between the lineage of microprocessors, computer systems, and benchmarks. Although these data sets are currently available to the public in one form or another, they are often scattered across disparate sources and require significant wrangling before meaningful insights can be made. Through programmatic and manual efforts, we integrate these siloed data sets, augmented with lineage data, into the unified iLORE repository.

Through our independent recreation of the results of previously-published analyses, we have gained confidence in the efficacy of our methods. Through our new analyses, we have demonstrated the utility of our still-growing database. This work represents a solid first step towards the CSGenome project's overarching goal of understanding the history of computer performance.

Bibliography

[1] F. Faggin, "The making of the first microprocessor," *IEEE Solid-State Circuits Magazine*, vol. 1, no. 1, pp. 8–21, 2009.

[2] SPEC, "speccpu benchmarks," https://www.spec.org/benchmarks.html, 2021, accessed: 2021-01-19.

[3] TOP500, "top500 lists," https://www.top500.org/, 2021, accessed: 2021-01-13.

[4] W. Feng and K. Cameron, "The green500 list: Encouraging sustainable supercomputing," *Computer*, vol. 40, no. 12, pp. 50–55, 2007.

[5] A. Danowitz, "stanford cpu db," http://cpudb.stanford.edu/, 2014, accessed: 2021-01-13.

[6] CPU-World, "Cpu-world," https://www.cpu-world.com/, 2021, accessed: 2021-01-24.

[7] "Wikichip," https://en.wikichip.org/wiki/WikiChip, 2021, accessed: 2021-01-24.

[8] TOP500, "Highlights - november 2020," https://www.top500.org/lists/top500/2020/11/highs/, 2020, accessed: 2021-01-24.

[9] "Gpuzoo," https://www.gpuzoo.com/, 2021, accessed: 2021-01-24.

[10] "Userbenchmark," https://cpu.userbenchmark.com/, 2021, accessed: 2021-01-24.

[11] R. N. Noyce and M. E. Hoff, "A history of microprocessor development at intel," *IEEE Micro*, vol. 1, no. 1, pp. 8–21, 1981.

[12] Morse, Raveiel, Mazor, and Pohiman, "Intel microprocessors–8008 to 8086," *Computer*, vol. 13, no. 10, pp. 42–60, 1980.

[13] S. Mazor, "Intel 8080 cpu chip development," *IEEE Annals of the History of Computing*, vol. 29, no. 2, pp. 70–73, 2007.

[14] ——, "Intel's 8086," *IEEE Annals of the History of Computing*, vol. 32, no. 1, pp. 75–79, 2010.

[15] T. R. Gross, N. P. Jouppi, J. L. Hennessy, S. Przybylski, and C. Rowen, "A retrospective on "mips: A microprocessor architecture"," *IEEE Computer Society*, vol. 36, no. 4, pp. 73–76, 2016.

[16] B. Evans, "System/360: A retrospective view," *Annals of the History of Computing*, vol. 8, no. 2, pp. 155–179, 1986.

[17] S. Mazor, "The history of the microcomputer-invention and evolution," *Proceedings of the IEEE*, vol. 83, no. 12, pp. 1601–1608, 1995.

[18] ——, "Moore's law, microcomputer, and me," *IEEE Solid-State Circuits Magazine*, vol. 1, no. 1, pp. 29–38, 2009.

[19] A. Hartstein and T. R. Puzak, "The optimum pipeline depth for a microprocessor," in *Proceedings of the 29th Annual International Symposium on Computer Architecture*, ser. ISCA '02. Anchorage, Alaska: IEEE Computer Society, 2002, pp. 7—13.

[20] P. K. Dubey and M. J. Flynn, "Optimal pipelining," *Journal of Parallel and Distributed Computing*, vol. 8, no. 1, pp. 10–19, 1990.

[21] E. Sprangle and D. Carmean, "Increasing processor performance by implementing deeper pipelines," *SIGARCH Comput. Archit. News*, vol. 30, no. 2, pp. 25–34, 2002.

[22] J. E. Smith, "A study of branch prediction strategies," in *Proceedings of the 8th Annual Symposium on Computer Architecture*, ser. ISCA '81. Washington, DC: IEEE Computer Society, 1981, p. 135–148.

[23] T.-Y. Yeh and Y. N. Patt, "Alternative implementations of two-level adaptive branch prediction," in *25 Years of the International Symposia on Computer Architecture (selected papers)*, ser. ISCA '98. New York, NY: Association for Computing Machinery, 1998, p. 451–461.

[24] S. McFarling, "Combining branch predictors," in *WRLTechnical Note TN-36*, ser. WRLTechnical Note TN-36. Palo Alto, CA: Digital Western Research Laboratory, 1993, pp. 1–2.

[25] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," *SIGARCH Comput. Archit. News*, vol. 23, no. 2, pp. 392–403, 1995.

[26] B. Nayfeh and K. Olukotun, "A single-chip multiprocessor," *Computer*, vol. 30, no. 9, pp. 79–85, 1997.

[27] D. Papworth, "Tuning the pentium pro microarchitecture," *IEEE Micro*, vol. 16, no. 2, pp. 8–15, 1996.

[28] J. Tendler, J. S. Dodson, J. S. Fields, H. Q. Le, and B. Sinharoy, "Power4 system microarchitecture," *IBM J. Res. Dev.*, vol. 46, no. 1, pp. 5–26, 2002.

[29] D. A. Koufaty and D. Marr, "Hyperthreading technology in the netburst microarchitecture," *IEEE Micro*, vol. 23, no. 1, pp. 56–65, 2003.

[30] Y. Patt, "Requirements, bottlenecks, and good fortune: agents for microprocessor evolution," *Proceedings of the IEEE*, vol. 89, no. 11, pp. 1553–1559, 2001.

[31] C. E. Kozyrakis and D. A. Patterson, "A new direction for computer architecture research," *Computer*, vol. 31, no. 11, pp. 24–32, 1998.

[32] R. Ronen, A. Mendelson, K. Lai, Shih-Lien Lu, F. Pollack, and J. P. Shen, "Coming challenges in microarchitecture and architecture," *Proceedings of the IEEE*, vol. 89, no. 3, pp. 325–340, 2001.

[33] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23–29, 1999.

[34] S. Borkar and A. A. Chien, "The future of microprocessors," *Commun. ACM*, vol. 54, no. 5, pp. 67–77, 2011.

[35] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA '11. New York, NY: Association for Computing Machinery, 2011, pp. 365–376.

[36] ——, "Power challenges may end the multicore era," *Commun. ACM*, vol. 56, no. 2, pp. 93–102, 2013.

[37] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[38] TOP500, "Top500 list: The linpack benchmark," https://top500.org/project/linpack/, 2021, accessed: 2021-01-19.

[39] ——, "Top500 list: The linpack benchmark," https://top500.org/project/linpack/, 2021, accessed: 2021-01-19.

[40] E. E. H. W. Group, "Power measurement methodology," https://github.com/EEHPCWG/PowerMeasurementMethodology, 2015, accessed: 2021-05-27.

[41] "Spec cpu 2017," https://www.spec.org/cpu2017/, 2021, accessed: 2021-05-27.

[42] A. Danowitz, K. Kelley, J. Mao, J. Stevenson, and M. Horowitz, "Cpu db: Recording microprocessor history," *Communications of The ACM - CACM*, vol. 55, no. 1, pp. 55–63, 2012.

[43] J. J. Dujmovic and I. Dujmovic, "Evolution and evaluation of spec benchmarks," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 2—9, 1998.

[44] J. L. Henning, "Spec cpu suite growth an historical perspective," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 1, pp. 65–68, 2007.

[45] R. Panda, S. Song, J. Dean, and L. K. John, "Wait of a decade: Did spec cpu 2017 broaden the performance horizon?" in *2018 IEEE International Symposium on High Performance Computer Architecture*, ser. HPCA '18. Washington, DC: IEEE Computer Society, 2018, pp. 271–282.

[46] A. Phansalkar, A. Joshi, and L. K. John, "Subsetting the spec cpu2006 benchmark suite," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 69—76, 2007.

[47] ——, "Analysis of redundancy and application balance in the spec cpu2006 benchmark suite," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY: Association for Computing Machinery, 2007, pp. 412–423.

[48] H. Vandierendonck and K. De Bosschere, "Many benchmarks stress the same bottlenecks," in *Workshop on Computer Architecture Evaluation Using Commercial Workloads*, ser. CAECW '04. Washington, DC: IEEE Computer Society, 2004, pp. 57–64.

[49] A. Phansalkar, A. Joshi, L. Eeckhout, and L. John, "Measuring program similarity: Experiments with spec cpu benchmark suites," in *IEEE International Symposium on*

Performance Analysis of Systems and Software, 2005, ser. ISPASS '05. Washington, DC: IEEE Computer Society, 2005, pp. 10–20.

[50] J. D. Gee, M. Hill, D. Pnevmatikatos, and A. J. Smith, "Cache performance of the spec92 benchmark suite," *IEEE Micro*, vol. 13, no. 1, pp. 17–27, 1993.

[51] M. J. Charney and T. R. Puzak, "Prefetching and memory system behavior of the spec95 benchmark suite," *IBM Journal of Research and Development*, vol. 41, no. 3, pp. 265–286, 1997.

[52] S. Sair and M. Charney, "Memory behavior of the spec 2000 benchmark suite," IBM Thomas J. Watson Research Center, Ossining, NY, Tech. Rep. RC-21852, 2000.

[53] A. Jaleel *et al.*, "Memory characterization of workloads using instrumentation-driven simulation–a pin-based memory characterization of the spec cpu2000 and spec cpu2006 benchmark suites," Intel, Santa Clara, CA, Tech. Rep. VSSAD, 2007.

[54] S. Singh and M. Awasthi, "Memory centric characterization and analysis of spec cpu2017 suite," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '19. New York, NY: Association for Computing Machinery, 2019, pp. 285–292.

[55] A. Kejariwal, A. V. Veidenbaum, A. Nicolau, X. Tian, M. Girkar, H. Saito, and U. Banerjee, "Comparative architectural characterization of spec cpu2000 and cpu2006 benchmarks on the intel® core™ 2 duo processor," in *2008 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, ser. IC-SAMOS '08. Washington, DC: IEEE Computer Society, 2008, pp. 132–141.

[56] H. Oi, "Evaluation of ryzen 5 and core i7 processors with spec cpu 2017," in *2019 IEEE*

International Systems Conference (SysCon), ser. SysCon '19. Washington, DC: IEEE Computer Society, 2019, pp. 1–6.

[57] D. Bhandarkar and J. Ding, "Performance characterization of the pentium pro processor," in *1997 IEEE International Symposium on High Performance Computer Architecture*, ser. HPCA '97. Washington, DC: IEEE Computer Society, 1997, pp. 288–297.

[58] J. Dongarra, J. Bunch, C. Moler, and G. Stewart, *LINPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1979.

[59] J. Dongarra, "The linpack benchmark: An explanation," in *ICS*, ser. LNCS, volume 297. Argonne, Illinois: Argonne National Laboratory, 1987, pp. 1–2.

[60] ——, "Performance of various computers using standard linear equations software," *SIGARCH Comput. Archit. News*, vol. 20, no. 3, pp. 22–44, 1992.

[61] J. Dongarra, P. Luszczek, and A. Petitet, "The linpack benchmark: past, present and future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 1, pp. 803–820, 2003.

[62] J. Dongarra and P. Luszczek, *HPC Challenge Benchmark*. Boston, MA: Springer US, 2011, pp. 844–850.

[63] J. Dongarra and M. A. Heroux, "Toward a new metric for ranking high performance computing systems," *Sandia Report, SAND2013-4744*, vol. 312, no. 1, p. 150, 2013.

[64] J. Dongarra, M. Heroux, and P. Luszczek, "A new metric for ranking high performance computing systems," *National Science Review*, vol. 3, no. 1, pp. 1–2, 2016.

[65] "Beautiful Soup," https://www.crummy.com/software/BeautifulSoup/bs4/doc/, 2021, accessed: 2021-05-27.

[66] "Pandas," https://pandas.pydata.org/, 2021, accessed: 2021-05-27.

[67] "xlrd," https://pypi.org/project/xlrd/, 2021, accessed: 2021-06-06.

[68] "The performance database server," http://performance.netlib.org/performance/html/spec.html, 1996, accessed: 2021-01-19.

[69] AMD, "amd product specifications," https://www.amd.com/en/products/specifications/processors, 2021, accessed: 2021-01-24.

[70] Intel, "arkintel product specifications," https://ark.intel.com/content/www/us/en/ark.html, 2021, accessed: 2021-01-24.

[71] P. Weissmann, "Early hp 9000 pa-risc systems," https://www.openpa.net/systems/hp_early-systems.html, 2021, accessed: 2021-05-27.

[72] "Lucidchart overview," https://www.lucidchart.com/pages/product, 2021, accessed: 2021-05-27.

[73] "Graphviz - graph visualization software," https://graphviz.org/, 2021, accessed: 2021-05-27.

[74] L. Gwennap, "Ibm crams power2 onto single chip," *Microprocessor Report*, vol. 256, no. 1, p. 128, 1996.

[75] M. P. Robert, R. Dinkjian, M. Mayfield, P. Lenk, and B. Ciarfella, "Power3: Next generation 64-bit powerpc processor design authors," IBM, Austin, Texas, Tech. Rep., 1998.

[76] "Am186 - amd," https://en.wikichip.org/wiki/amd/am186, 2016, accessed: 2021-05-27.

[77] Intel, *INTRODUCTION TO THE iAPX 432 ARCHITECTURE*, Intel, Santa Clara, CA, 1981.

[78] L. Kohn and N. Margulis, "Introducing the intel i860 64-bit microprocessor," *IEEE Micro*, vol. 9, no. 4, pp. 15–30, 1989.

[79] "Intel instruction set extensions technology," https://www.intel.com/content/www/us/en/support/articles/000005779/processors.html, 2021, accessed: 2021-05-27.

[80] "Intel tick tock model," https://www.intel.com/content/www/us/en/silicon-innovations/intel-tick-tock-model-general.html, 2021, accessed: 2021-05-27.

[81] "Core name / codename," https://www.cpu-world.com/Glossary/C/Core_name.html, 2018, accessed: 2021-05-27.

[82] "Core stepping," https://www.cpu-world.com/Glossary/C/Core_stepping.html, 2018, accessed: 2021-05-27.

[83] "Sqlalchemy," https://www.sqlalchemy.org/, 2021, accessed: 2021-03-15.

[84] E. Rahm and H. Do, "Data cleaning: Problems and current approaches," *IEEE Data Eng. Bull.*, vol. 23, no. 1, pp. 3–13, 2000.

[85] "Tpc," http://www.tpc.org/information/benchmarks5.asp, 2021, accessed: 2021-01-24.

[86] L. Getoor and A. Machanavajjhala, "Entity resolution: Tutorial," https://users.umiacs.umd.edu/~getoor/Tutorials/ER_VLDB2012.pdf, 2012, accessed: 2021-05-27.

[87] "Dedupe," https://github.com/dedupeio/dedupe, 2021, accessed: 2021-03-15.

[88] "Marshmallow: Simplified object serialization," https://alembic.sqlalchemy.org/en/latest/, 2021, accessed: 2021-03-15.

[89] "Flask," https://alembic.sqlalchemy.org/en/latest/, 2021, accessed: 2021-03-15.

[90] "Alembic," https://alembic.sqlalchemy.org/en/latest/, 2021, accessed: 2021-03-15.

[91] SPEC, "Spec cpu2006 press background," https://www.spec.org/cpu2006/Docs/readme1st.html, 2006, accessed: 2021-05-15.

[92] R. Munafo, "The spec benchmarks," https://mrob.com/pub/comp/benchmarks/spec.html, accessed: 2021-04-13.

[93] K. Rupp, "40 years of microprocessor trend data," https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/, 2015, accessed: 2021-05-26.

[94] ——, "42 years of microprocessor trend data," https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/, 2018, accessed: 2021-05-10.

[95] ——, "48 years of microprocessor trend data," https://github.com/karlrupp/microprocessor-trend-data, 2020, accessed: 2021-05-10.

[96] G. E. Moore, "Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff." *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, 2006.

[97] M. Bohr, "A 30 year retrospective on dennard's mosfet scaling paper," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 11–13, 2007.

[98] T. Mudge, "Power: a first-class architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52–58, 2001.

[99] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The case for a single-chip multiprocessor," in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS VII. New York, NY: Association for Computing Machinery, 1996, pp. 2–11.

[100] Intel, "Intel quark microcontroller d1000," https://ark.intel.com/content/www/us/en/ark/products/86826/intel-quark-microcontroller-d1000.html, 2015, accessed: 2021-06-04.

[101] TOP500, "Top500 performance development," https://www.top500.org/statistics/perfdevel/, 2021, accessed: 2021-06-08.

[102] ——, "Top500 list statistics," https://www.top500.org/statistics/, 2021, accessed: 2021-05-10.

[103] ——, "Top500 vendor system share over time," https://www.top500.org/statistics/overtime/, 2021, accessed: 2021-06-08.

[104] J. Stokes, "Sgi: anatomy of a fall," https://arstechnica.com/uncategorized/2005/11/5541-2/, 2005, accessed: 2021-05-26.

[105] V. Tech, "Accelerators at virginia tech," https://accel.cs.vt.edu/facilities.php, 2012, accessed: 2021-05-26.

[106] D. M. Ewalt, "Sony playstation2 sales reach 150 million units," https://accel.cs.vt.edu/facilities.php, 2012, accessed: 2021-05-26.

[107] M. T. Inc., *MIPS R5000 Microprocessor Technical Backgrounder*, MIPS Technologies Inc., Santa Clara, CA, 1996.

[108] K. Diefendorff, "Sony's emotionally charged chip killer floating-point "emotion engine" to power playstation 2000," *Microprocessor Report*, vol. 13, no. 5, pp. 1–7, 1999.

[109] B. Zivkov, B. Ferguson, and M. Gupta, "R4200: a high-performance mips microprocessor for portables," in *Proceedings of COMPCON '94*, ser. COMPCON '94. Los Alamitos, CA: IEEE Computer Society, 1994, pp. 18–25.

[110] I. MIPS Technologies, "Mips and nec announce advanced 64-bit risc processor to power new class of interactive consumer products," http://www.sgidepot.co.uk/mips-nec.html, 1995, accessed: 2021-05-26.

[111] J. Peddie, "Famous graphics chips: Nintendo 64," https://www.computer.org/publications/tech-news/nintendo-64, n.d., accessed: 2021-05-26.

[112] I. M. Systems and T. Group, *IBM PowerPC 750CX/CXe/CXr RISC Microprocessor User's Manual*, IBM, Hopewell Junction, NY, 2005.

[113] ——, *IBM Gekko RISC Microprocessor User's Manual*, IBM, Hopewell Junction, NY, 2000.

[114] ——, *IBM PowerPC 750CL Microprocessor Revision Level DD2.X Datasheet*, IBM, Hopewell Junction, NY, 2006.

[115] ——, *IBM Broadway RISC Microprocessor User's Manual*, IBM, Hopewell Junction, NY, 2006.

[116] R. George, "Wii u cpu, gpu details uncovered," https://www.ign.com/articles/2012/11/29/wii-u-cpu-gpu-details-uncovered, 2012, accessed: 2021-05-26.

[117] ORNL, "Ornl launches summit supercomputer," https://www.ornl.gov/news/ornl-launches-summit-supercomputer, 2018, accessed: 2021-05-26.

[118] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2011.

[119] D. Bhandarkar and D. W. Clark, "Performance from architecture: Comparing a risc and a cisc with similar hardware organization," in *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS IV. New York, NY: Association for Computing Machinery, 1991, pp. 310–319.

[120] D. Bhandarkar, "Risc versus cisc: A tale of two chips," *SIGARCH Comput. Archit. News*, vol. 25, no. 1, pp. 1–12, 1997.

[121] L. Gwennap, "Brainiacs, speed demons, and farewell," *Microprocessor Report*, vol. 13, no. 15, pp. 1–2, 1999.

[122] Intel, "Intel xeon processor w3520," https://ark.intel.com/content/www/us/en/ark/products/39718/intel-xeon-processor-w3520-8m-cache-2-66-ghz-4-80-gt-s-intel-qpi.html, 2009, accessed: 2021-06-04.

[123] ——, "Intel xeon processor e3-1230l v3," https://ark.intel.com/content/www/us/en/ark/products/75053/intel-xeon-processor-e3-1230l-v3-8m-cache-1-80-ghz.html, 2013, accessed: 2021-06-04.

[124] ——, "Intel xeon processor e3-1265l v2," https://ark.intel.com/content/www/us/en/ark/products/65728/intel-xeon-processor-e3-1265l-v2-8m-cache-2-50-ghz.html, 2012, accessed: 2021-06-04.

[125] ——, "Intel xeon processor e3-1220l," https://ark.intel.com/content/www/us/en/ark/products/53401/intel-xeon-processor-e3-1220l-3m-cache-2-20-ghz.html, 2011, accessed: 2021-06-04.

[126] ——, "Intel xeon processor x3350," https://ark.intel.com/content/www/us/en/ark/

products/33932/intel-xeon-processor-x3350-12m-cache-2-66-ghz-1333-mhz-fsb.html, 2008, accessed: 2021-06-04.

[127] ——, "Intel xeon processor e7-8860," https://ark.intel.com/content/www/us/en/ark/products/53572/intel-xeon-processor-e7-8860-24m-cache-2-26-ghz-6-40-gt-s-intel-qpi.html, 2011, accessed: 2021-06-04.

[128] TOP500, "Top500 june 2020," https://www.top500.org/lists/top500/2020/06/, 2020, accessed: 2021-02-01.

[129] "Apple unleashes m1," https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/, 2020, accessed: 2021-05-27.

[130] "Identify cpu features (beta)," https://www.cpu-world.com/Download/CPU_identification.html, 2021, accessed: 2021-02-01.