Abstract—As computer architecture integrates multiple concepts such as microarchitecture, design, the hardware-software interface, compilers, and operating systems, there is an always increasing need to develop new methods for learning and exploring the field. Parallelism in computer systems is a key focus in computer architecture and some core parallel concepts include Amdahl’s law, efficiency, and overhead. While there are a number of ways to examine these topics in traditional lectures and assignments, a unique way is to leverage Python-based programming environments that allow students to independently explore concepts and their governing parameters.

This paper presents the highlights of PyCompArch Python module developed using the IPython Notebook environment to help the study of concepts in computer architecture. Python is a widely used general-purpose, high-level programming language, but traditionally the language does not play a leading role in the education of computer architecture. IPython Notebooks allow developers to interactively run Python code cells and to construct Python codes that execute on remote servers that eliminate any system requirements of the individual. In this way, the environment supports web-based remote “in the cloud” code development that can be modified during lectures or in homework assignments. The PyCompArch module supports a number of ways to help individuals learn concepts of parallelism related to computer architecture as well as explore experiments in computer performance and control. For example, PyCompArch supports the evaluation of performance of real-world benchmarks such as Open Computer Vision (OpenCV) and dynamic frequency scaling (DFS) in Raspberry Pi systems. Overall, the PyCompArch supports student learning and development of experiments in computer architecture.

I. INTRODUCTION

New technologies are constantly emerging to complement the study of the hardware and software paradigms of the computer architecture field. In many ways, it is critical to deploy the latest technologies into the education process. For example, even with an understanding of the fundamental aspects of computer organization, there are often barriers to comprehend the impact of computer organization structures on overall application performance. Such barriers are especially true for understanding issues related to parallel execution and parallel performance analysis.

There is a rich landscape of computer architecture education tools that crosses several categories such as architecture simulators, tracing tools such as Pin [10], and hardware event monitoring programs. Each of these categories play a role in examining computer architecture by addressing specific areas of interest in the field. For instance, architecture simulators provide insight into microarchitecture design and the behavior of the individual hardware components. Simulators are usually designed so that students can integrate additional emulated hardware components into the simulation system and analyze the impact of design parameters on the simulated processor. Likewise, performance monitoring support allows students to study the performance impact of the different architecture components and compiler optimizations have on the overall system. Most importantly, monitoring systems provide accurate feedback on real workload applications.

While there are a number of ways to examine computer architecture topics in traditional lectures and assignments, a unique way is to leverage Python-based environments. Python is a widely used general-purpose, high-level programming language, but traditionally the language has not played a leading role in the education or research of computer architecture. As Python is interpreted and not regarded for close-to-the-machine performance, it has not traditionally been used within computer architecture education. However the Python syntax allows programmers to express concepts in fewer lines of code and enables a number of modules that create an active learning environment. For example, as Python programming environments can generate visual graphs, learners can in a single framework, build computer architecture experiments that then automatically measure and display performance results for analysis.

In this paper, PyCompArch, a Python-based module is presented and used as a learning framework for exploring computer architecture concepts and experiments.
concepts in computer organization. The PyCompArch framework is illustrated in Figure 1. has the benefit of being set by instructors in computer organization education and modified by student learners. There are two distinct systems constructed within PyCompArch: architecture concepts and architecture experiments. Each of these systems are supported with functionalities such as parameter settings, visualizations, and the automation of the collection of experimental results.

In terms of architecture concepts, a set of modules are designed that demonstrate parallel concepts such as Amdahl’s law, efficiency, overhead, and job scheduling. In each of these cases, active learning themes are constructed such that learners can control parameters that then create visual outcomes in terms of timelines or performance graphs. In addition to teaching core architecture concepts, the PyCompArch framework increases accessibility of resources for students by supporting remote access to servers using Interactive Python (IPython) Notebooks. IPython Notebooks allow developers to interact with existing Python programming cells to construct Python solutions that execute on remote servers accessible from any web browser. Bash shell commands and direct command execution are supported also in cells, as well as the ability to capture output directly into Python variables, allowing architecture experiments to assess output parsing skills to wrangle text into formats that can be graphed. Overall, such forms of development in which programmers develop and run on remote servers using web-based interfaces have significant potential in broadening teaching efforts in both computer architecture and the related field of Parallel and Distributed Systems (PDS). Similarly, for real-time driven computing domains as high-performance computing and computer vision, there are demanding requirements of large data files (videos, image, and data sets), extensive library support, and other requirements.

PyCompArch includes a second set of Python-based controls constructed that can evaluate and collect experimental results from architecture experiments. As Python is a core technology integrated into the Raspberry Pi [5], the PyCompArch framework has been used to evaluate principles of Dynamic Frequency Scaling (DFS) as well as a comparison study between applications on the Raspberry Pi 1.0 and 2.0 models. The following sections provide some background on IPython Notebooks and the PyCompArch modules.

II. PYTHON AND IPYTHON NOTEBOOK BACKGROUND

Python is a high-level programming language that emphasizes code readability. The widely recognized view is that the Python syntax allows programmers to express concepts in fewer lines of code than in languages such as C++ or Java. Moreover, Python supports several programming models, including object-oriented, imperative, and functional that can eliminate substantial development time. As Python is interpreted and not regarded for close-to-the-machine performance, it has not traditionally been used within computer architecture education. Python’s comprehensive standard library and dynamic typing system make the language easy to use and to generate examples in computer architecture concepts.

There are other emerging ways that Python has started to be integrated in areas of computer architecture research and education. PyMTL [9] leverages the Python programming language to create a highly productive domain-specific embedded language for concurrent-structural modeling and hardware design. The PyMTL framework supports the philosophy of “modeling towards layout” in which a microarchitecture is incrementally refined from a high-level functional-level model, to a timing-approximate cycle-level model, to a bit-accurate RTL implementation. PyMTL supports the rapid design space exploration of microarchitectures for novel accelerators, specialized coprocessors, or any design proposal that could benefit from the additional credibility provided by an RTL implementation. Pydgin [8] is a Python framework for rapidly developing instruction-set simulators (ISSs) from architecture description language. Pydgin adapts existing meta-tracing JIT compilation frameworks designed for general-purpose dynamic programming languages to automatically generate ISSs augmented with dynamic binary translation. Pydgin is suitable for generating very fast ISSs for general-purpose instruction sets, but is particularly well-suited for exploring the hardware/software abstraction of emerging specialized architectures.

One web-based framework that supports running a remote server accessible over web-browser systems is the IPython Notebook [1]. The IPython Notebook is a web-based interactive computational environment that in addition to providing a well-structured code development environment, also provides a framework for observing and recording results of code execution, linking text such as comments, equation generators for mathematics, plots and other rich media formatting options. The IPython notebooks are built on JSON format files that can be distributed to class participants that allow learners to interact with code demonstrated lectures by being able to “touch” the code of lecture slides and make individual changes to parameters and statements. As the IPython environment is interactive, code cells, or small sections of code can be tested in-place without requiring the entire program example to be changed. Changes can be rolled back also, encouraging experimentation without creating excessive copies of source material. The cloud coding advantage is that the IPython Notebook Viewer renders the code as a web page and users can read and interact with a remote system without having to install anything on their device. Figure 2 shows an example IPython Notebook with code and execution results.

III. PyCOMPARCH: COMPUTER ARCHITECTURE-BASED IPYTHON NOTEBOOKS

A. Exploring Computer Architecture Concepts

At the concept level, the goal of the PyCompArch module was to provide a framework that can help visualize parallel characteristics of synthetic workloads as well as then also display parallel algorithm results within the same system. The current module construction is to provide some introduction to parallelism and Amdahl’s law. Both of these solutions have visualization solutions to help learners. Since the framework is built to work as IPython Notebooks, students can run the examples on multiple machines and collect those results to compare characteristics between machines or generations of machines.

1) Multiprocessing in Python to Demonstrate Parallelism:

The multiprocessing module of Python is a straightforward way to introduce the concept of multiple processes running
on a target system. Figure 3 provides an example of the code to get started with multi-process parallelism. The goal is not to introduce a formal parallel language but to demonstrate the availability of processing cores and their throughput.

![Parallel execution in Python using multiprocessing.](image)

**Fig. 3.** Parallel execution in Python using multiprocessing.

For an easily parallel example, the Mandelbrot set fractal is one example that has been effective in demonstrating PyCompArch. Figure 4 illustrates an example display as well as the concept of dividing the calculations into independent blocks capable of concurrent execution. The fractal example is helpful as the result presents a visual of the final result versus traditional examples such as summing the values of a large array.

A component of the PyCompArch is the ability to generate the parallel evaluations of code to allow developers to locate the optimal parallel execution of a system. The developer simply provides the constraints on the number of processes to explore. Figure 5 shows another developed feature to generate the time execution.

![Automatic plotting of Mandelbrot execution time evaluation.](image)

**Fig. 5.** Automatic plotting of Mandelbrot execution time evaluation.

Finally, the framework can generate results that compare to ideal performance (no overhead and no serial component of work). The comparison of Figure 6 helps student learners recognize the issues with serial and parallel task contributions. Overall, some of initial goals were to provide the standard performance plotting and graph comparison tools to allow learning of the parallel characteristics of various codes.

![Mandelbrot fractal performance.](image)

**Fig. 6.** Mandelbrot fractal performance.

2) Providing Python Code for Theoretical and Actual Amdahl’s Law Concepts: Building on the previous subsection, a lecture module within PyCompArch was structured to allow learner to visualize and manipulate experiments involving Amdahl’s law. The goal was to provide a synthetic evaluation to explore the maximum expected improvement to system (or code) when only part of the system (or code) is improved.
Since Amdahl’s law can be used to predict the theoretical maximum speedup using multiple processors, the goal was to help developers compare real application performance to the synthetic performance. In this way, the speedup of a program using multiple processors in parallel computing could be compared to the synthetic model to help determine the sequential fraction of the real program. Figure 7 shows an example output and Figure 8 shows some of the PyCompArch codebase that provides the support.

![Amdahl’s law evaluation: actual performance.](image)

**Amdahl’s Law**

```python
In [49]: parallel = 0.9
serial = 0.1

total_time = df.ix[0, ‘time’]

total_time = [total_time + (serial * parallel / float(n)) for n in np]

In [50]: (fig, ax) = plt.subplots(figsize=(8, 4))

ax.plot(np.arange(1, 13), df[‘speedup’], linewidth=2, label=’serial’)

ax.plot(np.arange(1, 13), df[‘speedup’], linewidth=2, label=’Amdahl’)

ax.set_xlabel(‘number of cores’)

ax.set_ylabel(‘speedup’)

ax.legend(loc=’upper left’)

fig.show()
```

**Speedup and Efficiency**

\[
\text{speedup} = \frac{\text{sequential time}}{\text{parallel time}}
\]

\[
\epsilon = \frac{\text{sequential time}}{\text{number of processors} \times \text{parallel time}}
\]

Measures processor utilization

```python
In [21]: import pandas as pd

df = pd.DataFrame(‘n’: cpu, ‘n’: t)

In [22]: df[‘speedup’] = df[‘t’] / df[‘n’] + df[‘speedup’] * df[‘n’]
```

**Fig. 9.** IPython description and code for illustration of Amdahl’s law.

Part of the goal of exploring synthetic case studies using the Python tools was to allow learners of computer architecture to have more precise understanding of the limits of parallelization. Figure 10 shows the support of using Python control bars in the examples. Specifically the controls are to change the sequential fraction of code and the overhead of coordinating parallel processes. The third control parameters is the number of processors (or processors) that can be assigned to the problem.

![Active learning modification of serial fraction and overhead applied to Amdahl’s law.](image)

**Fig. 10.** Active learning modification of serial fraction and overhead applied to Amdahl’s law.

Figure 11 shows several demonstrated curves using the parameter controls. The visual results allow learners to explore unique combinations of the parameters to generate the resulting performance curves.

![Scalable evaluation of number processors and overhead.](image)

**Fig. 11.** Scalable evaluation of number processors and overhead.

**3) Observing Job Scheduling Tasks across N-Processors:**

Another synthetic study with visualization support is job sectioning. Figure 12 shows the single process execution of multiple jobs and Figure 13 shows when N processors execute the same job mix. The parameters that can be controlled are the number of processors as well as the variation in job mix. The timeline is effective in showing utilization issues when job times are irregular.

**B. Exploring Computer Architecture Experiments with Python**

A significant trend in the broader adoption and study of computer systems has been the use of inexpensive microcontroller platforms such as Arduino and Raspberry Pi platforms.
The Raspberry Pi is a series of small yet powerful computers developed by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools. A core theme of the Raspberry Pi solutions are that with $35, developers have access to a full computing platform with a substantial community of collaborators. Overall, the easily transportable and applicable solution of Raspberry Pi, makes for a great learning environment for students.

The PyCompArch has been deployed to the Raspberry Pi for the purpose of performing application benchmark studies that examine the performance versus high-performance desktop systems. Two example experiments have been successfully completed with the PyCompArch framework: evaluation of impact of frequency scaling on application performance and comparison of two consecutive generations of ARM-based Raspberry Pi platforms. While these are used to study specific Raspberry Pi platform performance, the solutions can also evaluate other platforms.

1) OpenCV on Raspberry Pi for Studying Dynamic Frequency Scaling (DFS): Computer vision is a critically important application workload for modern computers and represents a number of significant algorithms.

To study computer vision performance requires significant investment in system setup. However, with Python-based support, the Open Computer Vision (OpenCV) Library [7] is a way to deploy a full real-world solution. OpenCV makes it easy for students to utilize and has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV has over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a template interface that works seamlessly with STL containers.

One of the most effective ways to reduce power consumption and heat output on a computer system is to use CPUfreq. CPUfreq also referred to as CPU speed scaling allows the clock speed of the processor to be adjusted on the fly. This enables the system to run at a reduced clock speed to save power. The rules for shifting frequencies, whether to a faster or slower clock speed, and when to shift frequencies, are defined by the CPUfreq governor. The governor defines the power characteristics of the system CPU, which in turn affects CPU performance. Each governor has its own unique behavior, purpose, and suitability in terms of workload.

- **Performance governor** - The Performance governor forces the CPU to use the highest possible clock frequency. This frequency will be statically set, and will not change. As such, this particular governor offers no power saving benefit. It is only suitable for hours of heavy workload, and even then only during times wherein the CPU is rarely (or never) idle.

- **On-demand governor** - The On-demand governor is a dynamic governor that allows the CPU to achieve maximum clock frequency when system load is high, and also minimum clock frequency when the system is idle. While this allows the system to adjust power consumption accordingly with respect to system load, it does so at the expense of latency between frequency switching. This is the default governor for the Raspberry Pi.

- **Userspace governor** - The Userspace governor allows userspace programs to set the frequency. This governor is normally used in conjunction with the cpuspeed daemon. Of all the governors, Userspace is the most customizable; and depending on how it is configured, it can offer the best balance between performance and consumption for a system.

Control for these governors on Linux has been support on PyCompArch, and the evaluation of different frequency scalings are set with an experimental evaluation function. Figure 14 shows shows the impact on the execution time when the CPU frequency is changed using the PyCompArch package. The PyCompArch experiment module ran the OpenCV code at a various system frequencies until 70 samples of faces and
then increased the frequency by 100 MHz. The execution time decreases with the increase in clock frequency as expected. Rather than copy results from files to Microsoft Excel or use GnuPlot, the benefit of PyCompArch is to allow students to quickly capture and display the results of their experiments.

2) Benchmarking Raspberry Pi 1.0 versus 2.0: In this section, the PyCompArch module is used to explore OpenCV image processing functions on two ARM-based devices, Raspberry Pi 1.0 and Raspberry Pi 2.0. The Raspberry Pi 1.0B+ is a 700MHz quad-core ARM11 CPU with 512 MB SRAM. The Raspberry Pi 2.0 is a 900MHz quad-core ARM Cortex-A7 CPU with 1GB DDR2 SDRAM and has complete compatibility with Raspberry Pi 1.0. Overall, the reported performance improvement of the Raspberry Pi 2.0 over the Raspberry Pi 1.0 is 6x performance. Using the PyCompArch the results of Figure 15 demonstrated. The execution time evaluates several image processing algorithms that use the OpenCV library built in C/C++. As such the execution is not overly influenced by the Python code that launches each core library function.

3) Exploring Performance Bottlenecks and Tuning: One capability of the PyCompArch module is Code Box that allow developers to examine C code samples. Python does provide support for directly translating small C code samples using SciPy Weave [6] as well as PyCC [4] within the same Python interpretation overhead. While the concept is still being developed, the goal is to provide a pathway for student learners to write small C code examples to test array access patterns and their respective cache performance by using PyCompArch to run the experiments across input sizes and automatically generate the results. From the results, students should be able to understand the performance impact of influencing concepts such as cache misses, loop unrolling, etc. By constructing IPython Notebooks that can serve as starting points for the evaluation, the goal is to provide students with a full testbed for experimenting with computer architecture concepts.

IV. Conclusions and Future Work

The goals of the PyCompArch module is to support computer architecture concepts and to automate the control of computer architecture experiments. The foundation of IPython Notebook environment creates a collaborative opportunity to share assignments, programming problems, and experimental frameworks between developers. For architecture concepts such as Amdahl’s law, the interactive programming environment enables students to modify problem parameters and visualize the results. The support for experiment result collection and graphing in PyCompArch have already been used in multiple course projects to explorations such as OpenCV benchmarking and performance trade-offs with frequency scaling on the Raspberry Pi. In addition, as PyCompArch’s Code Box allows for some compilation of Python strings of C code to compiled to hardware, which creates an interface for comparing performance of C language microbenchmarks and analyzing performance results.

The IPython Notebook environments is becoming generalized into a project called Jupyter [2] with the goal of allowing arbitrary language support. Thus there are more reasons than just Python for the computer architecture education community to develop create learning systems within interactive frameworks. Future work, already in development, will explore the use of Python based models for GPU execution including Open Computer Language (OpenCL) principles. Currently, Python-based Numba [3] supports compilation of Python to run on either CPU or GPU hardware.

REFERENCES