Herding LLaMaS: Using LLMs as an OS Module

Aditya K Kamath*
University of Washington
Seattle, Washington, USA
akkamath@uw.edu

Sujay Yadalam*
University of Wisconsin–Madison
Madison, Wisconsin, USA
sujayyadalam@cs.wisc.edu

1 INTRODUCTION

Computer systems are becoming increasingly heterogeneous with the emergence of new memory technologies and compute devices. GPUs alongside CPUs have become commonplace and CXL is poised to be a mainstay of cloud systems. The operating system is responsible for managing these hardware resources, requiring modification every time a new device is released. Years of research and development are sunk into tuning the OS for high performance with each new heterogeneous device [1–4, 9, 10, 12–14]. With the recent explosion in memory technologies and domain-specific accelerators, it would be beneficial to have an OS that could provide high performance for new devices without significant effort.

We propose LLaMaS which can adapt to new devices easily. LLaMaS uses Large Language Models (LLMs) to extract the useful features of new devices from their textual description and uses these features to make operating system decisions at runtime. Adding support to LLaMaS for a new device is as simple as describing the system and new device properties in plaintext.

LLaMaS reduces the burden on system administrators to enable easy integration of new devices into production systems.

Preliminary evaluation using ChatGPT [11] shows that LLMs are capable of extracting device features from text and make correct OS decisions based on those features.

2 BURDEN OF HETEROGENEITY ON THE OS

The end of Moore's law and Dennard scaling has made the use of heterogeneous systems necessary. Modern high-performance systems are embracing heterogeneity in both memory and compute. These systems combine the best properties of different memory technologies to optimize for latency, bandwidth, capacity, and cost. For processors, the adoption of GPUs and other domain-specific accelerators (DSAs) has helped push the boundaries of compute.

Different applications exhibit different memory requirements, necessitating a diverse set of memory devices to satisfy all of them. A modern HPC system could be connected to local DRAM and NVM, and have disaggregated memory over CXL. Non-volatile memory (NVM) [6] provides high capacities, but experiences read/write asymmetry as well as reduced bandwidth. Similarly, CXL provides greater memory capacity than on-board DRAM, at the expense of increased latencies and lower bandwidth [9].

Data-intensive applications like machine learning or scientific computations require high throughput that is not met by conventional architectures. This has led to the development of accelerators such as GPUs and specialized hardware. In the face of this explosive growth of diverse DSAs each with their own unique API, there has been significant effort being put into unifying application development [7]. The RISC-V group endeavors to provide a unified ISA that can support the unique attributes that these different accelerators require [5]. On the compiler side, the MLIR project [8] is providing an intermediate layer that allows developers to code in their language of choice and then compile the source code into optimized binaries for a chosen processor. In the face of these advancements, we envision a future where an application binary could be deployed on any processing device without programmer input. The operating system (OS) would be tasked with selecting the optimal processing device for the application.

The operating system is the gatekeeper between applications and hardware devices. Beyond providing minimal support for these devices, the OS must be aware of the different intricacies and characteristics under which the devices perform optimally, to remove the reliance on application programmers.

This requirement of OS modification leads to a significant amount of research effort being spent in trying to devise the best method for handling these devices. For example, there has been significant work in page placement for NVM [13, 14] and CXL [9, 10]. In addition, many works have explored techniques for managing data placement and replication for NUMA systems [1, 4]. Similarly, we foresee that significant effort will need to be made in order to allow the OS to select the optimal processing device.

It would be beneficial to have an operating system that could adapt to any heterogeneous system quickly. Such an operating system would reduce the burden of researchers and system administrators. It would also reduce the effort required to integrate new devices into production systems.

3 OUR SYSTEM: HERDING LLAMAS

Our goal is to design an operating system that would be able to (1) adapt to new heterogeneous technologies while (2) requiring minimal intervention from the programmer/system administrator.

To this end, we propose using Large Language Models as an OS Module (LLaMaS) for managing heterogeneous resources and devices (a.k.a., the herd)[1].

Language models are a class of Natural Language Processing algorithms that aim to recognize, summarize, translate, predict and generate text. Large language models or LLMs are models trained on very large datasets with billions to trillions of parameters. OpenAI's GPT-3 has over 100 billion parameters and is trained on almost the entirety of the internet. The recent success of LLMs is attributed to few-shot or zero-shot learning. LLMs can solve various tasks by simply training the models on a few examples (few-shot) or by providing instructions describing the task (zero-shot).

LLaMaS takes advantage of LLMs' ability to perform zero-shot learning. LLaMaS is able to flexibly adapt to new devices with

*Both authors contributed equally to this research.

[1]It is worth noting that for our title "Herding LLaMaS", LLaMaS is responsible for managing the herd, and so is performing the herding, not being herded.

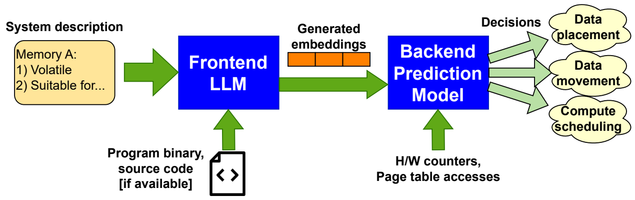

Figure 1: LLaMaS system design.

quirky characteristics, requiring only a simple description of the system and hardware characteristics.

Figure 1 describes how LLaMaS functions. There are two main components of LLaMaS: (1) frontend LLM and (2) backend prediction model. The frontend LLM takes the system description as a text file along with the program binary and/or source code if available. It processes them to identify the important features and patterns that could influence decisions during program execution and generates embeddings that contain these information.

The backend prediction model (BPM) is responsible for making runtime OS decisions. The embeddings generated by the frontend LLM guide the BPM on which heuristics influence decisions and when the decisions need to be made. For instance, the LLM could inform the BPM to migrate data when the accesses per second to a page exceed a certain value.

As an example, the frontend LLM may be informed that a system contains two types of memory, repetitive access tech (RAT) which provides low latency for frequently accessed data but high latency for infrequently accessed data, and DRAM which provides predictable latencies for all data. Through the embeddings, it will inform the BPM that frequently accessed data should be kept in RAT. During an application's runtime, the hardware counters may indicate that data in DRAM is being frequently accessed, causing the BPM to move it to RAT. Through this process, the OS teaches itself how to handle new technology with minimal external support.

4 FEASIBILITY STUDY

To study the feasibility of LLaMaS, we conducted experiments using ChatGPT [11], a chatbot built on top of OpenAI's GPT-3.5 and GPT-4 language models. The goal of the experiments was to understand the abilities of a language model to extract insights from a textual description of device characteristics. The prompts to ChatGPT had 2 components: (1) descrition of a heterogeneous system configuration and device properties, and (2) OS decisions to be made. Below we share the results of 3 such experiments. We first show the prompt to ChatGPT then its response:

Experiment 1: Here, we wanted to ensure that LLMs could understand system descriptions to extract useful features.

Prompt: "System you manage has CPU and GPU. CPU is connected to DRAM and GPU is connected to HBM. For the GPU matrix multiplication program shown below, identify variables that should be allocated on DRAM and those on HBM."

ChatGPT: "Variable h_A is used to store the input matrix on the host, so allocate in host memory. Variable d_A is used to store the input matrix on the device, so allocate in device memory."

ChatGPT was able to successfully understand the system description. Since ChatGPT had been trained on data involving CPUs and GPUs, we didn't have to describe the system to it. It was able to understand the use of different variables and map them to the host or device memory.

Experiment 2: While the above experiment proved that LLMs could work well on data it was trained on, LLaMaS requires that the OS module adapts to new devices with unseen characteristics. Therefore, our second experiment imagined a heterogenous system with a futuristic device.

Prompt: "System you manage has CPU connected to DRAM and Repetitive Access Technology (RAT) memories. RAT is a new memory technology that provides low latency for frequently accessed data. Given below program, identify variables that should be allocated in RAT memory.

ChatGPT: "RAT memory should be used for variables that are accessed repeatedly such as arrays or matrices. DRAM should be used for variables that are accessed randomly and infrequently. These variables should be allocated in RAT: priceArr, avg, ..."

ChatGPT was able to conceptualize the new memory technology and capture its properties. It identified the important features to consider while making memory placement decisions.

Experiment 3: Task placement: Above experiments demonstrated that abilities of LLMs to aid in memory management. We then tried to understand the abilities of LLMs to perform task mapping to different computational platforms.

Prompt: You manage a system with a CPU of 24 cores and a GPU. For best performance, where would you run the following functions?

ChatGPT: Func_A involves matrix multiplication and is highly parallelizable. Therefore, we would prefer to run it on the GPU. On the other hand, Func_B involves pointer manipulation and memory operations, so we would prefer to run on the CPU.

ChatGPT was able to understand functions' goals and properties (parallelizable, memory access patterns) and match it with the properties of the underlying hardware.

REFERENCES

[1] Reto Achermann, Ashish Panwar, Abhishek Bhattacharjee, Timothy Roscoe, and Jayneel Gandhi. 2020. Mitosis: Transparently Self-Replicating Page-Tables for Large-Memory Machines. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 283–300. https://doi.org/10.1145/3373376.3378468

[2] Neha Agarwal, David Nellans, Mark Stephenson, Mike O'Connor, and Stephen W. Keckler. 2015. Page Placement Strategies for GPUs within Heterogeneous Memory Systems. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems* (Istanbul, Turkey) *(ASPLOS '15)*. Association for Computing Machinery, New York, NY, USA, 607–618. https://doi.org/10.1145/2694344.2694381

[3] Rachata Ausavarungnirun, Joshua Landgraf, Vance Miller, Saugata Ghose, Jayneel Gandhi, Christopher J. Rossbach, and Onur Mutlu. 2017. Mosaic: A GPU Memory

Manager with Application-Transparent Support for Multiple Page Sizes. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (Cambridge, Massachusetts) *(MICRO-50 '17)*. Association for Computing Machinery, New York, NY, USA, 136–150. https://doi.org/10.1145/3123939.3123975

[4] Mohammad Dashti, Alexandra Fedorova, Justin Funston, Fabien Gaud, Renaud Lachaize, Baptiste Lepers, Vivien Quema, and Mark Roth. 2013. Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Houston, Texas, USA) *(ASPLOS '13)*. Association for Computing Machinery, New York, NY, USA, 381–394. https://doi.org/10.1145/2451116.2451157

[5] Jamie Feller. 2018. SiFive Core IP 7 Series Creates New Class of Embedded Intelligent Devices Powered by RISC-V. https://www.sifive.com/press/sifive-core-ip-7-series-creates-new-class-of-embedded.

[6] Intel. 2019. Intel® Optane™ memory - revolutionary memory: What is optane memory? https://www.intel.com/content/www/us/en/products/details/memory-storage/optane-memory.html.

[7] Chris Lattner. 2021. *The Golden Age of Compiler Design in an Era of HW/SW Co-design*. Youtube. https://www.youtube.com/watch?v=4HgShra-KnY ASPLOS '21 Keynote.

[8] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2–14. https://doi.org/10.1109/CGO51591.2021.9370308

[9] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 574–587. https://doi.org/10.1145/3575693.3578835

[10] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 742–755. https://doi.org/10.1145/3582016.3582063

[11] OpenAI. 2022. Introducing ChatGPT. https://openai.com/blog/chatgpt.

[12] Ashish Panwar, Reto Achermann, Arkaprava Basu, Abhishek Bhattacharjee, K. Gopinath, and Jayneel Gandhi. 2021. Fast Local Page-Tables for Virtualized NUMA Servers with VMitosis. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 194–210. https://doi.org/10.1145/3445814.3446709

[13] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. 2021. HeMem: Scalable Tiered Memory Management for Big Data Applications and Real NVM. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 392–407. https://doi.org/10.1145/3477132.3483550

[14] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. 2019. Nimble Page Management for Tiered Memory Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 331–345. https://doi.org/10.1145/3297858.3304024