Project Handout for EECS 570: Parallel Computer Architecture

Yatin A. Manerkar

January 21, 2022

1 Important Dates

The following are the important dates for this project:

Proposal Due: 11:59 PM, 31st Jan., via Canvas
1-Page Milestone Report Due: 11:59 PM, 20th March, via Canvas
Milestone Meetings: Week of 21st March
Final report Due: 11:59 PM, 19th April, via Canvas
Poster session: Week of 18th April, time and place TBA

2 Introduction

In this project, you will innovate new designs or evaluate extensions to designs presented in class or a recent architecture/systems conference. **The project must be broadly related to multiprocessor systems.** The purpose of this project is to conduct and generate publication-quality research and to improve the state of the art. The project report will be in the form of a 6-page research article similar to those we have been discussing in the class. Project groups will include 4-5 students, and work expectations will be calibrated accordingly. Groups will present their work in a final poster session.

The project will account for 25% of your final course grade. It will be graded out of 100 points.

- Proposal and milestone report (5 points)
- Final report (75 points)
  - Problem definition and motivation (15 points)
  - Survey of previous related work (15 points)
  - Description of design (15 points)
  - Experimentation methodology (15 points)
  - Analysis of results (15 points)
  - Statement of contribution of each team member
- Poster presentation (20 points)

3 Proposal

The proposal is a written 1-2 page (2 pages is a hard limit) document including:

1. Clearly state your group name and members.
2. A problem definition and motivation.
3. A brief survey of related work with at least four papers you have found and read directly related to the topic. Most of the project ideas below have links to at least one relevant paper to get you started. You can look at that paper's citations as well as papers which cited it to find more work in that area. Google Scholar and DBLP are your friends. Conferences that cover multiprocessor system research include ISCA, ASPLOS, MICRO, HPCA, PACT, ICS, SC, ICPP, IPDPS, SOSP, OSDI, SIGMETRICS. You will also find the World Wide Computer Architecture web page `www.cs.wisc.edu/~arch/www` to be a good source of information.
4. A detailed description of your experimental setup including the infrastructure you will use and any significant development effort you must invest to build new infrastructure.
5. Project milestones and schedule. What do you expect to see by the milestone meeting? Where do you plan to go based on your observations? You can draw a flow chart to clarify.
6. Division of labor. How is each group member going to contribute to the project?

I will read through your proposal and offer feedback. I may ask you to increase or decrease the scope of your project so as to have it fit within the expectations of a course project for EECS 570.

4 Milestone Report and Meeting

You will hand in a one-page (this is a hard limit) write-up describing your progress and initial results. These results form the basis for the final outcome of your research. Based on these results, explain your plans for the rest of the semester. If there are any changes to plans, you should bring them up in this report.

Each group must also make an appointment to meet with me during the week of March 21st (i.e. the week after the Milestone reports are due). A link for making the appointments will be posted on Canvas.

For the meeting, you must prepare graphs or figures of your experimental results to date and discuss them with me in the status meeting. You should paste each graph/chart into a PowerPoint slide with 1-2 bullets summarizing the main conclusion of the experiment. These graphs should form the basis of your final report and poster.

5 Final Report

You will submit a 6-page final report in the format of conference papers like those we are reviewing (see ISLPED for an example of what 6-page 2-column papers typically look like). The report should include an abstract, an introduction, followed by a detailed description of the design, a methodology section, a results section, and a conclusion section. You may choose to write a related work section preceding the conclusions, or a background section following the introduction. Finally, you must include a brief statement of each team member's specific contribution to the project. You will need all the relevant citations in a reference section at the end. (Other outlines for your report may be appropriate, but if doing something different, you must check with me to make sure you are doing something sensible first.)

6 Posters

During the final week of lectures, we will hold a poster session, open to the campus community, in which teams will get to present their results. The GSIs and I will review your posters and pose questions related to your project to team members.

7 Best Projects

Reports should be sufficiently polished so that they could be submitted to a high-quality workshop without further modification. I anticipate that the best projects will be submitted to computer system conferences/workshops for publication.

8 Infrastructure

For projects that propose hardware modifications (e.g., to CPUs or the memory system), the gem5 full- system timing-accurate simulation infrastructure is the preferred evaluation tool. We currently have the PARSEC benchmarks in gem5 as well as a few other workloads. For GPU projects, you may wish to use gpgpusim. The servers and accelerators in Typhon and Typhon2 are also available for the projects. The BookSim network-on-chip simulator is also available for projects that will focus on interconnect design. Finally, you can use the Pin binary instrumentation tool. If you are interested in writing RTL for custom hardware designs and running it on FPGAs on the cloud, Amazon's AWS provides new users with access to their EC2 F1 instances.

9 Research Project Topics

Some of the project topics below are undisclosed ideas and are confidential. Please do not distribute this list.

I will be open to ideas you may have for a research project if you can convince me that it is connected to one of the course themes and that it is worth pursuing. **If this is your intention, please contact me before you submit your proposal to discuss.**

Here is a list of suggested projects.

9.1 Interactive Simulators

Build an interactive web tool that helps teach a concept in parallel computer architecture. Some of the key 570 concepts that you may want to consider for this project:

1. Cache Coherence protocols: MSI, MESI, MOESI.
2. Memory consistency models: SC, TSO, RMO, DRF0
3. Network-on-chip: different topologies, routing algorithms
4. Synchronization errors: Data-races and deadlocks

You will develop a web-front end, and a back-end that simulates the behavior of the above functionalities. Each team can pick one of the above functionalities. Such tools have been built for caches and 5-stage pipelines. These tools are now widely used by our students in EECS 370, and they find it very helpful.

`http://vhosts.eecs.umich.edu/370simulators/pipeline/simulator.html`
`http://vhosts.eecs.umich.edu/370simulators/cache/simulator.html`

9.2 Genome Sequencing Applications

Over the last decade there has been dramatic improvement in sequencer technology. Cost for assembling whole human genome, for example, has reduced from over $1 million to less than $1000. Portable sequencers like ONT's minION now make it possible to sequence in the field, which helps discover new organisms, detect and monitor the spread of infectious diseases.

There are a variety of bioinformatics tools available for analyzing sequencing data. While they are highly data-parallel, many of them are irregular, which makes it challenging to use accelerators like GPUs.

The goal of this project would be to identify a bioinformatics software that performs poorly on CPUs. Accelerate it using GPUs, or custom hardware accelerators. A few applications that you may want to consider are: Guppy basecaller, BWA-MEM for read alignment, metagenomic analysis, variant calling, de novo assembly, proteomics, liquid bioposy, etc.

Prior work at Michigan:

https://ieeexplore.ieee.org/document/8416819 https://github.com/bwa-mem2/bwa-mem2 (ERT)

ONT Sequencers:

https://nanoporetech.com/products/minion https://github.com/nanoporetech

9.3 Low-Overhead Memory Access Profiling

Memory access patterns offer important insights into program behavior. For example, one can infer memory regions that are hot and cold from such access patterns, which can then even be correlated to data structures to gauge which data structures are frequently accessed. However, profiling programs to obtain their memory access patterns is expensive. Tools like Intel's Pin are still too expensive (usually about 0.5-2× slower) for them to be employed in a production environment. The aim of this project is to build a sampling memory access profiler that outputs a suitable sample of memory accesses from which program properties can be inferred. The profiler should be able to run with very low overhead (ideally <1%). As a second step, you can try to design a scheme that automatically tunes the sampling frequency to be under a given maximum performance overhead.

- Generating Miss Rate Curves with Low Overhead Using Existing Hardware

  https://pdfs.semanticscholar.org/3987/7cd79f45036d507ec2ad221c8306c2c0611b.pdf

- Computer Performance Microscopy with SHIM

  https://www.microsoft.com/en-us/research/publication/computer-performance-microscopywi

- Advanced Hardware Profiling and Sampling (PEBS, IBS, etc.): Creating a New PAPI Sampling Interface

  http://web.eece.maine.edu/~vweaver/projects/perf_events/sampling/pebs_ibs_sampling.pdf

9.4 Designing Scalable Synchronization Primitives for Many-core Systems

Modern libraries provide efficient synchronization for different situations. For example, MCS locks provide maximum throughput to contended critical sections, while pthreads/Linux futex's ensures that threads never wait on mutexes held by a sleeping thread (rather, the sleeping thread is immediately scheduled so the critical section makes forward progress). At the same time, MCS locks do not scale as the number of threads exceeds the number of CPUs and pthreads performs poorly under contention. The need to choose the correct synchronization implementation complicates program design. Recent work ("Decoupling Contention Management from Scheduling" by Johnson) proposes a new lock primitive, based on preemptable MCS locks that provides the best of both

worlds: critical section throughput is maximized for various numbers of threads both with and without contention.

Design new lock and synchronization primitives that perform well even with more threads than CPUs. Consider primitives besides mutex (pthreads condition variables are particularly bad), and primitives that can be implemented entirely in user-space (e.g., by leveraging pthreads inside your synchronization primitives).

9.5 Reconfigurable Many-Core Architecture for Specialized Workloads

The demise of Moore's Law has led to renewed interest in accelerators to improve performance and reduce energy and cost. In order to address these issues for applications involving sparse matrix computations, we propose a custom accelerator, OuterSPACE, that consists of asynchronous Single Program, Multiple Data (SPMD)-style processing units with dynamically-reconfigurable non-coherent caches and crossbars. OuterSPACE is designed to work with the unconventional outer product based matrix multiplication approach, which involves multiplying the ith column of the first matrix (A) with the ith row of the second matrix (B), for all i. An important next step would be to extend this architecture to support dense matrix multiplication, stochastic gradient descent, graph-search algorithms, recurrent neural networks, long short-term memories, etc.

`https://drive.google.com/file/d/1-E72T4rk1qaHKb664PDIeZPdrn6Q-OBX/view?usp=sharing`

9.6 Formal Verification of CPU or GPU Coherence Protocols

In programming assignment 2, you will be verifying a standard MSI protocol using a formal verification tool called Murphi. The involvement of transient states in addition to the stable states makes the coherence state machine complicated. This directly translates to more rigorous verification effort to ensure correctness under all conditions. The need for high-performance and scalable machines has made cache coherence protocols much more complex than simple MSI. As faster, larger systems are designed and built, the complexity of cache protocols will continue to increase. For this project, you need to implement and formally verify an unconventional CPU or GPU coherence protocol and optimize with advanced features such as speculative requesting, cruise missile invalidations, migratory sharing optimizations, etc.

`http://ieeexplore.ieee.org/abstract/document/6522351/`
`http://ieeexplore.ieee.org/abstract/document/7851544/`

9.7 Near/In-Memory Computation to Accelerate Specialized Workloads

Emerging 3D stacked memory systems provide significantly more bandwidth than current DDR modules. However, general purpose processors do not take full advantage of these resources offered by the memory modules. Taking advantage of the increased bandwidth requires the use of specialized processing units. Some targeted applications that can generate memory requests more frequently are large-scale graph analytics, sparse matrix multiplication, data mining, computational biology, multimedia, speech recognition and financial modeling. You will implement a near/in-memory accelerator targeted at one such bandwidth-bottlenecked application for this task.

`http://ieeexplore.ieee.org/abstract/document/6846276/`
`http://ieeexplore.ieee.org/abstract/document/7284059/`
`http://ieeexplore.ieee.org/abstract/document/6670336/`

9.8 Compiler-Based Approach for Parallelizing Irregular Programs

Design compiler-based static or dynamic techniques that would identify sections of a program that can be parallelized across multiple threads. One approach would be to target loops that do not have irregular memory accesses and have limited loop carried dependencies as implemented in `http://dl.acm.org/citation.cfm?id=2259028`. In this project, you can implement a LLVM compiler pass that can inspect the control-flow graph and identify parallelizable sections of the code.

9.9 Extend a Machine Learning Accelerator Design

Extensive research is being conducted on the hardware front to accelerate machine learning workloads for various applications, ranging from embedded systems to data centers to business analytics. Although GPUs are highly efficient for DNN training and thus are predominant in the industry, FPGAs have recently gained traction in this area. The task would be to extend an existing design, based on literature survey, to accelerate machine learning kernels.

`http://www.morganclaypool.com/doi/10.2200/S00783ED1V01Y201706CAC041`

9.10 Data Logging for Autonomous Vehicle Training

The recent boom in interest in self-driving vehicles has been fueled by advancements in the fields of machine learning and artificial intelligence. This has given rise to to the complication of storing and processing the large amounts of data generated by sensors that are mounted on the autonomous vehicle. The task here is to develop algorithms to efficiently compress sensor data generated through autonomous vehicle simulators.

9.11 FPGA Implementation of Accelerators for Specialized Applications

Increasing demand for power-efficient, high performance computing has spurred a growing number and diversity of hardware accelerators in mobile and server Systems on Chip (SoCs). Furthermore, FPGAs represent an important way to integrate accelerators on an SoC, since they can be reprogrammed as needs change and allow multiple accelerators to share optimized communication infrastructure. In this project, you may choose to implement an accelerator for genome sequencing, intrusion detection, SMT solvers or other emerging applications using the Amazon EC2 F1 FPGA instances.

9.12 Hardware Acceleration of Blockchain Applications

Recent years have seen the emergence of a new class of currencies, called cryptocurrencies. These currencies use cryptography to provide security and peer-to-peer networking to provide a decentralized system. Bitcoin is the most popular of these currencies. It uses a two-pass SHA-256 hash at its core. Producing new bitcoins is done through a process referred to as *mining*, which involves a brute-force search for a hash with a specific value. This process requires large amounts of computing power. The task is to implement an accelerator using FPGAs to accelerate the underlying blockchain application used to mine bitcoins. You may use simulation tools or Amazon EC2 F1 FPGA instances for this.

9.13 Characterization on Contemporary Architectures

- **Part 1: Characterize Workloads**

  New applications and programming paradigms continue to emerge rapidly as the diversity and performance of computers increase. Whether they are in smartphones and deeply embedded systems at the low end or in massively parallel systems at the high end, the design of future computing machines can be significantly improved if we understand the characteristics of the workloads that are expected to run on them. In this part of the project, you are expected to select one or more applications and profile them on contemporary parallel architectures such as CPUs and GPUs.

- **Part 2: Measure and Analyze the Power/Energy Consumption of a Mobile Device**

  Limiting power and energy consumption presents a critical issue in computing, particularly in portable and mobile platforms such as laptop computers and cell phones. Uneven energy drain is noticed to be increased when running applications that require intensive computations on the mobile devices. For this part you are to profile and analyze power/energy consumption patterns for varying applications run on a device with parallel processors.

9.14 Investigate Parallel Implementations of malloc

Many modern programs frequently use dynamic memory allocation. However, modern programs increasingly are multithreaded and parallel to take advantage of increasingly parallel processors. Unfortunately, this trend conflicts with the fact that there is a single heap in most current programs. Consequently, research into parallel memory allocators is topical and important. You are tasked with innovating over an existing malloc implementation and accelerate it on a CPU or an FPGA.

9.15 Optimizing Parallelization of 3-D Ultrasound Beamforming

Ultrasound beamforming is a highly parallel, memory dominated workload. One of the key challenges in speeding up this workload for modern hardware is determining how to parallelize while maximizing data locality. For this project you can take a serial beamforming implementation and undertake the task of exploring multiple parallelization strategies such as multi-threading, SIMD, GPU, etc. Additionally you will need to determine the best method of dividing the inherent parallelism in the algorithm such as across transducer channels or across image scanlines to maximize efficiency. You may also want to examine how your solution changes across hardware configuration to develop a more robust solution that is not tailored to a specific memory system size or core number.

`http://ieeexplore.ieee.org/document/6522329/`

9.16 Secure Parallel Architectures

Significant security vulnerabilities have been recently exposed in modern hardware such as cold boot attacks, Rowhammer, Spectre, Meltdown. In addition to these attacks, a multiprocessor system can concurrently execute different applications, which share hardware resources (e.g., shared caches, interconnect, memory, etc.). Such sharing can lead to side-channels that compromise isolation. You can try to replicate known side-channels, study new vulnerabilities, design and evaluate defenses. You may want to think about secure GPU architectures, which are relatively under-explored.

A related but different problem is to guarantee privacy for computation executing on a third-party system (e.g., public cloud) where you cannot trust even the system software (e.g., cloud OS). This is important for application developers that would like to use cloud (banks, hospitals, defense, etc.) but are wary of computing on sensitive private data on the cloud. Modern processors provide specialized hardware support in the form trusted hardware execution environments (e.g., Intel SGX). You can identify a parallel application that operates on sensitive private data (e.g., analyzing genome sequencing data) and use Intel SGX to off-load sensitive parts of computation to the secure hardware. Alternatively, you can study the challenges and solutions in designing secure hardware such as Intel SGX.

References:

- Chapter 7 on "Multiprocessor and Many-Core Protections" in "Principles of Secure Processor Architecture Design". J. Szefer. Morgan and Claypool Synthesis Lecture.
- Spectre Attacks: Exploiting Speculative Execution. Kocher et al. Oakland 2019.
- InvisiPage: oblivious demand paging for secure enclaves. ISCA 2019. Aga and Narayanasamy.
- InvisiMem: Smart Memory Defenses for Memory Bus Side Channel. ISCA 2017. Aga and Narayanasamy.
- Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. Bulck et al. Usenix Security. 2018

9.17 Verify the Memory Consistency of an RTL Implementation with RTLCheck

There is a need to verify that RTL implementations of processors conform to their memory consistency model (MCM). RTLCheck (`https://ieeexplore.ieee.org/document/8686596`) is a methodology and tool for conducting formal MCM verification of RTL implementations for litmus test programs (small 4-8 instruction programs used to test specific MCM scenarios). In this project, you will use RTLCheck to formally verify the Verilog implementation of a parallel processor for conformance with an MCM.

9.18 Verify the Memory Consistency of a Microarchitectural Design with PipeCheck

PipeCheck (`https://ieeexplore.ieee.org/document/7011423`, `https://dl.acm.org/doi/10.1145/2872362.2872399`) is a methodology and tool for conducting formal MCM verification of microarchitectural ordering specifications against their consistency model for litmus tests. In this project you will use PipeCheck (specifically its COATCheck incarnation: `https://github.com/daniellustig/coatcheck`) to verify a complex parallel microarchitectural design against its ISA-level consistency model.

This tutorial (`https://check.cs.princeton.edu/tutorial.html`) can help you learn how PipeCheck works.

9.19 Verify the Hardware Security of a Microarchitectural Design (or synthesise exploits for it) using CheckMate

CheckMate (`https://ieeexplore.ieee.org/document/8574598`) is a methodology and tool for formally verifying hardware security properties on microarchi-

tectural designs up to a bound. If the verification fails, CheckMate synthesises an exploit program showing the security vulnerability. In this project you will use CheckMate to verify hardware security properties on a complex parallel microarchitectural design.

This tutorial (`https://check.cs.princeton.edu/tutorial.html`) can help you learn how CheckMate works.

9.20 Hardware-Accelerated Garbage Collection

Garbage collection is a key component of certain programming languages that have the capability to manage memory for programmers (e.g., Java, OCaml). Hardware acceleration of such a routine could significantly improve the performance of programs written in such languages. In this project, you will develop and evaluate a hardware accelerator for the garbage collection routine in a programming language that uses garbage collection.

`https://ieeexplore.ieee.org/document/8416824`