EECS 570: Parallel Computer Architecture
Project Handout F26

Sep 18, 2026

1 Important Dates

- **Proposal:** Due October 7 via Canvas.
- **Group Sign-up:** Via Google Sheet.
- **Project milestone:** Due Nov 12th.
- **Poster presentations:** December 9, 1:30–4:00 PM; location TBA.
- **Final report:** Deadline tentatively Dec 13th, via Canvas.

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, multicore systems, GPUs, or other parallel architectures. 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 class. Project groups will include 4–5 students. Groups will present their work in front of the class during the last few lecture and discussion sessions.
The project will account for **25% of your final course grade**. It will be graded out of 100 points.

- Proposal and milestone reports (5 points)
- Final report (15 points)
  - Problem definition and motivation (3 points)
  - Survey of previous related work (3 points)
  - Description of design (3 points)
  - Experimentation methodology (3 points)
  - Analysis of results (3 points)
  - Statement of contribution of each team member
- Presentation (15 points)
- Assessment of problem, idea, and execution (research quality). (65 points)

3 Proposal

You are encouraged to meet with the teaching team prior to submitting the proposal if you have any questions. When in doubt, meet with us. Please make an appointment or use the office hours.
The proposal is a written two-page (at most) document including:

1. A problem definition and motivation.
2. A brief survey of related work with at least four papers you have found and read directly related to the topic (ask me or the GSI for pointers or post on Piazza). Conferences that cover multiprocessor system research include ISCA, ASPLOS, MICRO, HPCA, PACT, ICS, SC, ICPP, IPDPS, SOSP, OSDI, and SIGMETRICS. You can search the online pages and/or contact me for pointers. You will also find the World Wide Computer Architecture web page `www.cs.wisc.edu/~arch/www` a good source of information.
3. 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.
4. Project milestones and schedule. What do you expect to see by each milestone? Where do you plan to go based on your observations? You can draw a flow chart to clarify.
5. Division of labor. How is each group member going to contribute to the project?

4 Milestone Report

You will hand in 5–10 slides describing your progress: problem context/motivation, solution, tools, and initial results. These results form the basis for the final outcome of your research. You will make an appointment to meet with me to go over the project status. The appointments are made by filling out an appointment sheet online prior to the meetings.

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 check with me to make sure you are doing something sensible first.)

6 Poster Presentation

In the last lecture, we will have a poster session where each team will present their work. The audience and the teaching team will ask questions about your work. You will be graded on the quality of your presentation and your ability to answer questions about your work. You should aim to present your work to anyone at varying degrees of depth. Prepare a 2-minute elevator pitch, a 5-minute overview, and then dive into any part of your project based on questions asked. You will be required to submit a PDF version of the poster after the presentation to Canvas.
The physical poster should be 30 inches x 40 inches and can be printed using UM ITS or a private service such as FedEx. The UM printing service should be covered by your printing allocation, but note that they get busy near the end of the semester and may have long turnaround times.

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 systems conferences/workshops for publication (several of the papers on our reading list began as course research projects).

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.
SniperSim is another cycle-accurate simulator that you may use. It is less complicated than gem5, therefore less accurate but easier to use and still proficient for evaluating most hardware modifications. It comes with SPEC and PARSEC benchmarks.
If your project involves changing the memory system, DRAMsim3 is a good simulator for memory performance.
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. Pinplay is the advanced version of Pinplay that provides much functionality for tracing. If you are using SniperSim, you might wish to check out LoopPoint.
CPU profiling tools such as VTune and perf are a good starting point if you work on an acceleration project, and Nsight tools are useful for GPU workloads.
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 its EC2 F1 instances.
If you need GPU resources, we can provide some via the Great Lakes system. Please indicate that in your proposal.

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 any ideas you may have for a research project if you can convince me it is connected to one of the course themes and that it is worth pursuing. You are welcome to coordinate your project in this course with projects in other courses or ongoing research. Here is a list of suggested projects.

9.1 Democratizing AI Kernel Optimizations for TPUs

The ML "Software Lottery." The advancement of AI depends on specialized hardware like GPUs and TPUs. However, achieving optimal performance has become a "performance lottery," accessible primarily to teams capable of developing custom, low-level kernels for specific hardware. This requires a rare combination of cross-stack expertise in algorithms, parallel programming, and hardware architecture, creating a significant innovation bottleneck. This challenge is particularly acute for critical operations like attention, where efficient, fused kernels are notoriously difficult to generate using traditional compilers.
A Proven Blueprint. Joy Dong (UM PhD student) led FlexAttention on GPUs. The solution is to bridge the gap by combining the performance of expert-tuned kernels (e.g., FlashAttention) with the flexibility of just-in-time (JIT) compilation, delivered through a high-level, customizable abstraction. The viability of this compiler-driven approach is demonstrated by FlexAttention, a programming

model in PyTorch that allows researchers to express novel attention mechanisms in high-level Python. FlexAttention provides simple yet powerful callables (score mod, mask mod) that are JIT-compiled via `torch.compile` into a single, fused kernel, achieving performance competitive with hand-tuned CUDA code. By separating algorithmic intent (what) from implementation (how), FlexAttention has successfully democratized attention research on GPUs. Its wide adoption—evidenced by integration into core libraries like HuggingFace Transformers and citation in over 50 research papers—proves the efficacy of this model.

Project ideas:

Flexible Kernels on TPUs. Develop PL/compiler optimizations for TPUs. Unlike general-purpose GPUs, TPUs are purpose-built for ML tasks, yielding significant advantages in performance and power efficiency. However, this hardware specialization translates to a less mature software ecosystem and a scarcity of experts capable of optimizing kernels for these architectures, presenting a steeper learning curve and higher integration barriers for the broader research community. Developing raw OpenXLA code is complex, as it requires manual memory management and deep architectural knowledge.

An initial goal could be to build a flexible attention abstraction for the JAX-OpenXLA ecosystem. For key algorithmic classes, such as scaled-dot-product attention and its variants, a single, expert-tuned template can be broadcast through our proposed JAX abstraction, making high performance widely accessible.

Tools and Infrastructure: This work would use the JAX and OpenXLA ecosystem by introducing a novel framework that bridges high-level expressivity with low-level performance. Your solution could consist of one or more of these synergistic components:

- **A High-Level JAX API:** A user-facing function, `flex_attention_tpu`, specialized for primitives like attention. Users will define custom algorithmic logic in JAX, without considering hardware scheduling details.
- **A Library of Expert-Tuned Pallas Templates:** A backend repository of parameterized, performance-optimized Pallas kernels. These templates will be developed by experts and will encapsulate best practices for TPU performance, such as memory pipelining, hardware-aware tiling for the MXU, and automatic padding to handle hardware constraints.
- **A JIT Compiler Bridge:** A custom lowering rule within `jax.jit` that intelligently connects the user's API call to the appropriate Pallas template. The compiler will analyze the user's code, select the best template, specialize it with the correct parameters, and hand it to the XLA compiler for final code generation.

9.2 Machine Learning Accelerator

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 a literature survey, to accelerate machine learning kernels.

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

You can think of customizing an ML accelerator for different use cases: edge, training, robotics, agentic AI, etc.

9.3 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). In this project, you may choose to implement an accelerator for robotics, genome sequencing, intrusion detection, SMT solvers, garbage collection, or other emerging applications. You may choose to implement and

evaluate your accelerator in Verilog or simulate it using a simulator like gem5 or a C++ simulator like SniperSim.
https://dl.acm.org/doi/abs/10.1145/3445814.3446746
https://dl.acm.org/doi/10.1145/3579371.3589060

9.4 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.
Samsung simulator: https://github.com/SAITPublic/PIMSimulator
Recent paper from UM: https://dl.acm.org/doi/10.1145/3676641.3716267
http://ieeexplore.ieee.org/abstract/document/6846276/
http://ieeexplore.ieee.org/abstract/document/7284059/
http://ieeexplore.ieee.org/abstract/document/6670336/

9.5 Exploring Tensor Core Utilization and Mixed Precision

Goal: Study how effectively NVIDIA Tensor Cores can be applied beyond deep learning.
Tasks:

- Select a kernel (e.g., matrix multiplication, FFT butterflies, polynomial multiplication, or modular arithmetic). One of the teams in recent semesters showed that you can rewrite computation in a Boids simulation using matrix multiplication so that it can exploit tensor cores.
- Implement variants using FP16, BF16, TF32, and FP64 accumulation modes.
- Compare throughput, accuracy, and energy efficiency across modes.

Open choices for students: Kernel type, precision mode, and baseline versus Tensor Core implementations.
Deliverables: A comparative report showing performance-versus-accuracy tradeoffs across precisions.

9.6 Dynamic Programming Acceleration with NVIDIA DPX Instructions

Goal: Evaluate the new DPX instructions (Hopper/Blackwell) for accelerating dynamic programming workloads.
Tasks:

- Implement a DP algorithm such as Smith–Waterman, edit distance, or Viterbi decoding.
- Develop two versions: one with standard CUDA cores and one with DPX intrinsics.
- Measure speedup, occupancy, and memory access efficiency.

Open choices for students: Algorithm selection, dataset, and baseline implementation style.
Deliverables: Benchmark results and analysis of whether DPX shifts the compute-versus-memory balance.

9.7 GPU Memory Hierarchy Experiments with Async Copy and Shared Memory

Goal: Explore NVIDIA's `cp.async` and cooperative groups for hiding global memory latency.
Tasks:

- Implement a tiled kernel such as matrix multiplication, stencil, or reduction.
- Compare naive global memory access against async copy into shared memory.
- Evaluate overlap of compute and memory, and quantify speedup.

Open choices for students: Kernel type, tiling strategy, and problem size.
Deliverables: Roofline analysis showing how async copy changes arithmetic intensity.

9.8 Thread Block Clusters and Multi-SM Scalability

Goal: Investigate NVIDIA Hopper's "thread block clusters" for workloads with inter-block communication.
Tasks:

- Implement a workload with cross-block communication (e.g., prefix sum, FFT stages, or graph traversal).
- Compare performance with and without clusters enabled.
- Study scaling behavior as problem size increases.

Open choices for students: Algorithm selection, communication strategy, and problem size.
Deliverables: Strong and weak scaling plots with analysis of programmability and efficiency tradeoffs.

9.9 Hardware Acceleration for Open-Source Genome Sequencing Pipelines

Goal: Explore how hardware-level optimizations and accelerator design can speed up widely used open-source genome sequencing tools (e.g., BWA-MEM, Minimap2, SPAdes, GATK, Canu/Flye).
Background: Genome sequencing involves compute-heavy stages such as read mapping, sequence alignment, k-mer counting, graph assembly, and variant calling. Open-source software often becomes the bottleneck due to software overheads, memory bandwidth limits, and irregular data structures. Specialized hardware—GPUs (Tensor Cores, DPX instructions), FPGAs, NPUs, or ASIC-style accelerators—can provide significant speedups if the mapping is done thoughtfully.
Possible directions (students choose one or combine):

- **GPU-oriented acceleration:** Use recent GPU features (e.g., Hopper/Blackwell DPX instructions, async copy, or Tensor Cores) to accelerate dynamic programming alignment or variant calling kernels. Analyze tradeoffs between compute-bound and memory-bound stages in tools like Minimap2 or GATK.
- **ASIC/NPU-level acceleration:** Design a systolic array accelerator for sequence alignment (Smith–Waterman, Pair-HMM). Propose a specialized k-mer counting engine with on-chip SRAM buffers to reduce memory bottlenecks. Model throughput/energy tradeoffs for mapping genome assembly graph construction to custom hardware.
- **System-level integration:** Consider how such accelerators would fit into an end-to-end genomics pipeline (I/O bottlenecks, streaming reads from storage, and overlapping compute and communication). Evaluate whether heterogeneity (CPU + GPU + DPU/NPU) can reduce bottlenecks in real-world pipelines.

Deliverables:

- A performance profile of an open-source tool identifying key hardware bottlenecks.
- A hardware-level prototype or design exploration (GPU kernel using new instructions, FPGA/NPU mock-up, or analytical model of an ASIC accelerator).
- A report analyzing expected speedup, energy efficiency, and integration challenges.

9.10 Modeling Confidential Computing on GPU

In 2023, NVIDIA introduced confidential computing extensions (`https://www.nvidia.com/en-us/data-center/solutions/confidential-computing/`) to its Hopper-generation GPU to secure data and AI models deployed on GPGPUs. It protects sensitive data and proprietary AI models from unauthorized access using strong hardware-based security even when they are deployed on untrusted third-party platforms such as public clouds.
The added security comes at the cost of additional cryptographic checks during data transfers and memory accesses, introducing memory access performance overhead. For this project, you need to implement a performance model of confidential computing on GPUs and analyze its performance on various workloads to help optimize confidential computing on GPUs. You can start from GPGPU-Sim (`http://www.gpgpu-sim.org/`), an open-source cycle-accurate GPU simulator.

9.11 Modeling Multi-Core Processor with Intel SGX

Intel's Software Guard Extensions (SGX) (`https://eprint.iacr.org/2016/086.pdf`) is a set of extensions to the Intel architecture that aims to provide secure remote computation (e.g., on a public cloud) by leveraging trusted hardware. It provides confidentiality and integrity guarantees on sensitive computations consuming confidential data performed on a computer where privileged software (kernel, hypervisor, etc.) and the hardware platform (the adversary can probe off-chip channels such as DDR and PCIe) are potentially malicious.
Cryptographic protection of memory is an essential ingredient to allow software to run in a trustworthy manner and handle secrets while its external memory is susceptible to eavesdropping and tampering. The Memory Encryption Engine (MEE, `https://eprint.iacr.org/2016/204`) in the Intel SGX extension provides that guarantee but at non-negligible performance overhead. As the main memory size grows in modern platforms, there is increasing need to understand this overhead and how it scales as the memory size grows.
For this project, you need to implement a performance model of Intel SGX and analyze its performance on various workloads and how it scales with protected memory size. A cycle-accurate multicore simulator, SniperSim (`https://snipersim.org//w/The_Sniper_Multi-Core_Simulator`), is a good base simulator to start with.

9.12 Limited Memory Algorithms for Trusted Hardware Applications

In the past decade, data analytics have gained massive popularity. One area where this has gained traction is in the health sector, where patient records can be analyzed to better understand disease, genetics, etc. Working with highly sensitive data requires privacy-respecting technologies, one of which is trusted hardware like the commercially available Intel SGX (`https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html`). While trusted hardware has much promise, one of the technical challenges to overcome is its limited memory capacity (128–256 MB for SGX). This project requires students to take a target data analytics kernel and create an algorithm and protocol that works with this limited memory hardware.
Example kernels: Principal Component Analysis (`https://builtin.com/data-science/step-step-explanation-principal-component-analysis`) and Support Vector Machine (`https://www.`

analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/).
Groups interested in this project may reach out to jonaher@umich.edu to discuss options.

9.13 Optimizing Parallelization of SAT Solvers

This project can be done based on GPU, with SIMD optimization, or on a new ASIC hardware design. In project 2, you will use a formal verification tool called Murphi, which verifies that your coherence protocol does not violate any rules you set and does not cause a deadlock. This verification process is slow, especially as the protocol and rules get more complicated. Underneath the surface, Murphi is solving a Boolean Satisfiability Problem (SAT). There are different techniques to parallelize this NP-hard problem to speed up the formal verification process.
https://pubs.aip.org/aip/acp/article/1836/1/020059/586346/A-survey-of-SAT-solver
Also check out Microsoft's implementation: https://www.microsoft.com/en-us/research/project/z3-3/.
Also see Prof. Karem Sakallah's work: https://scholar.google.com/citations?user=M441iZ8AAAAJ&hl=en.

9.14 Modeling CPU-GPU Unified Memory

To mitigate the cost of moving data between system memory and GPU memory, the latest architecture unifies system memory and GPU memory by sharing unified memory between these two components. For example, NVIDIA's Grace-Hopper superchip (https://resources.nvidia.com/en-us-grace-cpu/grace-hopper-superchip) shares a coherent memory between the Grace CPU and the Hopper GPU through NVLink. An accurate simulator to evaluate the performance overhead and benefit from this new architecture is needed.
You could start with the GPU simulator GPGPU-Sim and the CPU simulator SniperSim, and model CPU-GPU coherent memory from there.
https://ieeexplore.ieee.org/abstract/document/9789536?casa_token=bBO5sgUJLQYAAAAA:uuK54kZsG6TVges9WZayhse8wB9Hb694uNJzY8AosDb45GuBGrW9tWIA
https://ieeexplore.ieee.org/abstract/document/10067395

9.15 CPU-Accelerator Coherence via CXL

The latest development of Compute Express Link (CXL) allows low-latency communication between CPUs and accelerators via a uniform PCIe physical link. This allows an accelerator to access host memory coherently through this channel with acceptable latency.
This project is about designing a coherence protocol between an accelerator (for example, a GPU or an FPGA) and verifying its performance. Note that this coherence protocol should interact nicely with the host's coherence protocol and the device's memory model. Additional consideration must be given to the different latency and bandwidth limitations on the processor intra-core network, the CXL link, and the accelerator, as well as the latency requirement of the accelerator.
https://www.computeexpresslink.org/download-the-specification
https://ieeexplore.ieee.org/abstract/document/10015193

9.16 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 $\sim$0.5–2$\times$ 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-microscopy-with-shim/`

- **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`

9.17 Consistency for Compute in Storage

Traditionally, persistent storage like SSDs and hard drives are considered stateless. Once a block of data is sent to it, it is stored as is and will never change. However, recent work has shown that it is possible to perform computation inside storage devices. This opens up the possibility of using storage devices as accelerators. Now that the storage has the freedom to modify data in parallel with the host system, it is important to ensure that the host system sees a consistent view of the data.
The goal of this project is to design a programming model and system functions to access the process-in-storage (PIS) device. Both programmability and performance considerations need to be taken.

9.18 Designing Scalable Synchronization Primitives for Many-Core or GPU Systems

Modern libraries provide efficient synchronization for different situations. For example, MCS locks provide maximum throughput to contended critical sections, while pthreads/Linux futexes ensure 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).

9.19 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 it

with advanced features such as speculative requesting, cruise missile invalidations, migratory sharing optimizations, etc.
Formalizing CXL cache coherence: https://arxiv.org/abs/2410.15908
http://ieeexplore.ieee.org/abstract/document/6522351/
http://ieeexplore.ieee.org/abstract/document/7851544/

9.20 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 an LLVM compiler pass that can inspect the control-flow graph and identify parallelizable sections of the code.

9.21 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 accelerating it on a CPU or an FPGA.

9.22 Secure Coherence

In programming assignment 2, you will be verifying a standard MSI protocol using a formal verification tool called Murphi. It assumes all messages sent to the network will eventually arrive at their destination. We assume the inter-core network is honest, which means it will not tamper with or snoop your message intentionally to hurt the system.
Secure coherence protocol assumes a powerful adversary who has access to the inter-core network where it can snoop and tamper with the messages sent on it. This means this adversary can potentially force-drop messages or forge messages to control the contents in cache, and thus attack the system. For example, it can forge a data reply and trick the core into accepting whatever value the adversary wants to inject.
This project asks you to design and verify a secure coherence protocol using security primitives such as block ciphers and MACs. You can assume each party node is trusted and has established a common secret (key) beforehand.
Such a secure coherence protocol is an important component in building trusted hardware (https://dl.acm.org/doi/abs/10.1145/1278480.1278482, https://eprint.iacr.org/2016/086.pdf) on modern multiprocessor platforms or on chiplet-design processors where the inter-chiplet network might be vulnerable.

9.23 Secure Parallel Architectures

Significant security vulnerabilities have been recently exposed in modern hardware, such as cold boot attacks, Rowhammer, Spectre, and Meltdown. 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. Speculative execution can also lead to data leakage through side channels; attacks that employ this method (like Spectre and Meltdown) are known as transient execution attacks. In this project, you can study new vulnerabilities as well as design and evaluate defenses against them. For instance, you can try to find vulnerabilities in open-source processors (either using formal methods or other approaches). If you discover vulnerabilities, you

can try modifying the processor to fix them. 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., a public cloud) where you cannot trust even the system software (e.g., the cloud OS). This is important for application developers that would like to use the cloud (banks, hospitals, defense, etc.) but are wary of computing on sensitive private data on the cloud. To accomplish this, modern processors provide specialized hardware support in the form of 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 offload sensitive parts of computation to 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.

9.24 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.

9.25 Verify the Hardware Security of a Microarchitectural Design (or Synthesize Exploits for It) Using CheckMate

CheckMate (`https://ieeexplore.ieee.org/document/8574598`) is a methodology and tool for formally verifying hardware security properties on microarchitectural designs up to a bound. If the verification fails, CheckMate synthesizes 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.