Lecture 4.1
OpenMP: Serial to Parallel

EN 600.320/420

Instructor: Randal Burns

7 February 2018

Department of Computer Science, *Johns Hopkins University*

Serial to Parallel

- My code's not running fast enough: the most common parallel development scenario
  - Video: data delays produce jitter, stalls
  - Web: page render time causes user loss, discomfort
  - Batch processing, indexing, analysis not completing in time
  - High-throughput finance: other models running faster and beating mine to a decision -> lose arbitrage opportunity
- This leads to a natural software engineering process
  - Profile code: find out what's slow
  - Parallelize slow part(s) only
  - Migrate from serial implementation to parallel implementation

Serial to Parallel (ii)

- Not the best process
  - Serial to parallel doesn't produce the best designs
  - Best parallel implementation may require a totally different design
  - No natural evolution
- Just the easiest
  - Compared to a clean-slate redesign

What is OpenMP

- Parallel programming environment (not language) for:
  - Master/slave and/or fork/join execution model
  - Loop parallelism patterns
  - Shared-memory architectures
- But this doesn't mean anything yet.

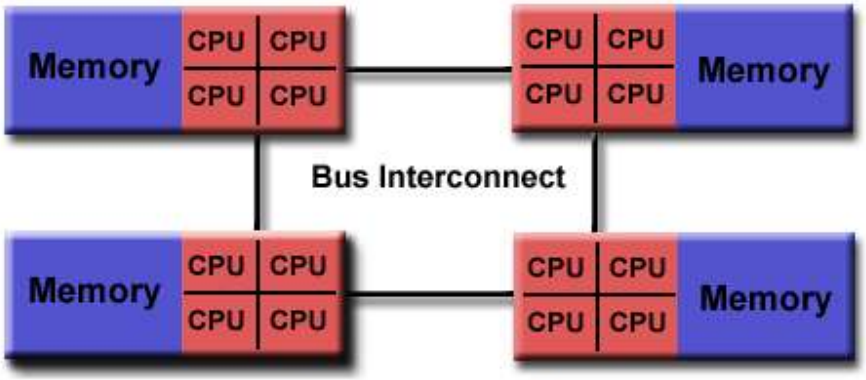
- It's the simplest approach to parallelism
  - Write a serial program in a language that you know (C++ or Fortran)
  - Add directives to parallelize portions of the code
  - Get a parallel program that computes that exact same result (*serial to parallel equivalence)*

OpenMP and Serial to Parallel

- Not fundamentally a serial to parallel environment
- But mostly used in this way because:
  - Programs have a mix of serial to parallel parts
  - Supports serial equivalence
  - All code is serial, parallel parts are defined with directives
- Conforms nicely with Amdahl's law
  - *Why did I say that?*

Shared Memory

- Coherent read/write to common memory from multiple cores/processors/(machines)
  - Coherent = repeatable read, read last write, ….
  - Abstraction that there is a single memory for all processors
  - Data sharing by reading/writing to memory
- Hardware that provides this abstraction are called *shared memory architectures* (typically in a "single machine")
  - Even if there are different physical memories
  - Non-Uniform memory architectures are typical today



Shared Memory (NUMA)

https://computing.llnl.gov/tutorials/parallel_comp/

Hardware for OpenMP

- Intel XEON Phi
  - x86 compatible co-processors (P54C – original Pentium)
  - 72 cores, 1.5 GHz
  - 115 GB/s memory bandwidth

- Compares well with Tesla GPUs, but programmable via OpenMP