Parallel Computing Wrapup

John Mellor-Crummey

Department of Computer Science
Rice University

johnmc@rice.edu

Course Objectives

- **Learn fundamentals of parallel computing**
  - —principles of parallel algorithm design
  - —programming models and methods
  - —parallel computer architectures
  - —modeling and analysis of parallel programs and systems
  - —parallel algorithms
- **Develop skill writing parallel programs**
  - —programming assignments employing a variety of models
- **Develop skill analyzing parallel computing problems**
  - —develop parallelizations for different styles of computations

Review: Parallel Algorithm Design

Recipe to solve a problem using multiple processors

Typical steps for constructing a parallel algorithm

- identify what pieces of work can be performed concurrently
- partition and map work onto independent processors
- distribute a program's input, output, and intermediate data
- coordinate accesses to shared data: avoid conflicts
- ensure proper order of work using synchronization

Why "typical"? Some of the steps may be omitted.

- if data is in shared memory, distributing it may be unnecessary
- if using message passing, there may not be shared data
- the mapping of work to processors can be done statically by the programmer or dynamically by the runtime

Principles of Parallel Algorithm Design

- **Algorithm models**
  - —data-parallel, task graph, work pool
  - —master slave, pipeline, hybrid
- **Decomposition techniques**
  - —recursive
  - —data driven: input data, output data, intermediate data
  - —hybrid decomposition
  - —exploratory decomposition
  - —speculative decomposition
- **Task generation**
  - —static vs. dynamic

assignment 4

assignment 1

4

assignment 2

assignment 3

assignment 1

1

2,3,4

Implementation Techniques

- Concurrency and mapping
  - static mapping strategies for regular problems
  - dynamic mapping
    - centralized task queue
    - work stealing
- Communication model
  - one-sided vs. two sided
- Collective communication
  - flavors
    - one-to-all: broadcast
    - all-to-one: reduce
    - all-to-all
    - parallel prefix computations: scan
    - gather/scatter
  - implementation techniques
    - broadcast of large messages as scatter + all-to-all

Programming Models

- **Shared-memory parallel programming**
  - —Cilk/Cilk++
  - —OpenMP
  - —Pthreads
- **Global address space programming models**
  - —Unified Parallel C (UPC)
- **Message passing and MPI**
- **GPU programming with CUDA**
- **MapReduce**

Parallel Architectures

- **Control structure and communication models**
  - —control structure: SIMD, MIMD
  - —communication models
    - shared address space
    - message passing platforms
- **Network topologies**
  - —static/direct vs. dynamic/indirect networks
  - —bus, crossbar, omega, hypercube, fat tree, mesh, Kautz graph
  - —hybrid interconnects
  - —evaluation metrics
    - degree, diameter, bisection width, channel width & rate, cost
- **Coherence, routing, and network embeddings**
  - —blocking vs. non-blocking networks
  - —routing techniques: store & forward, packet, wormhole
  - —cache coherence: protocols, snoopy caches, directories, SCI
  - —embeddings: dilation, congestion

Synchronization

- **Insufficient synchronization causes data races**
  - —unordered, conflicting operations
- **Mutual exclusion: classical algorithms for locks**
  - —explore formal reasoning about concurrent operations
- **Lock synchronization with atomic primitives**
  - —practical algorithms for pairwise coordination
- **Barrier synchronization**
  - —separate phases to prevent overlap of conflicting operations
  - —strategies for fast, primitive collective synchronization

Parallel Algorithms

- **Parallel sorting**
- **Dense matrix algorithms**
  - —Cannon's algorithm
  - —2.5D matrix multiply

Top Ten Tips for Parallel Computing

It's all about the performance

- **Use an efficient algorithm**
  - —clever implementation will yield to asymptotic inefficiency at scale
- **Partition your data and computation carefully**
  - —the wrong data partitioning can yield high communication volume
  - —the wrong computation partitioning can lead to load imbalance
    - – work stealing can help
- **Choose your programming model judiciously**
  - —shared-memory models make irregular problems easier
- **Avoid serialization**
  - —efficiency requires all processors and cores to be computing
  - —may require changes to algorithm and partitioning of data & computation
- **Choose the proper grain size for computation**
  - —wrong grain size can lead to excessive communication frequency

Top Ten Tips for Parallel Computing

- **Design carefully to avoid race conditions**
  —an ounce of design is worth a pound of debugging
- **Avoid contention**
  —shared variable “hot spots”
  —msg passing: contention for interconnect links or destinations
- **Use the cache**
  —on microprocessor-based systems, memory hierarchy is IMPORTANT
- **Don’t forget the microarchitecture**
  —an efficient algorithm kernel can boost performance by integer factors
- **Exploit parallelism at all levels**
  —SIMD instructions
  —instruction-level parallelism on pipelined processors
  —multiple cores; multiple threads per core (SMT, SIMT)
  —multi-socket nodes (SMP)
  —hardware accelerators (GPU, manycore) in nodes
  —clusters and supercomputers: nodes + interconnect