Lecture 37: Parallel Programming Systems and Models

Ioan Raicu
Department of Electrical Engineering & Computer Science
Northwestern University

EECS 211
Fundamentals of Computer Programming II
June 1st, 2010
• Standard sequential architecture
• Use multiple
  – Datapaths
  – Memory units
  – Processing units
• SIMD
  – Single instruction stream, multiple data stream
• SIMD
  – Advantages
    • Performs vector/matrix operations well
      – EX: Intel’s MMX chip
  – Disadvantages
    • Too dependent on type of computation
      – EX: Graphics
    • Performance/resource utilization suffers if computations aren’t “embarrassingly parallel”.
• MIMD
  – Multiple instruction stream, multiple data stream
Parallel Architectures

• MIMD
  – Advantages
    • Can be built with off-the-shelf components
    • Better suited to irregular data access patterns
  – Disadvantages
    • Requires more hardware (!sharing control unit)
    • Store program/OS at each processor

• Ex: Typical commodity SMP machines we see today.
• **Task Communication**
  - **Shared address space**
    • Use common memory to exchange data
    • Communication and replication are implicit
  - **Message passing**
    • Use send()/receive() primitives to exchange data
    • Communication and replication are explicit
• Shared address space
  – Uniform memory access (UMA)
    • Access to a memory location is independent of which processing unit makes the request.
  – Non-uniform memory access (NUMA)
    • Access to a memory location depends on the location of the processing unit relative to the memory accessed.
Parallel Architectures

• Message passing
  – Each processing unit has its own private memory
  – Exchange of messages used to pass data
  – APIs
    • Message Passing Interface (MPI)
    • Parallel Virtual Machine (PVM)
Parallel Algorithms

• Algorithm
  – a sequence of finite instructions, often used for calculation and data processing.

• Parallel Algorithm
  – An algorithm that can be executed a piece at a time on many different processing devices, and then put back together again at the end to get the correct result.
Parallel Algorithms

• Challenges
  – Identifying work that can be done concurrently.
  – Mapping work to processing units.
  – Distributing the work
  – Managing access to shared data
  – Synchronizing various stages of execution.
• Models
  – A way to structure a parallel algorithm by selecting decomposition and mapping techniques in a manner to minimize interactions.
Models

- Data-parallel
- Task graph
- Work pool
- Master-slave
- Pipeline
- Hybrid
Parallel Algorithms

• Data-parallel
  – Mapping of Work
    • Static
    • Tasks -> Processes
  – Mapping of Data
    • Independent data items assigned to processes (Data Parallelism)
Parallel Algorithms

• Data-parallel
  – Computation
    • Tasks process data, synchronize to get new data or exchange results, continue until all data processed
  – Load Balancing
    • Uniform partitioning of data
  – Synchronization
    • Minimal or barrier needed at end of a phase
  – Examples
    • Ray Tracing
• Data-parallel
• Task graph
  – Mapping of Work
    • Static
    • Tasks are mapped to nodes in a data dependency task dependency graph (Task parallelism)
  – Mapping of Data
    • Data moves through graph (Source to Sink)
• Task graph
  – Computation
    • Each node processes input from previous node(s) and send output to next node(s) in the graph
  – Load Balancing
    • Assign more processes to a given task
    • Eliminate graph bottlenecks
  – Synchronization
    • Node data exchange
  – Examples
    • Parallel Quicksort, Divide and Conquer approaches
    • Scientific Applications that can be expressed in workflows (e.g. DAGs)
• Task graph
• Work pool
  – Mapping of Work/Data
    • No desired pre-mapping
    • Any task performed by any process
    • Pull-model oriented
  – Computation
    • Processes work as data becomes available (or requests arrive)
Parallel Algorithms

• Work pool
  – Load Balancing
    • Dynamic mapping of tasks to processes
  – Synchronization
    • Adding/removing work from input queue
  – Examples
    • Web Server
    • Bag-of-tasks
• Work pool
Parallel Algorithms

• Master-slave
  – Modification to Worker Pool Model
    • One or more Master processes generate and assign work to worker processes
    • Push-model oriented
  – Load Balancing
    • A Master process can better distribute load to worker processes
• Pipeline
  – Mapping of work
    • Processes are assigned tasks that correspond to stages in the pipeline
    • Static
  – Mapping of Data
    • Data processed in FIFO order
      – Stream parallelism
Parallel Algorithms

- Pipeline
  - Computation
    - Data is passed through a succession of processes, each of which will perform some task on it
  - Load Balancing
    - Insure all stages of the pipeline are balanced (contain the same amount of work)
  - Synchronization
    - Producer/Consumer buffers between stages
- Ex: Processor pipeline, graphics pipeline
• Pipeline