Parallel Computation Models (Reading 4)

Computation model: a coherent collection of mechanisms for

- Communication
- Synchronization
- Partitioning
- Placement
- Scheduling

Across all levels of abstraction (algorithm, language, machine)

Flynn’s taxonomy:

<table>
<thead>
<tr>
<th></th>
<th>Single instruction</th>
<th>Multiple instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single data</strong></td>
<td>SISD: uniprocessor system doing sequential execution</td>
<td>MISD: decryption of unknown encrypted format, parallel transcode (YouTube video encoding)</td>
</tr>
<tr>
<td><strong>Multiple data</strong></td>
<td>SIMD: Vector, MMX, SSE…, AVX, GPUs (SIMT)</td>
<td>MIMD: multicore, manycore, PThreads, MPI,...</td>
</tr>
</tbody>
</table>

Computational model is a theoretical model of how computation should occur, agnostic of hardware, language,...

Communication model:

- Example: producer-consumer
  - Producer-consumer also implies a synchronization model with it, but not true for all communication models
- Topology of communication: what is allowed in communication between different parallel portions
- Broadcast: someone can broadcast to everyone such as master thread, distinct from all-to-all
- Nearest neighbor: weather code, higher dimensional spaces
- Trees
- Explicit communication: sender and receiver (though they may not be known to each other), program has to specify communication
- Implicit communication: no explicit send/receive communication mechanism, no requirement that anybody reads the data (such as shared memory), dataflow machines
Synchronization model:
- Communicating that something has occurred, rather than just data
- Producer-consumer: communication is synchronization across different parallel portions
- Barriers: all processes/threads/cores have to reach some point before any process/thread/core can go on
- Locks, mutexes, semaphores: designed for different entities from operating on same piece of data at the same
  - Mutex: one entity can be in critical section at a time
  - Semaphore: some number of entities can access section
- Different types of locks and what to do when trying to obtain a lock
- Speculative synchronization: guess and check, transactional programming model (abort and rollback if not successful)

Partitioning model:
- Grain size: how big/small to divide work
- What is the independent unit of work
- Explicit or implicit to programmer

Placement model:
- Not the same as scheduling
- Where to run something (thread or data)
- Explicit or implicit
- Static or dynamic

Scheduling model:
- When to run something
- Explicit or implicit
- Need to avoid deadlocks, forward progress guarantees

*Shared Memory (threads)*
Implementation: PThreads, other thread libraries
Communication: all-to-all topology (talk through shared memory), implicit
Synchronization: locks, barriers, maybe speculative
Partitioning: explicit, programmer has control but has to do all of the management, can get it wrong
Placement: implicit, left up to OS/runtime to place
Scheduling: implicit, left up to OS/runtime
Message Passing

Implementation: MPI: Message Passing Interface (OpenMPI, MPI Mich), other libraries as well
Communication: basic communication is through message, doesn’t say much about topology.
  • Explicit message passing: send(), receive(), unicast (one to one), multicast (one to multiple), broadcast (one to all)
Synchronization: producer-consumer, can build barriers on top
Partitioning: explicit
Placement: implicit, there are hints in newer APIs about which processors might need to communicate
Scheduling: implicit, runtime

Even vs uneven workloads, different models appropriate
  • Multigrid and sparse matrices
  • Work stealing

PRAM

Theoretical model for PRAM good for conceptualizing of time and space bounds for parallel computation

Turing Machine: theoretical machine
  • Infinitely long tape (unbounded memory) with discretized locations
  • Read/write head
  • State transition table
    ○ Maps <tape input, state> to <write to tape, updated head, new state>
    ○ States include a start state and an accept state
  • Church-Turing thesis and universality

Can simulate nondeterministic TM with deterministic TM

TM not very useful for big-O analysis, but can say whether computable in polynomial deterministic/nondeterministic time (complexity classes) or noncomputable

TMs and big-O used in sequential analysis
PRAMs used for parallel analysis of algorithms
Number of accesses to shared global memory most important part of analysis

Original paper: complete concurrency in the reads, unbounded number of processors, no concurrency in writes to the same location
CLRS paper: define exclusive/concurrent read and write models, there is a limitation on the # of processors available