Model Checking and Course Review

Mark Greenstreet

CpSc 418 – Nov. 27, 2012

Lecture Outline

Model Checking & Course Review

- Model Checking (finishing up)
  - Parallel Model Checking: Stern & Dill
  - Termination Detection
  - Scaling it up: PReach
- Course Summary

Stern and Dill's algorithm (w/o termination detection)

```
reach(Q, Nprocs) ->
   Pids = map(fun(I) -> spawn(doCheck, []) end,
              seq(0, Nprocs-1)),
   map(fun(Pid) -> Pid ! {allPids, Pids} end, Pids),
   sendStates(Pids, Q).
sendStates(Pids, [Hd | Tl]) ->
   nth(hashToPid(Hd), Pids) ! {state, Hd},
   sendStates(Pids, Tl).
doCheck() -> receive {allPids, Pids} -> doCheck(Pids, ∅) end.
doCheck(Pids, Q) ->
   receive {state, State} ->
      if
         State ∈ Q -> doCheck(Pids, Q);
         true ->
            sendState(Pids, successors(State)),
            doCheck(Pids, Q ∪ {State})
      end
   end.
```

Termination Detection

How do we know when we're done?

- Simple idea:
  - If root process is idle for a while
    - It sends messages to all other processes asking if they are idle.
    - If they all reply `idle`, then we conclude that we're done.
- What's wrong with this "solution"?

Stern & Dill's Approach (part 1)

Make `doCheck` keep track of how many messages sent and received:

```
doCheck() ->
   receive {allPids, Pids} -> doCheck(Pids, ∅, 0, 0, noDelay)
end.
doCheck(Pids, Q, Nsent, Nrecv, ToSend) ->
   TimeOut = if (hd(Pids) == self()) -> 1000;
                true -> infinity end,
   receive
      {state, State} ->
         if
            State ∈ Q ->
               doCheck(Pids, Q, Nsent, Nrecv+1, ToSend);
            true ->
               doCheck(Pids, Q ∪ {State},
                  Nsent+length(Q), Nrecv+1,
                  sendStates(Pids, successors(State), ToSend))
         end;
```

Stern & Dill's Approach (part 2)

```
...
receive
   ...
   doneQuery ->
      hd(Pids) ! {msgcnt, (Nsent - Nrecv)},
      doCheck(Pids, Q, Nsent, Nrecv, []);
   continue ->
      sendStates(Pids, ToSend, noDelay),
      doCheck(Pids, Q, Nsent + length(ToSend), Nrecv, noDelay);
   die -> ok
   after TimeOut ->
      [Pid ! doneQuery || Pid <- tl(Pids)], % send all Pids a done qu
      % check message counts
      M = lists:sum([ receive msgcnt, C -> C || _ <- Pids]), if
         M == 0 -> map(fun(Pid) -> Pid ! die end; Pids);
         M /= 0 -> map(fun(Pid) -> Pid ! continue end; Pids)
      end
end.
```

Stern & Dill's Approach (part 3)

```
sendStates(State, Pids, ToSend) ->
   if
      ToSend == noDelay ->
            nth(hashToPid(State), Pids) !  {state, State},
            noDelay;
      true -> [State, ToSend]
   end.
```

Why Stern & Dill's approach is correct

PReach: scaling it up to hundreds of processors

- Issues
  - Input queue overflow
  - Load balancing
  - Message batching
- Results
  - Used at Intel on clusters with several hundred machines.
  - Current work on
    - ★ Adding checks for liveness properties.
    - ★ Performance: we always want bigger and faster.
    - ★ Robustness: how to keep going if a machine crashes.

Course Summary

- Architectures
- Algorithms
- Performance
- Correctness
- Programming languages and APIs

Course Summary: Architectures (part 1)

- Pipelined machines and superscalar architectures
  - The hardware finds the parallelism.
  - Little or no effort needed by the programmer.
  - This “free” parallelism is limited, but useful.
  - And not really free: superscalars are complicated and power hungry.
- Shared memory machines
  - Caches and coherence protocols: MESI
  - Sequential consistency:
    - Memory acts as if all reads and writes done in some sequential order.
    - This order is consistent with program order on individual processors.
  - Weak consistency
    - Real machines make weaker guarantees than sequential consistency.
    - Allows reads to bypass writes.
    - Best to use synchronization methods from APIs such as pthreads or Java threads
      - Alternative is to thoroughly understand the messy details of real cache protocols.

Course Summary: Architectures (part 2)

- Message Passing Machines
  - Communication is explicit in the programs.
  - Much focus on network topologies:
    - Ring, star, crossbar, hypercube, meshes and tori
    - Why are 3D tori common starting point for large machines?
    - How are 3D tori extended to 5D or 6D? What are the benefits?
- GPGPUs
  - SIMD (single-instruction, multiple-data) parallelism
  - Deep pipelines: how does this impact the software?
  - How are conditionals handled in SIMD?

Course Summary: Algorithms

Course Summary: Performance

Course Summary: Correctness

Course Summary: Languages and APIs