Parallel Algorithms

Lecture 11: Fault tolerance

Ronald Veldema

Introduction (1)

- all networks are troublesome (bit errors)
- ``failures can be as frequent as one every two days on 2000 processors."
  - http://csmr.ca.sandia.gov/projects/ftalgs.html
- Large ibm blue gene:
  - 1 failure every 6 hours..
- large simulations can take weeks...

Introduction (2)

- intro: all machines are troublesome (hardware failures: disks, fans)
- cosmic rays: flip single bit in memory, bus.
- Maybe once every compute year.
- But what if 10000 parallel machines ?

What does this have to do with Parallel Algorithms ?

- Parallel programs need to deal with failures
- Parallelism not for speed but for fault tolerance
  - Execute some program twice in parallel and check that results are the same

What can we *expect* to fail

- network bit errors
  - single bit, burst errors
- machine crash
  - before or after generating faulty data
  - disk failures
- memory bit flip

Theory..

- fault tolerance = detection + recovery
- Theory people: fault tolerance = extra states in process state diagram

Exception handling

- ``fault tolerant program design''
- check return values
  - has anybody here ever checked the return value of ``printf'' ?
- try {} catch {}

Handle problems using parallel programming: work duplication

- duplicate all work
  - Wastes resources
- duplicate all data
  - Wastes resources
- duplicate all work and all data
  - Waste even more
- Question: does duplication handle
  - Data corruption ?
  - Machine errors ?

Checkpointing

- Checkpointing is the process of saving the complete state of your program to non-volatile storage:
  - Call-stack (local variables, parameters)
  - Allocated objects
  - Global variables
  - Etc.
- A 'restore' then takes the checkpoint, restores it and execution can continue as if nothing happened

Checkpointing (1)

- pessimistic / optimistic ?
- partial recompute
  - side effects ?
  - open file descriptors ?
- memory exclusion
- when ?
- where to store checkpoints
- checkpoint whole cluster or single machines/thread

Checkpointing (2)

- coordinated
  - coordinator tells everybody to create a checkpoint
  - needs all cpus in cluster to simultaniously checkpoint
  - Blocking
  - non-blocking

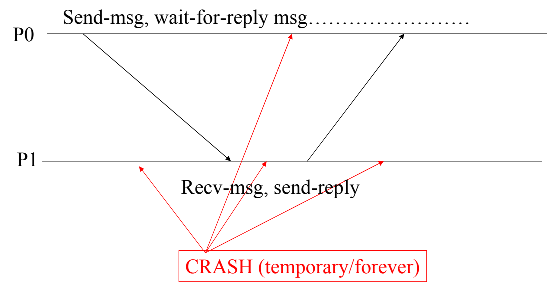
Checkpointing (3)

- Uncoordinated
  - ONLY when tasks independent
    - (as otherwise we would restore the cluster to inconsistent state when restoring machine 1 which assumes message sent while machine 2 is about to wait for it etc)
  - SPMD programs: checkpoint each barrier
  - pure divide and conquer:
    - depend only on parameters/return values

Checkpointing (4)

- in flight messages ?
  - Message arrives as checkpoint has just been made but sender already assumed message delived.
  - After restore, message needs to be reposted !

Checkpointing (5)



Atomic Transactions

- lost lock msg
- lost data msg
- lost unlock msg
- rollback
  - while other machines have already mutated data

thread/process migration

- When notified that machine X is going down.
- When user logs in on machine
  - Non dedicated clusters
  - Condor

denial of service

- sometimes because of bugs...
- challenge based solvers

inherently fault tolerant algorithms or *naturally fault tolerant algorithms*

- 2/3 group voting for correct answer
- approximation algorithms
- many AI codes are heuristic based:
  - on failure find slightly less optimal solution.

Inherently fault tolerant algorithms

- Group voting for best answer
  - 2/3 voting
  - Implement multiple heuristics where each returns a value and a ‘confidence’ factor
    - One machine fails we only have less to chooze from

Inherently fault tolerant algorithms (1)

- Genetic algorithms
  - Remove part of population
    - One processor dies -> part of population no longer available
    - Random mutation can reconstruct the population that we missed

Inherently fault tolerant algorithms (2)

- Large neural networks
  - Take away a single neuron and the system will ‘learn’ to function without it