PRAM MODEL

BMCS3003 DISTRIBUTED SYSTEMS AND PARALLEL COMPUTING

WHY MODELS?

- What is a machine model?
  - A abstraction describes the operation of a machine.
  - Allowing to associate a value (cost) to each machine operation.
- Why do we need models?
  - Make it easy to reason algorithms
  - Hide the machine implementation details so that general results that apply to a broad class of machines to be obtained.
  - Analyze the achievable complexity (time, space, etc) bounds
  - Analyze maximum parallelism
  - Models are directly related to algorithms.

RAM (RANDOM ACCESS MACHINE) MODEL

- Memory consists of infinite array (memory cells).
- Each memory cell holds an infinitely large number.
- Instructions execute sequentially one at a time.
- All instructions take unit time
  - Load/store
  - Arithmetic
  - Logic

- Running time of an algorithm is the number of instructions executed.
- Memory requirement is the number of memory cells used in the algorithm.

RAM (RANDOM ACCESS MACHINE) MODEL

- The RAM model is the base of algorithm analysis for sequential algorithms although it is not perfect.
  - Memory not infinite
  - Not all memory access take the same time
  - Not all arithmetic operations take the same time
  - Instruction pipelining is not taken into consideration
- The RAM model (with asymptotic analysis) often gives relatively realistic results.

PRAM (PARALLEL RAM)

- A model developed for parallel machines
  - An unbounded collection of processors
  - Each processor has an infinite number of registers
  - An unbounded collection of shared memory cells.
  - All processors can access all memory cells in unit time (when there is no memory conflict).
  - All processors execute PRAM instructions synchronously
    - Somewhat like SIMD, except that different processors can run different instructions in the lock step.
    - Some processors maybe idle.

PRAM (PARALLEL RAM)

- A model developed for parallel machines
  - Each PRAM instruction executes in 3-phase cycles
    - Read from a share memory cell (if needed)
    - Computation
    - Write to a share memory cell (if needed)
    - Example: for all I, do A[i] = A[i-1]+1;
      - Read A[i-1], compute add 1, write A[i]

– The only way processors exchange data is through the shared memory.

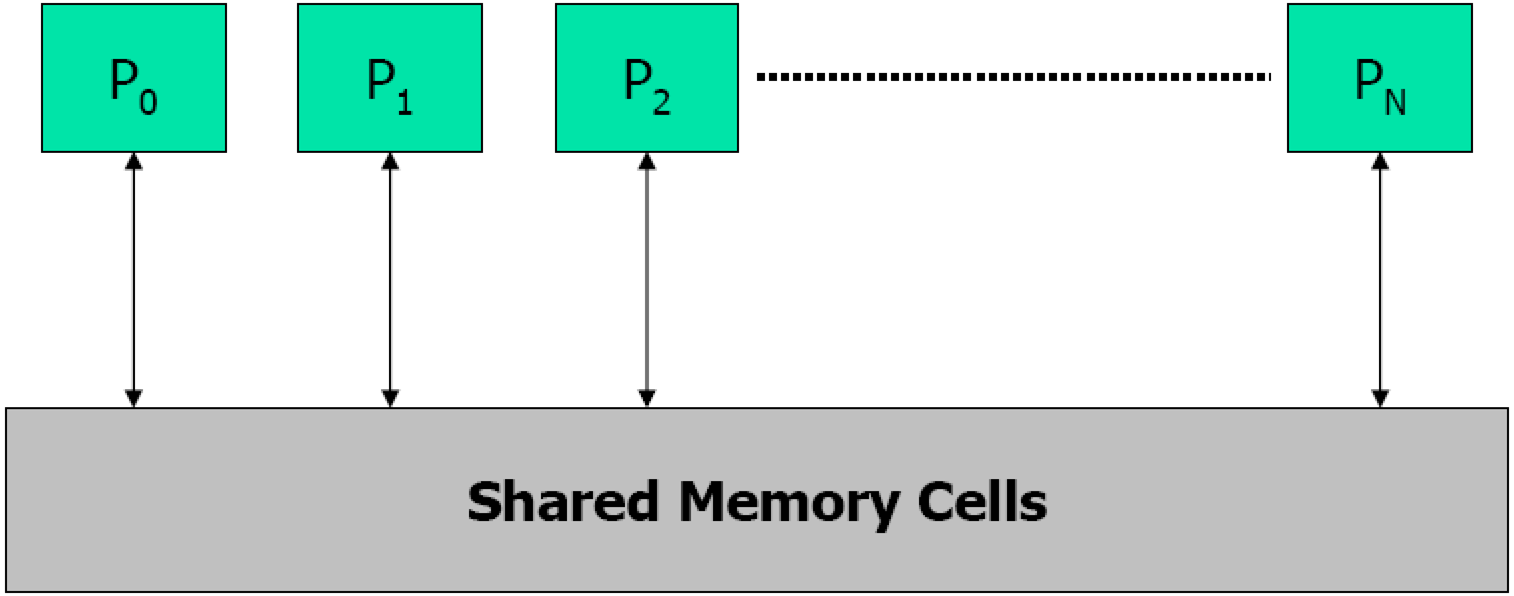
PRAM (PARALLEL RAM)

Parallel time complexity: the number of synchronous steps in the algorithm

Space complexity: the number of shared memory

Parallelism: the number of processors used

PRAM



All processors can do things in a synchronous manner (with infinite shared Memory and infinite local memory), how many steps do it take to complete the task?

PRAM – FURTHER REFINEMENT

- PRAMs are further classified based on how the memory conflicts are resolved.
  - Read
    - Exclusive Read (ER) – all processors can only simultaneously read from distinct memory location (but not the same location).
      - What if two processors want to read from the same location?
    - Concurrent Read (CR) – all processors can simultaneously read from all memory locations.

PRAM – FURTHER REFINEMENT

- PRAMs are further classified based on how the memory conflicts are resolved.
  - Write
    - Exclusive Write (EW) – all processors can only simultaneously write to distinct memory location (but not the same location).
    - Concurrent Write (CW) – all processors can simultaneously write to all memory locations.
      - Common CW: only allow same value to be written to the same location simultaneously.
      - Random CW: randomly pick a value
      - Priority CW: processors have priority, the value in the highest priority processor wins.

PRAM MODEL VARIATIONS

- EREW, CREW, CRCW (common), CRCW (random), CRCW (Priority)
  - Which model is closer to the practical SMP or multicore machines?
- Model A is computationally stronger than model B if and only if any algorithm written in B will run unchanged in A in the same parallel time, assuming the same basic properties.
  - EREW <= CREW <= CRCW (common) <= CRCW (random)

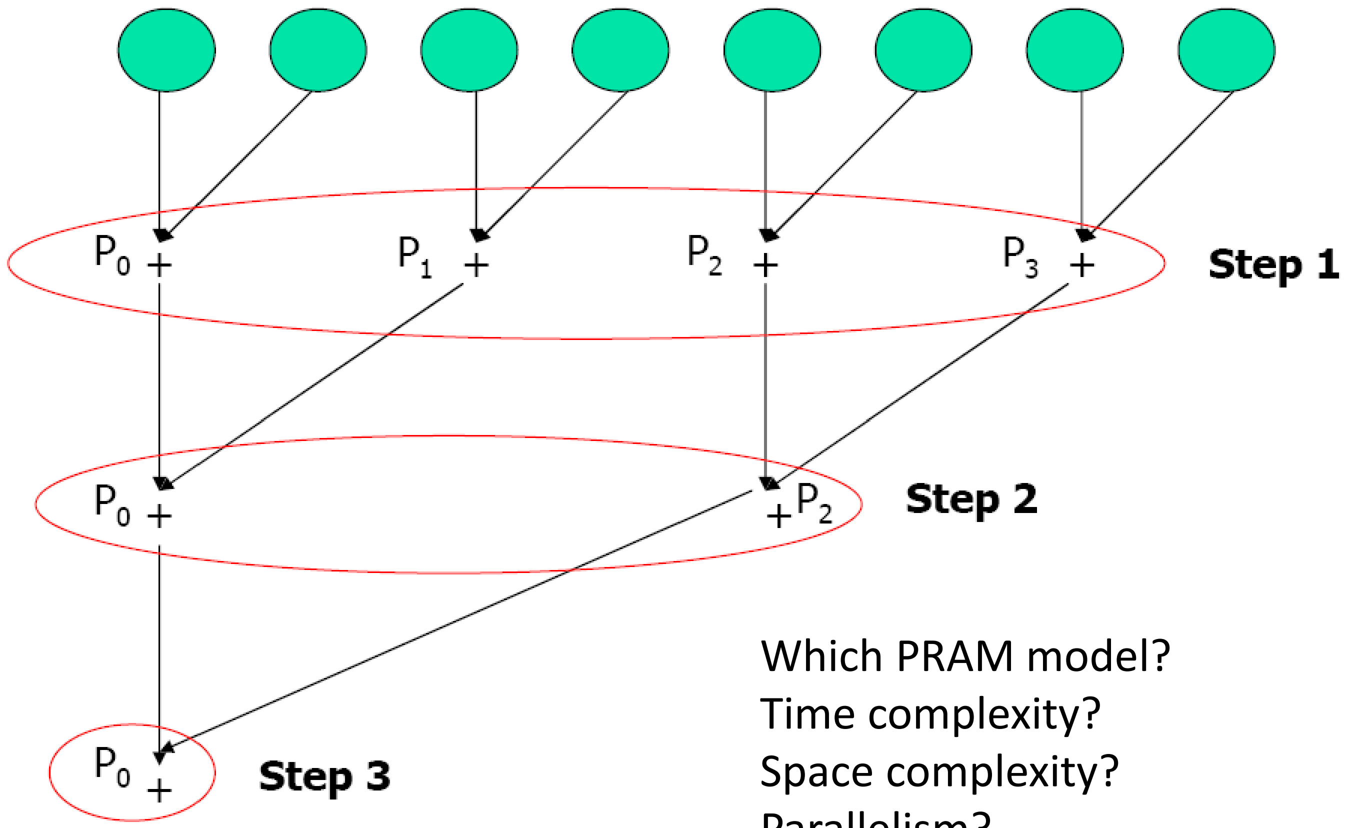
PRAM ALGORITHM EXAMPLE

- SUM: Add N numbers in memory M[0, 1, …, N-1]
- Sequential SUM algorithm (O(N) complexity)

```
for (i=0; i<N; i++) sum = sum + M[i];
```

- PRAM SUM algorithm?

PRAM SUM ALGORITHM



Which PRAM model?
Time complexity?
Space complexity?
Parallelism?
Speedup (.vs. sequential code)?

PARALLEL ADDITION

- Time complexity: log(n) steps
- Parallelism: n/2 processors
- Speed-up (vs sequential algorithm): n/log(n)

PARALLEL SEARCH ALGORITHM

- P processors PRAM with unsorted N numbers (P<=N)
- Does x exist in the N numbers?
- p_0 has x initially, p_0 must know the answer at the end.
- PRAM Algorithm:
  - Step 1: Inform everyone what x is
  - Step 2: every processor checks N/P numbers and sets a flag
  - Step 3: Check if any flag is set to 1.

PARALLEL SEARCH ALGORITHM

- PRAM Algorithm:
  - Step 1: Inform everyone what x is
  - Step 2: every processor checks N/P numbers and sets a flag
  - Step 3: Check if any flag is set to 1.
- EREW: O(log(N)) step 1, O(N/P) step 2, and O(log(N)) step 3.
- CREW: O(1) step 1, O(N/P) and O(log(N)) in step 2, and O(1) step 3.
- CRCW (common): O(1) step 1, O(N/P) step 2, and O(1) step 3.

PRAM MATRIX-VECTOR PRODUCT

- Given an n x n matrix A and a column vector X = (x[0], x[1], …, x[n-1]), B = A X
- Sequential code:

```
For(i=0; i<n; i++) for (j=0; j<n; j++) B[i] += A[i][j] * X[j];
```

- CREW PRAM algorithm
  - Time to compute the product?
  - Time to compute the sum?
  - Number of processors needed?
  - Why CREW instead of EREW?

PRAM MATRIX MULTIPLICATION

- CREW PRAM algorithm?
  - Time to compute the product?
  - Time to compute the sum?
  - Number of processors needed?

PRAM STRENGTHS

- Natural extension of RAM
- It is simple and easy to understand
  - Communication and synchronization issues are hided.
- Can be used as a benchmark
  - If an algorithm performs badly in the PRAM model, it will perform badly in reality.
  - A good PRAM program may not be practical though.
- It is useful to reason threaded algorithms for SMP/multicore machines.

PRAM WEAKNESSES

- Model inaccuracies due to assumptions below in PRAM model:
  - Unbounded local memory (register)
  - All operations take unit time
  - Processors run in lock steps
- Unaccounted costs
  - Non-local memory access
  - Latency
  - Bandwidth
  - Memory access contention

PRAM VARIATIONS

- Bounded memory PRAM, PRAM(m)
  - In a given step, only m memory accesses can be serviced.
  - Lemma: Assume $m' < m$. Any problem that can be solved on a p-processor and m-cell PRAM in t steps can be solved on a $max(p,m')$-processor m'-cell PRAM in $O(tm/m')$ steps.
- Bounded number of processors PRAM
  - Lemma: Any problem that can be solved by a p processor PRAM in t steps can be solved by a p' processor PRAM in t = O(tp/p') steps.
    - E.g. Matrix multiplication PRAM algorithm with time complexity O(log(N)) on N^3 processors→ on P processors, the problem can be solved in O(log(N)N^3/P).
- LPRAM
  - L units to access global memory
  - Lemma: Any algorithm that runs in a p processor PRAM can run in LPRAM with a loss of a factor of L.

PRAM SUMMARY

- The RAM model is widely used.
- PRAM is simple and easy to understand
  - This model never reaches beyond the algorithm community.
  - It is getting more important as threaded programming becomes more popular.
- The BSP (bulk synchronous parallel) model is another try after PRAM.
  - Asynchronously progress
  - Model latency and limited bandwidth