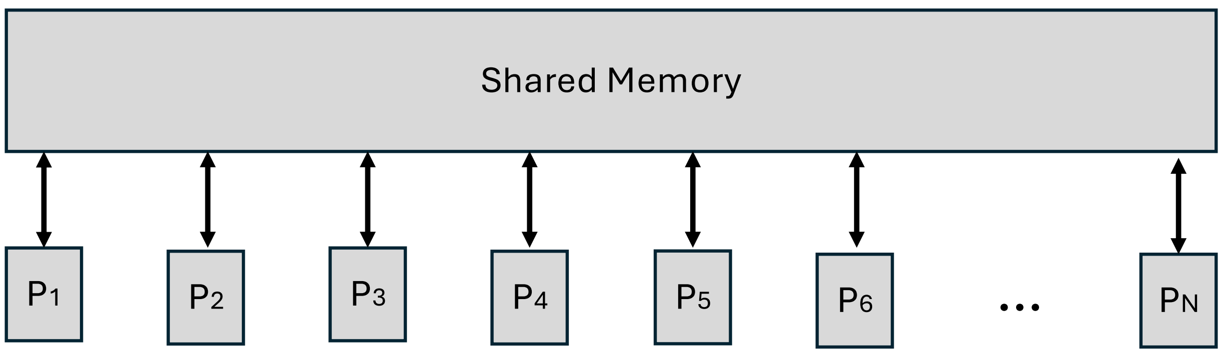
The PRAM Model

CSCE 626 – Parallel Algorithms

Roger Pearce

Parallel Random Access Memory (PRAM)

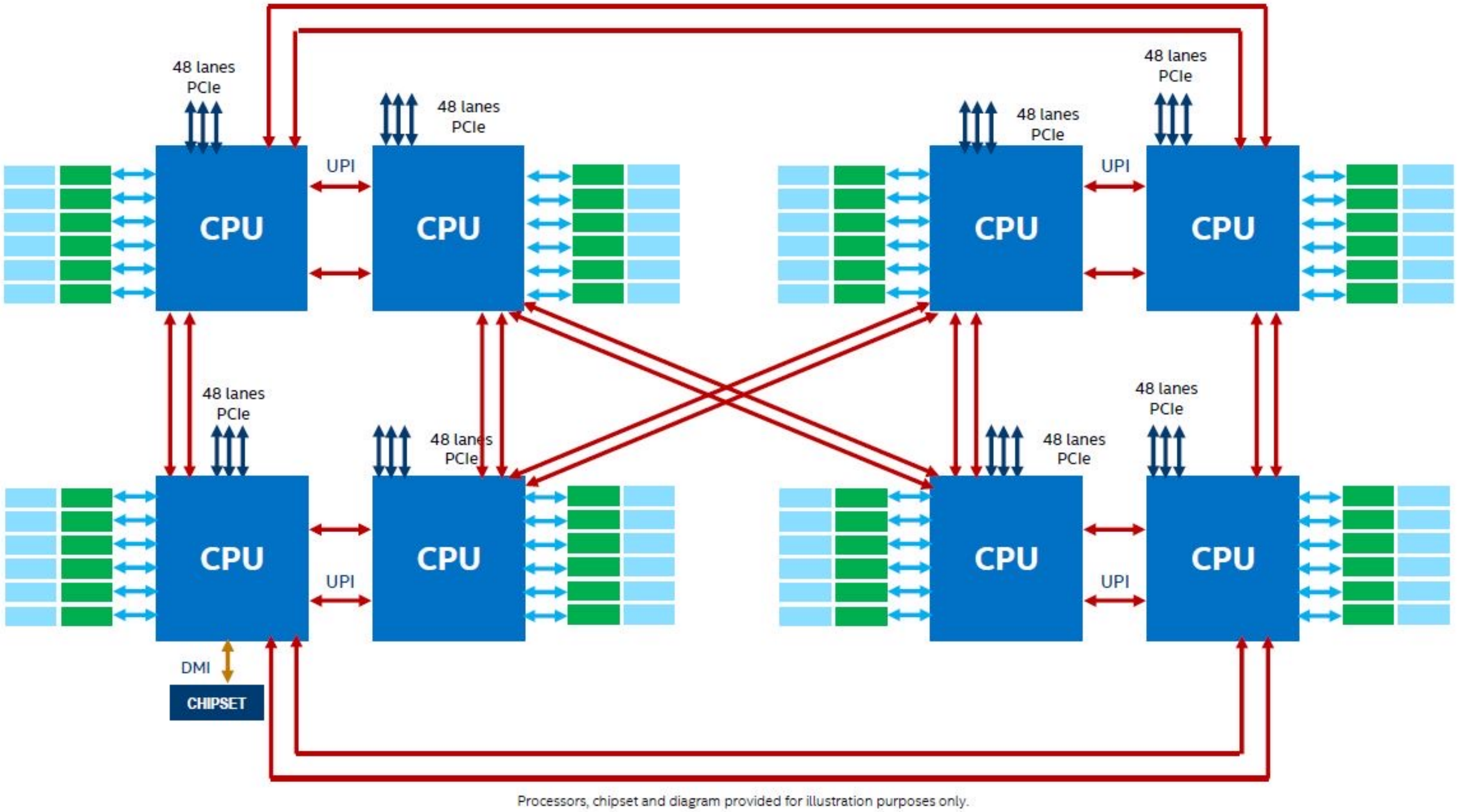
- Simplified parallelism model for asymptotic algorithm analysis
- Adds new parameter ‘P’ to represent the number of processors
- Each processors executes the same algorithm synchronously



3 memory modes of PRAM

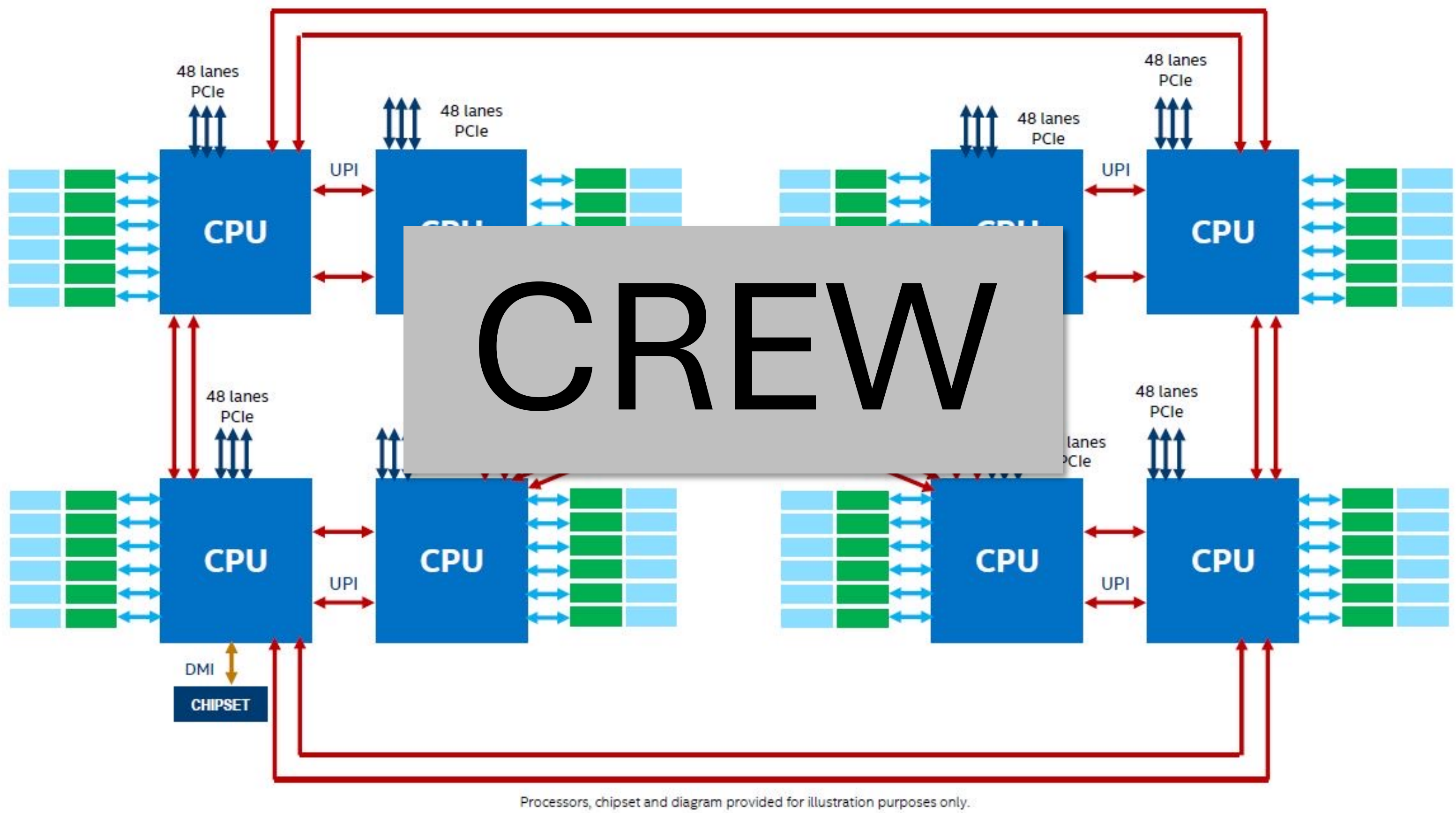
- CREW – Concurrent Read Exclusive Write
  - During a given algorithm step, each processor can can read the contents of a memory cell simultaneously, but at most 1 processor can write a value to a cell.
- CRCW – Concurrent Read Concurrent Write
  - During a given algorithm step, each processor can can read and write the contents of a memory cell simultaneously.
- EREW – Exclusive Read Exclusive Write
  - During a given algorithm step, only 1 processor can read and write to a memory cell at a time.

Which PRAM model is most like this machine?



[HPE Superdome notional diagram from servethehome.com]

Which PRAM model is most like this machine?



[HPE Superdome notional diagram from servethehome.com]

CREW

- Model that most closely represent today’s shared memory systems
- Ignores NUMA effects (variable memory latency cost due to variable distance to memory)
- When we program with OpenMP later, will primarily be for CREW algorithms

CRCW

- Most **powerful** yet **unrealistic** model
- Assumes an unbounded number of simultaneous writes to a single memory cell
- Several possible write modes exist:
  - Consistent model: All processors must write the same value
  - Arbitrary mode: Processors may write different values, but only one will be written, arbitrarily chosen. Causes **parallel nondeterminism.**
  - Priority mode: The processor with the lowest index (rank) wins
  - Fusion mode: An associative reduction is a performed on the fly (sum, min, max, etc.)

EREW

- Model that most closely represent today's distributed memory systems
- When we program with MPI later, will primarily be for EREW algorithms
- Only one processor can read or write a given cell at a time.

First Example: Find largest number (<u>very</u> <u>naïve</u> algorithm)

Input:

values – array of size N integers [0, 100000)

P – set of all processors indexes

Output:

largest – largest integer in values

```
largest = 0
private my_largest = 0
forall v in values parallel do
    my_largest = max( v, my_largest )
forall rank in P parallel do
    while my_largest > largest
        largest = my_largest        //controlled parallel nondeterminism
return largest
```

First Example: Find largest number (<u>very</u> <u>naïve</u> algorithm)

Input:

values – array of size N integers [0, 100000)

P – set of all processors indexes

Output:

largest – largest integer in values

```
1.  largest = 0
2.  private my_largest = 0
3.  forall v in values parallel do
4.      my_largest = max( v, my_largest )
5.  forall rank in P parallel do
6.      while my_largest > largest
7.          largest = my_largest        //controlled parallel nondeterminism
8.  return largest
```

Loop on line 3: O(N / P)
Line 5-7 is O(P) in worst case
Line 7 CRCW

Overall: O(N/P + P)

Can we do better?

Fork-join vs SPMD

- Many shared-memory programming models are fork-join, meaning there is a **main thread** that **forks workers** and then waits (joins) for their completion.
  - OpenMP is a prime example
- Many distributed-memory programming models are Single Program Multiple Data (SPMD), meaning the exact same code is always running on every processor all the time, starting at main()
  - MPI is a prime example

OpenMP Example: (Fork-join model)

```
#include <iostream>
#include <omp.h>

int main() {

  std::cout << "Hello World from main thread" << std::endl;

  #pragma omp parallel
  {
    int thread_id = omp_get_thread_num();
    std::cout << "Hello World from thread "
              << thread_id << std::endl;
  }

  return 0;
}
```

MPI Example: SPMD Model

```cpp
#include <iostream>
#include <mpi.h>

int main(int argc, char** argv) {
  // Initialize MPI
  MPI_Init(&argc, &argv);

  // Get the rank of the rank
  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  // Get the total number of ranks
  int size;
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  // Print a message from each rank
  std::cout << "Hello world from rank " << rank
            << " of " << size << std::endl;

  // Finalize MPI
  MPI_Finalize();

  return 0;
}
```

Questions ?

- Open discussion about PRAM model