EE 382C/361C: Multicore Computing **Fall 2016**

Lecture 1: August 24

Lecturer: Vijay Garg *Scribe: Rahul Jaisimha*

1.1 Introduction

This class was just an introduction to the course. It explained how to write concurrent programs using the different framework/languages we will be using in class. All files used in this lecture can be found in the `chapter1-threads` folder of the class github [1].

1.2 First Things First

Dr. Garg's office is a gun-free zone.

1.3 Writing Concurrent Programs

1.3.1 Java

See github file `java/HelloWorldThread.java`. Concurrent programming in java is implemented by extending the *Thread* class in java and overriding the `run()` function. `thread.start()` is used to start the thread. This program can be run on Unix with:

```
javac HelloWorldThread.java
java HelloWorldThread
```

1.3.2 OpenMP

See github file `openMP/hello.c`. Concurrent programming using openMP is implemented by including `omp.h` and using `-fopenmp` as an argument during compilation (as seen in compile.bat also on github). Use `#pragma` to run things in parallel. In `hello.c`, every thread should have a private copy of `tid`.

1.3.3 Pthreads

See github file `pthreads/hello.c`. This is the longest program. Compilation is normal unlike openMP. The program just has to include `pthread.h`.

1.3.4 Promela

See github file `spin/helloThreads.pml`. The program starts at `init` much like `main()` in other languages. Input the promela file and properties into *Spin* to run.
This program can be run on Unix with:
`spin helloThreads.pml`

1.3.5 Cuda

See github file `cuda/hello.cu`. Cuda is used to write GPU programs. More on Cuda programming in the next section.

1.4 GPU Programming

GPGPU = General Purpose GPU. GPGPUs are useful for matrix multiplication, neural networks, amongst other things. GPU programming relies on SIMD (Single Instruction Multiple Data). This means programming multiple processes to do the same thing on different data.

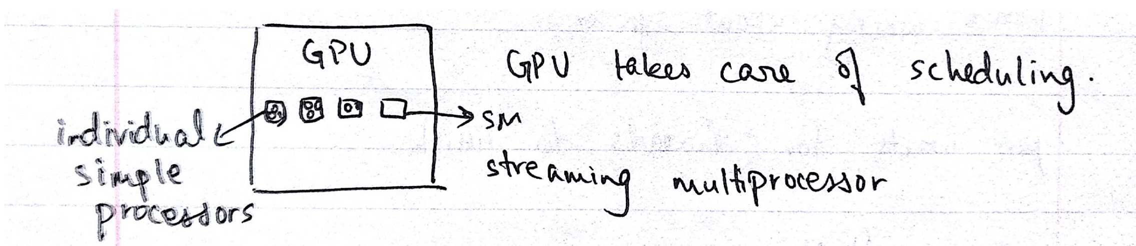

Figure 1.1: GPU

1.4.1 Cuda cont'd.

Cuda uses the notion of kernel. the `__global__` is used to denote running on the gpu. Cuda programs are compiled with the Nvidia Cuda compiler `nvcc`. Note that optimizing GPU programs is a lot harder than optimizing CPU programs.

1.5 PRAM

PRAM = Parallel RAM. PRAM is purely an abstract concept for developing parallel algorithms that assumes shared memory between many processing elements.

Algorithms for accessing the shared memory from each processing element are either of CRCW, CREW, ERCW, or EREW. E: Exclusive, C: Concurrent, R: Read, W: Write.

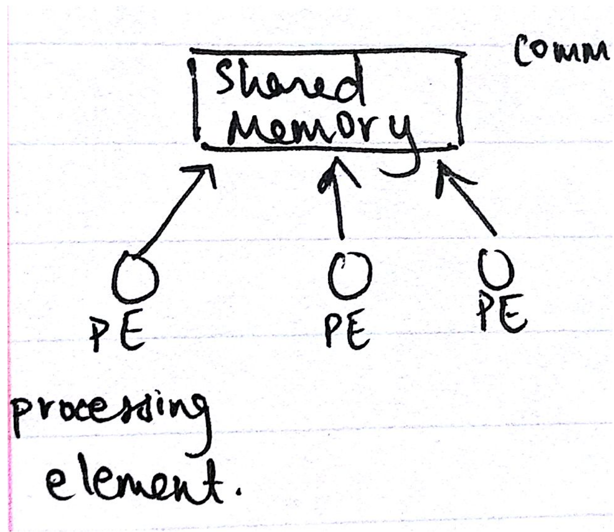

Figure 1.2: PRAM

1.6 The Dr. Garg Traditional First Day of Class Puzzle

Assume you have an array of unique natural numbers (non-negative numbers). Let its size be N. Find the largest element in the array.
In this section we use work complexity to denote number of compares. We also assume we have N^2 cores (or as many cores as we need).

1.6.1 Sequential Algorithm

Time Complexity = $O(N)$
Work Complexity = $O(N)$
Cores needed = 1

1.6.2 Binary Tree Algorithm

Split the array into a binary tree and compare two at a time recursively.

Time Complexity = $O(\log N)$
Work Complexity = $O(N)$
Cores needed = N

1.6.3 "Usain Bolt Algorithm"

Time Complexity = $O(1)$
Work Complexity = $O(N^2)$
Cores needed = N^2

Algorithm 1 "Usain Bolt algorithm"

$\forall i, isBiggest[i] := 1$
for all i, j **do**
  if $A[j] > A[i]$ **then**
    $isBiggest[i] \leftarrow 0$
  end if
end for
if $isBiggest[i]$ **then**
  $MAX := A[i]$
end if

1.6.4 Epilogue

Okay so that was pretty good, but can we solve this problem with a good time complexity like $O(\log N)$ without using so many cores?

References

[1] https://github.com/vijaygarg1/UT-Garg-EE382C-EE361C-Multicore/tree/master/chapter1-threads