CSC2/458 Parallel and Distributed Systems
A Case Study of a High-speed GPU NFA Implementation

Sreepathi Pai

April 20, 2022

URCS

Outline

Non-deterministic Finite Automata and the Automata Processor

Irregular Parallelism

Reducing Data Movement

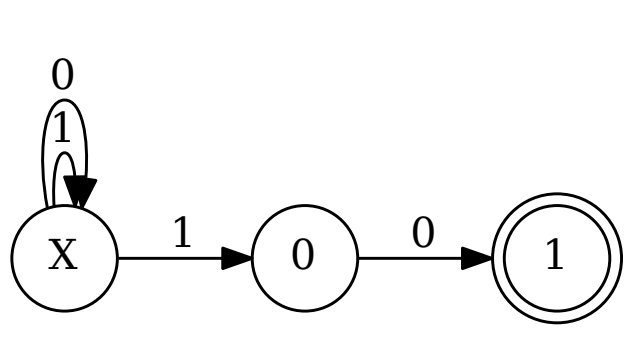
Improving Compute Utilization

Reflections

Outline

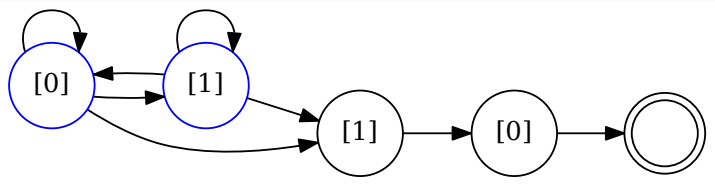
Non-deterministic Finite Automata (NFA)

- Recall CSC173
- NFA are machines that recognize regular languages
- Are "non-deterministic"
  - can be in multiple states at the same time
- Easy to parallelize
  - unlike DFA



Homogenous Non-Deterministic Finite Automata

- Traditionally, NFA match characters on their edges
- *Glushkov* NFAs or homogeneous NFAs match on the nodes instead
    - Also don't have ϵ transitions
- All children of a node are "activated" on a match
- Only activated nodes try to match a character
- Traditional NFAs can be converted to Homogeneous NFAs losslessly



Automata Processor

- Invented by Micron, the memory company
- Never made into a product, AFAIK
- Execution engine for Homogeneous NFAs
  - A non-von Neumann processor
- Ran at 133MHz, about 7.5ns per input symbol
  - Roughly 133MB/s

Putting it all together

Regular expression: `.*cat`

turns into NFA:

and is mapped to the AP as:

AP vs the GPU

Property	**AP**	**GPU**
Parallelism	49,152	163,840
Bandwidth	133MB/s	768GB/s

This is going to be easy.

- Obi Wan Kenobi, *Revenge of the Sith*

Outline

Regular vs Irregular Parallelism

Property	**Regular**	**Irregular**
Task Decomposition	Early	Late
Task Placement	Static	Dynamic
Task Activity	Static	Dynamic
Data Access Patterns	Fixed	Input-dependent
Communication Patterns	Fixed	Input-dependent
Example	MM	Sparse MM

(MM - Matrix Multiply)

NFA Task Decomposition

What shall we parallelize on?

- Each input symbol "must" be processed serially
- Multiple states can be active at the same time
- Pick states as tasks?

NFA Task Placement

How do we map tasks to GPU hardware?

- There can be thousands of states
- Each thread gets one NFA state?
- But each input byte must be processed serially
  - Can't split up an NFA across thread blocks
  - Otherwise would require global synchronization

NFA Data Access and Communication Patterns

- Each task must access:
  - input symbol (1 byte)
  - match set (1 byte in 32 bytes, depending on input symbol)
- If symbol matches:
  - children of current node must be activated (1 bit/child state)

Issues

What issues can we see in this scheme?

Activity

- Many threads are idle most of the time
  - Not all NFA states are active on every input symbol
- No useful work done
- One could use a *worklist* instead to track active states
  - Dynamic assignment of tasks to threads

Performance?

- 114 KB/second
  - Ouch!

So what did we do?

[according to the paper]

Outline

Strategies

- NewTran (pron. “neutron”)
- Match Set Compression

Outline

Strategies

- Fixed assignment for highly active states
  - Profiling
  - BFS “depth”
  - Hotstart
- Dynamic assignment for low activity states
  - Worklist

Outline

Performance

- Did we handily beat the AP?
- What kind of performance modeling is needed here?

Graph-based computations

- NFAs are graph-based computations
  - Not instruction-based programs
- Do you know of other graph-based computations?