HMMingbird: A HMM language for GPUs

Luke Cartey

Programming Tools Group, Computing Laboratory, University of Oxford

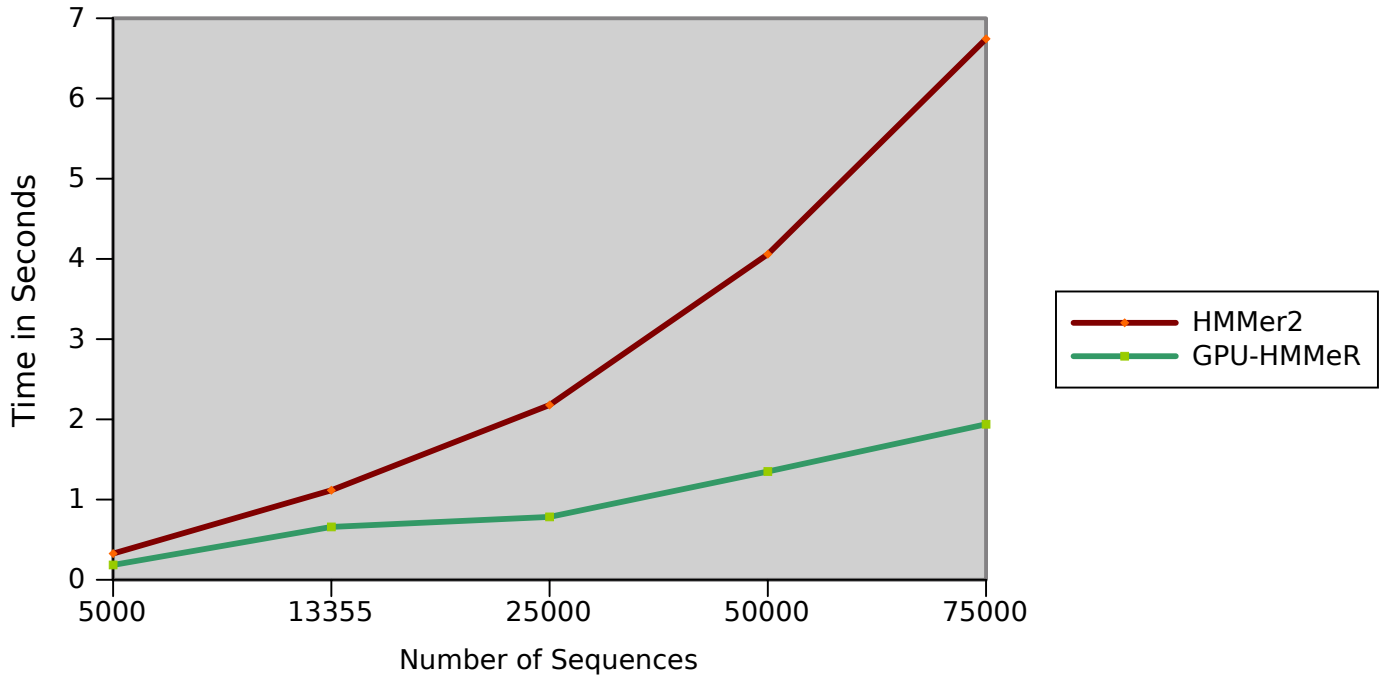
5th August 2010

Introduction

The rise of the “general purpose” graphics card.

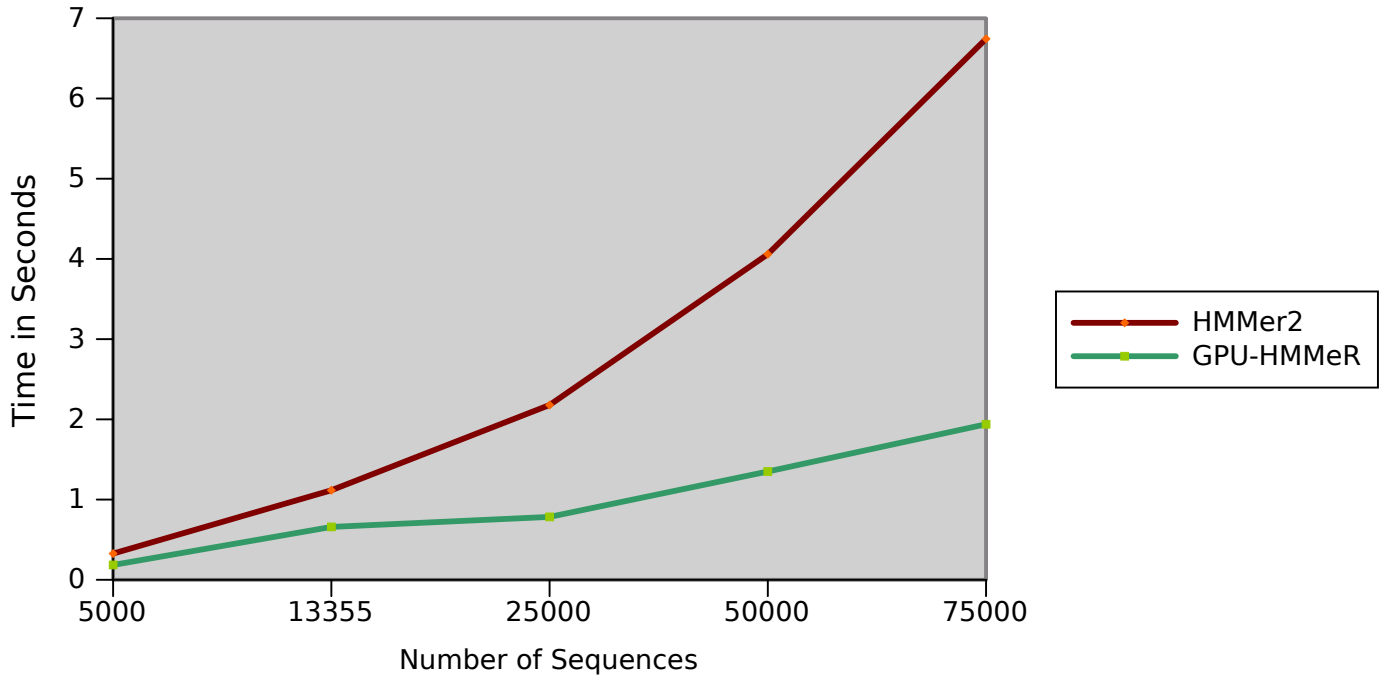
Introduction

The rise of the “general purpose” graphics card.



Introduction

The rise of the “general purpose” graphics card.



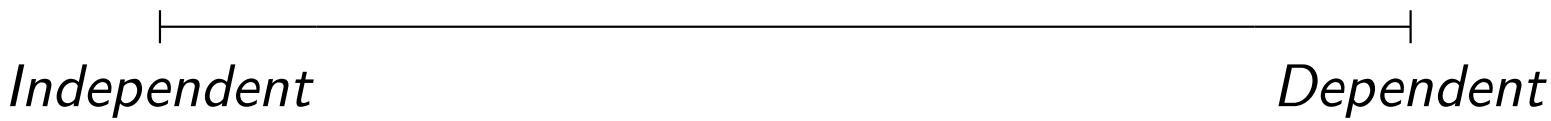
Graphics cards are a superbly effective tool for **suitable** problems.

Dependencies within Algorithms

Algorithms with few dependencies are ideal for graphics cards.

Dependencies within Algorithms

Algorithms with few dependencies are ideal for graphics cards.



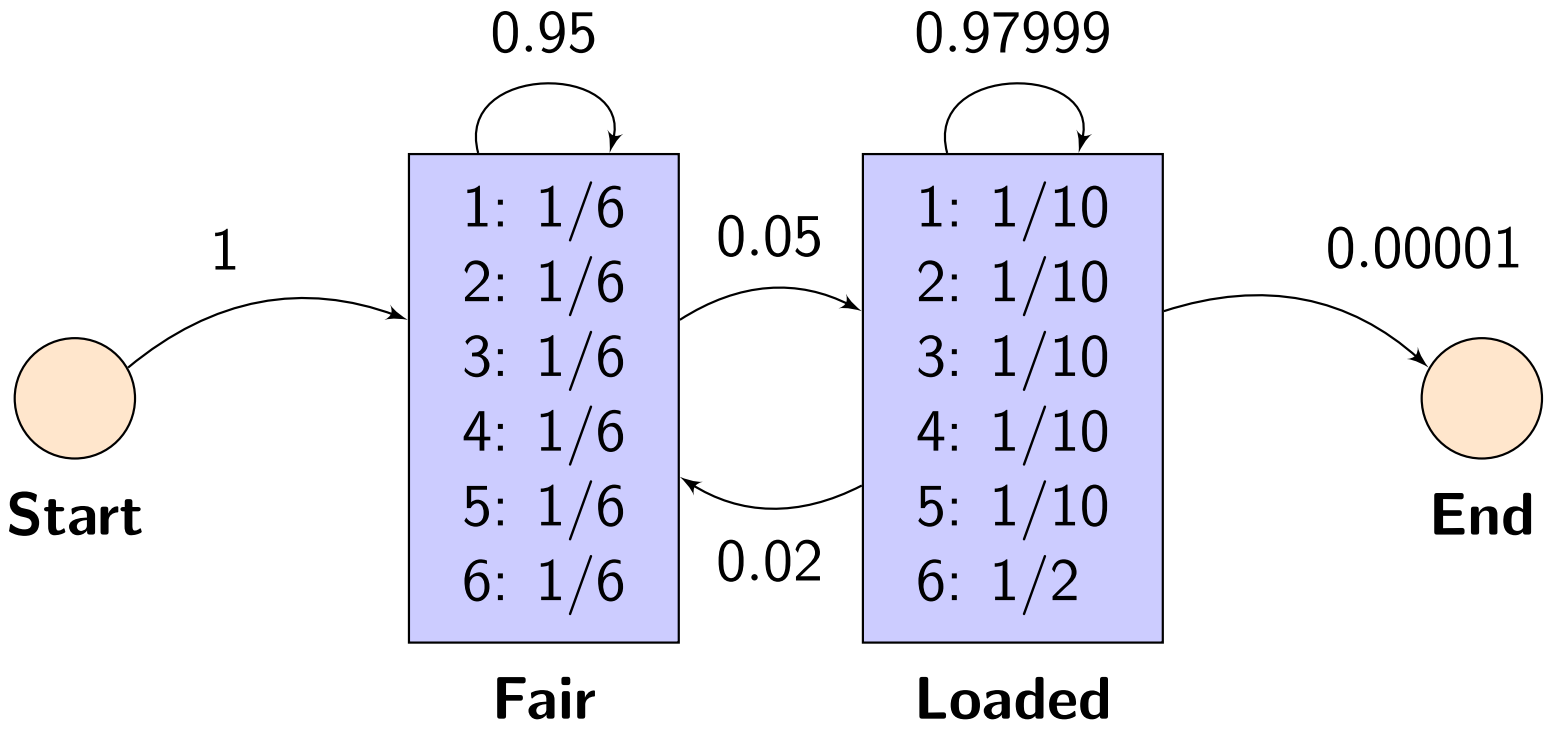
Dependencies within Algorithms

Algorithms with few dependencies are ideal for graphics cards.

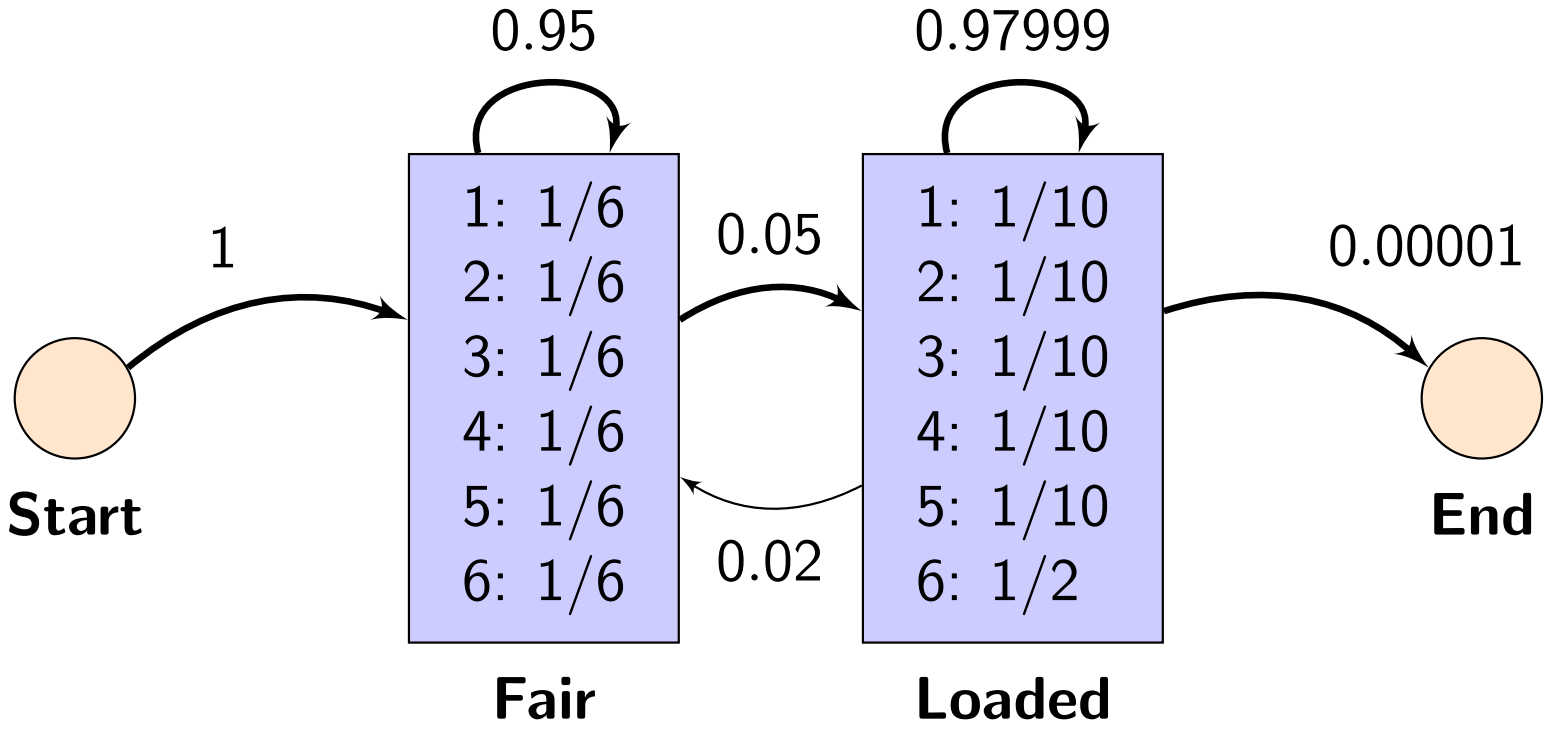
Domain Specific Languages

A **high-level** description of a domain that we can reduce to a **GPU** implementation.

Hidden Markov Models



Hidden Markov Models



Path: **Start** $\rightarrow$ **Fair(3)** $\rightarrow$ **Fair(1)** $\rightarrow$ **Loaded(6)** $\rightarrow$ **Loaded(6)** $\rightarrow$ **End**

Key Algorithms

Forward Algorithm
The probability of the model generating the output sequence.

Computed by summing over all paths generating the sequence.

Dynamic Programming Recursion

$$F(\ q\ ,\ i\ ) = \sum_{p:a_{p,q}>0} a_{p,q}\ \ e_{q,s[i]}\ F(p, i-1)$$

Dynamic Programming Recursion

- State
- Position in sequence

$$F(\ q\ ,\ i\ ) = \sum_{p:a_{p,q}>0} a_{p,q}\ \ e_{q,s[i]}\ F(p, i-1)$$

Dynamic Programming Recursion

- State
- Position in sequence
- Probability of transition from p to q

$$F(\ q\ ,\ i\ ) = \sum_{p:a_{p,q}>0} a_{p,q}\ \ e_{q,s[i]}\ F(p, i-1)$$

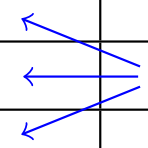
Dynamic Programming Recursion

- State
- Position in sequence
- Probability of transition from p to q
- Probability of emitting s[i] on state q

$$F(\ q\ ,\ i\ ) = \sum_{p:a_{p,q}>0} a_{p,q}\ \ e_{q,s[i]}\ F(p, i-1)$$

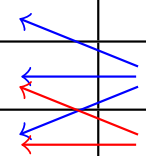
Computing a table

	Initial	'3'	'1'	'6'	'6'	...
Start						...
Fair						...
Loaded						...
End						...



Computing a table

	Initial	'3'	'1'	'6'	'6'	...
Start						...
Fair						...
Loaded						...
End						...

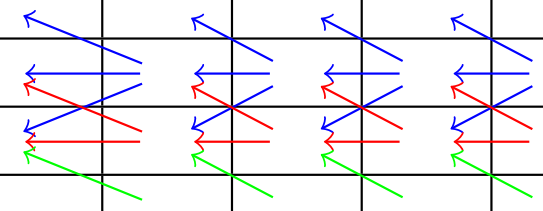


Computing a table

	Initial	'3'	'1'	'6'	'6'	...
Start						...
Fair						...
Loaded						...
End						...

Computing a table

	Initial	'3'	'1'	'6'	'6'	...
Start						...
Fair						...
Loaded						...
End						...



Key Algorithms

Backward - Computes the same value as the forward algorithm, but in reverse.

Viterbi - The probability of the most likely path for an output sequence.

Baum-Welch - Estimate the parameters of a model given some characteristic data.

Problems

- Laborious and repetitive to implement
- Efficient implementation can be difficult
- Difficult to port to new platforms

HMMingbird

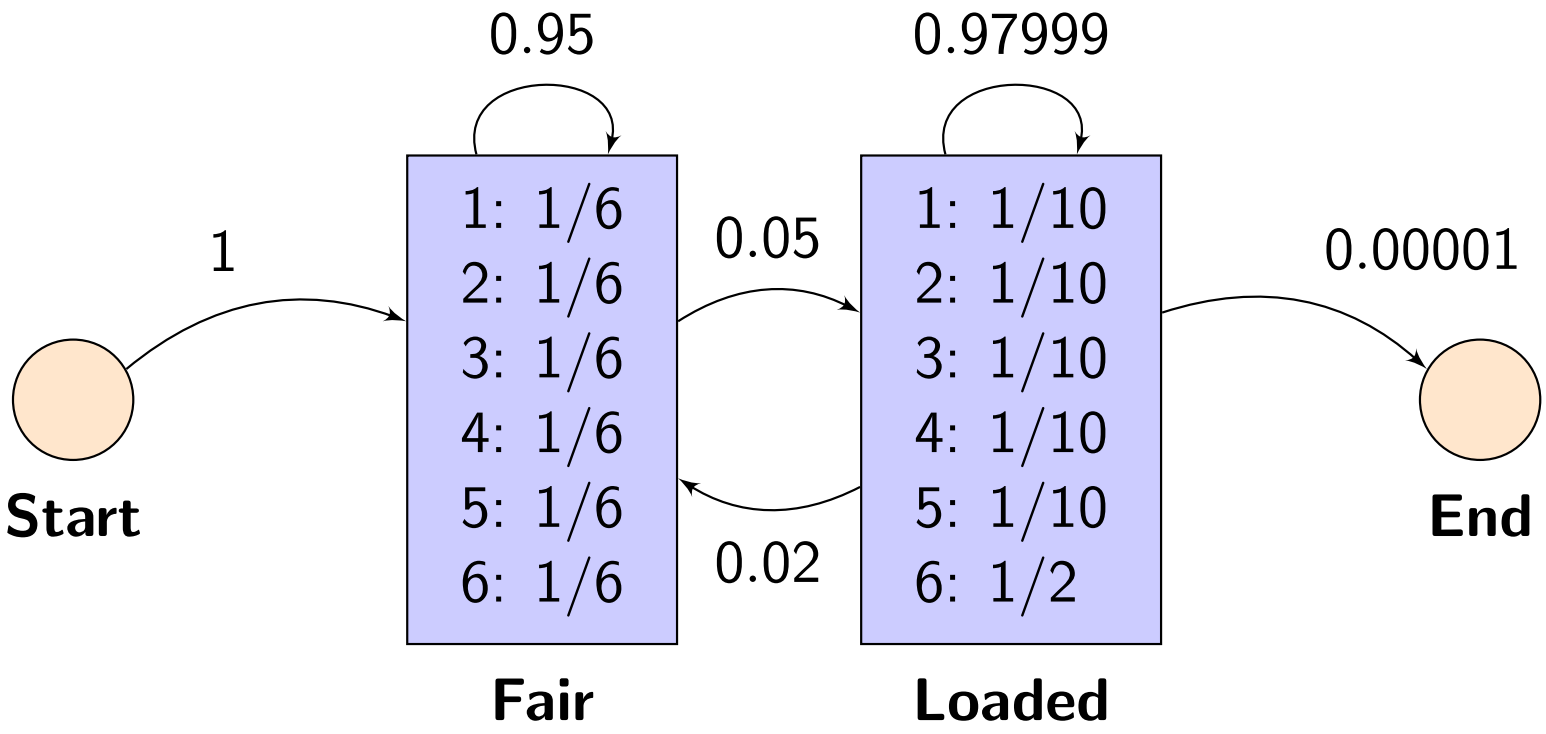
A compiler for generating efficient *Hidden Markov Model* algorithm code, targeting graphics cards using *CUDA*.

HMMingbird

A compiler for generating efficient *Hidden Markov Model* algorithm code, targeting graphics cards using *CUDA*.

- Simple description language
- Compiles to an efficient GPU implementation
- Generated code customised for each model and algorithm

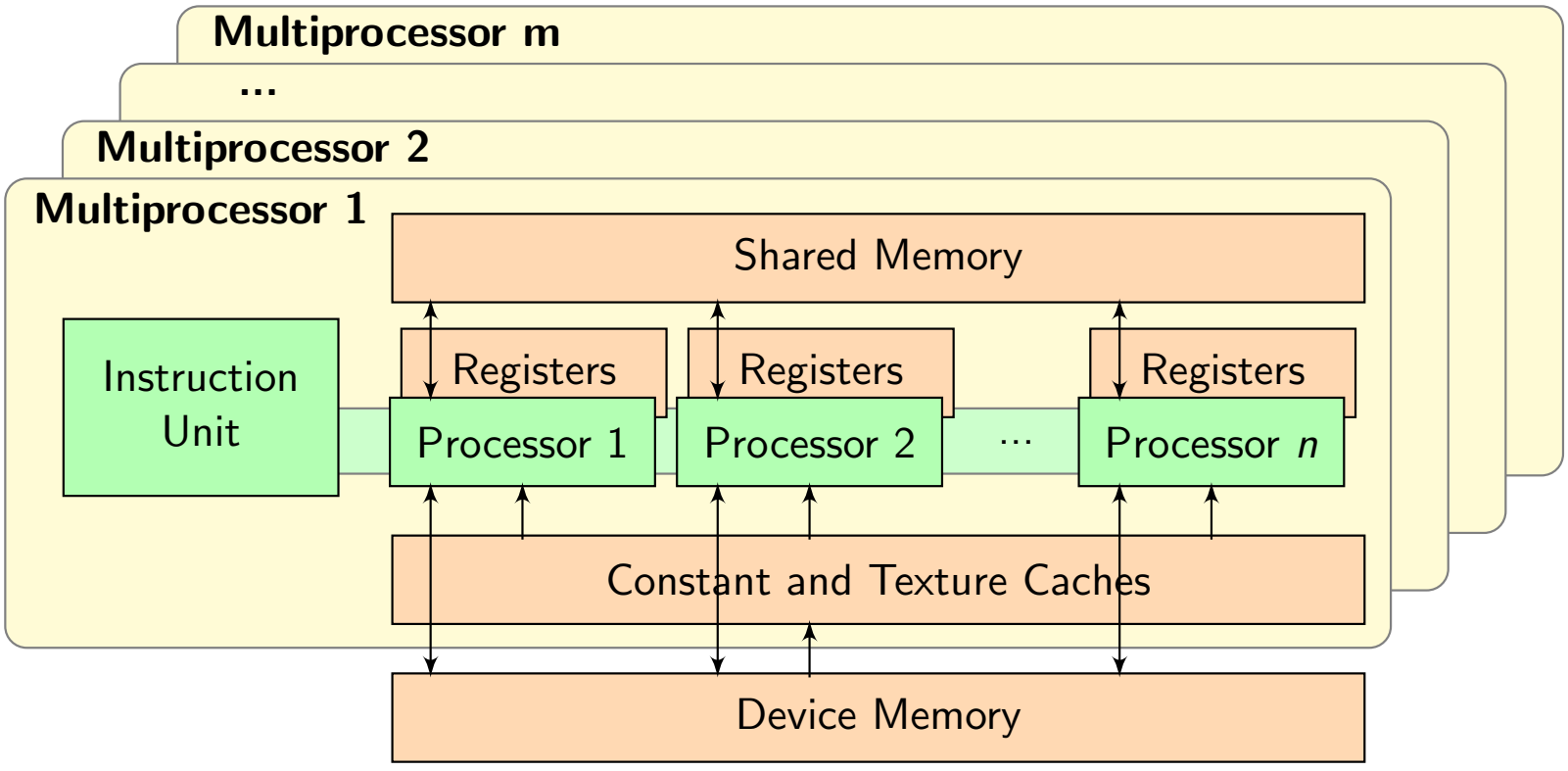
Casino Example - Diagram



Casino Example - Code

```
hmm casino {
    alphabet [1, 2, 3, 4, 5, 6];
    startstate start;
    state fair emits fairemission;
    state loaded emits loadedemission;
    endstate end;
    emission fairemission =   [1/6, 1/6, 1/6, 1/6, 1/6, 1/6];
    emission loadedemission = [0.1, 0.1, 0.1, 0.1, 0.1, 0.5];
    start -> fair 1;
    fair -> fair 0.95;
    fair -> loaded 0.05;
    loaded -> loaded 0.97999;
    loaded -> fair 0.02;
    loaded -> end 0.00001;
}
```

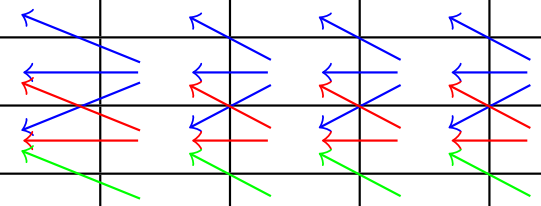
CUDA Architecture



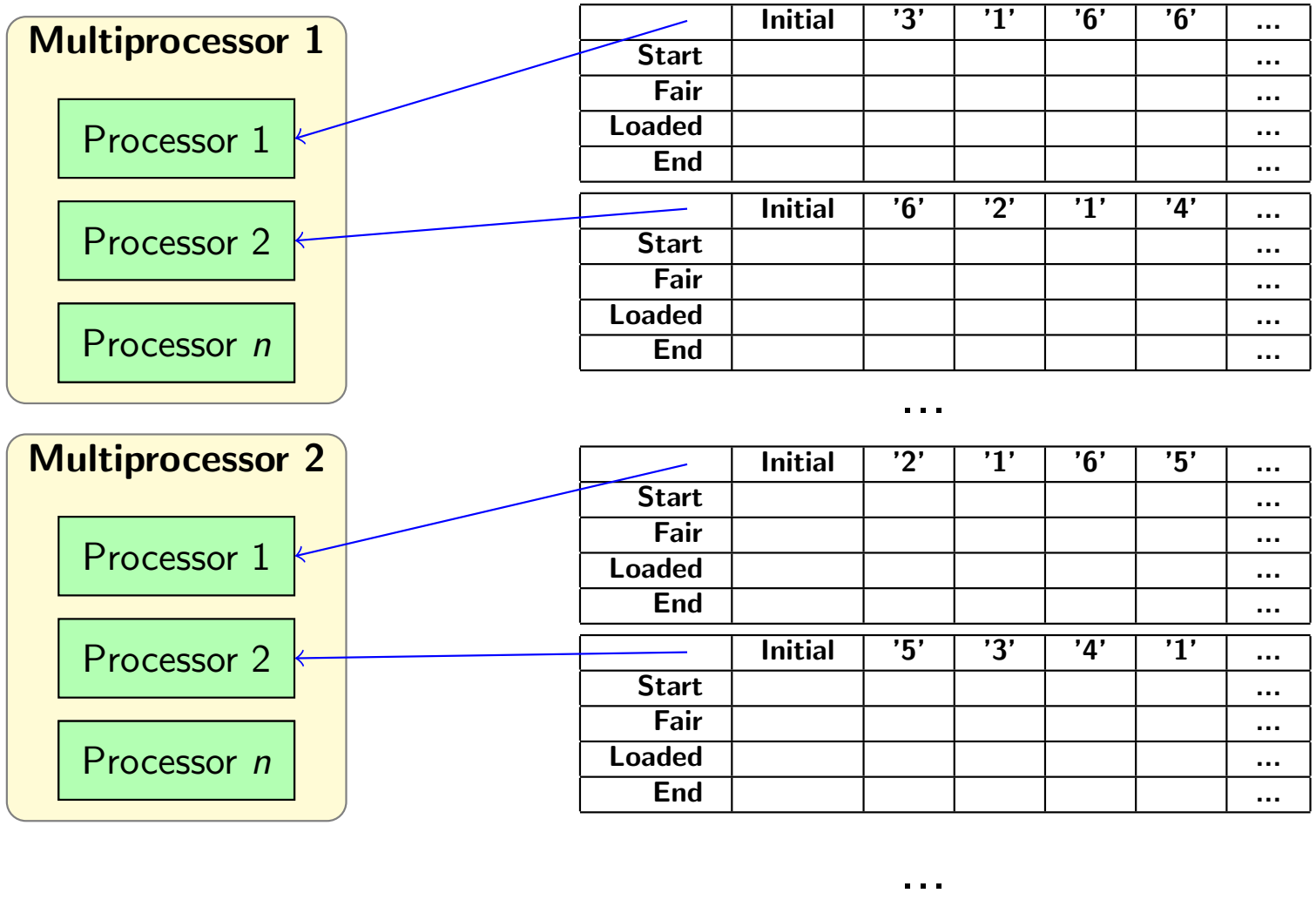
Parallel implementation

Compute many of these tables simultaneously on the graphics card.

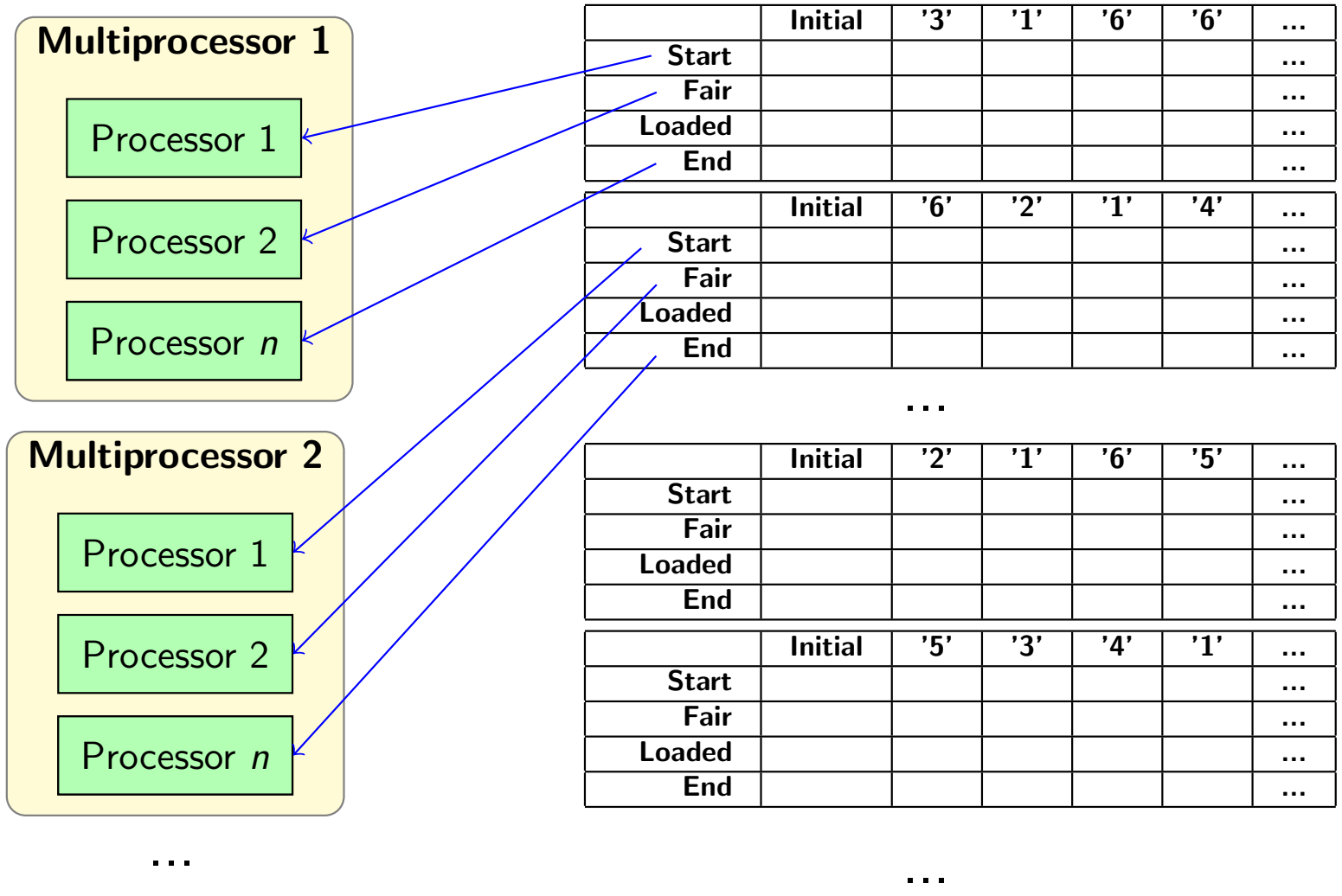
	Initial	**'3'**	**'1'**	**'6'**	**'6'**	**...**
Start						**...**
Fair						**...**
Loaded						**...**
End						**...**



Sequence-per-thread



Sequence-per-block



	Initial	'3'	'1'	'6'	'6'	...
Start						...
Fair						...
Loaded						...
End						...

	Initial	'6'	'2'	'1'	'4'	...
Start						...
Fair						...
Loaded						...
End						...

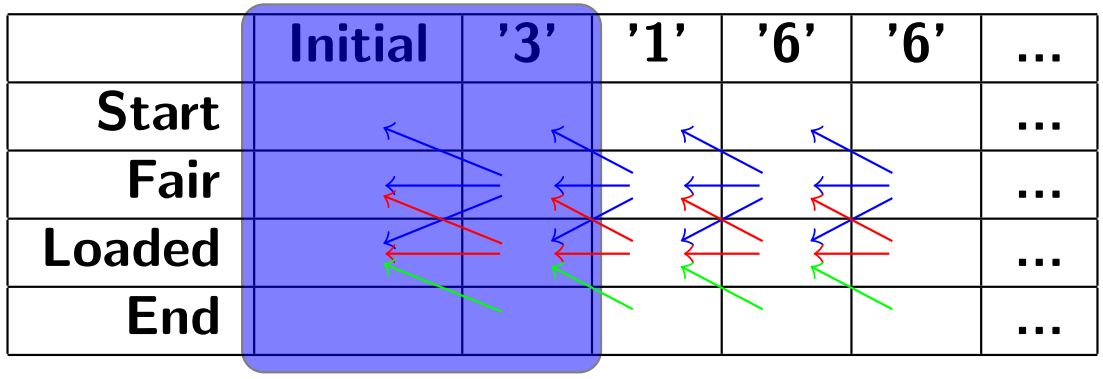
	Initial	'2'	'1'	'6'	'5'	...
Start						...
Fair						...
Loaded						...
End						...

	Initial	'5'	'3'	'4'	'1'	...
Start						...
Fair						...
Loaded						...
End						...

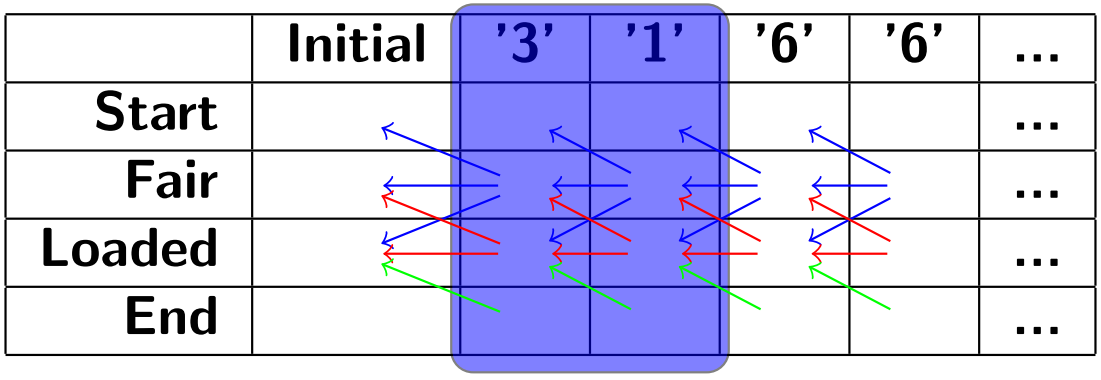
Algorithm

{s represents the dynamic programming table, indexed by state and position in sequence}
s = []
for character c in sequence at position i **do**
  parallel for state in states **do**
    $s_{state,i} = 0$
    for each input transition t_j with emission probability e_j **do**
      $s_{state,i} = s_{state,i}$ op $t_j e_{j,c} s_{startstate_{t_j},i-1}$
    end for
  end for
end for

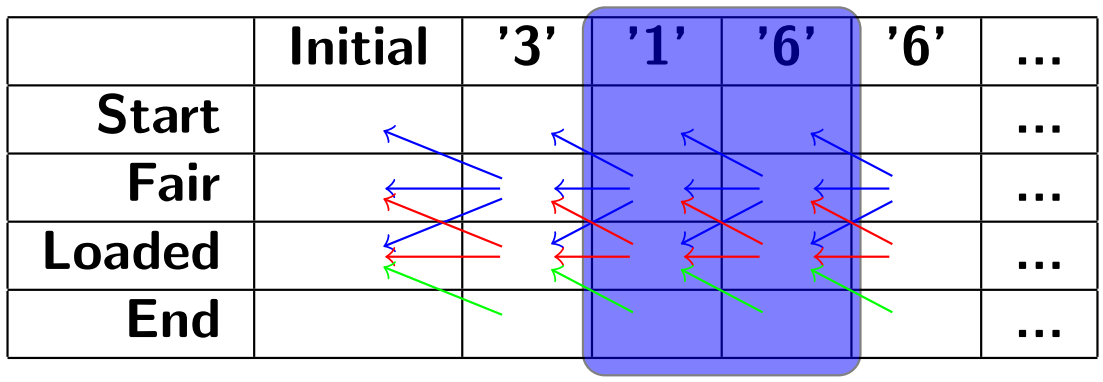
Sliding Window Optimisation



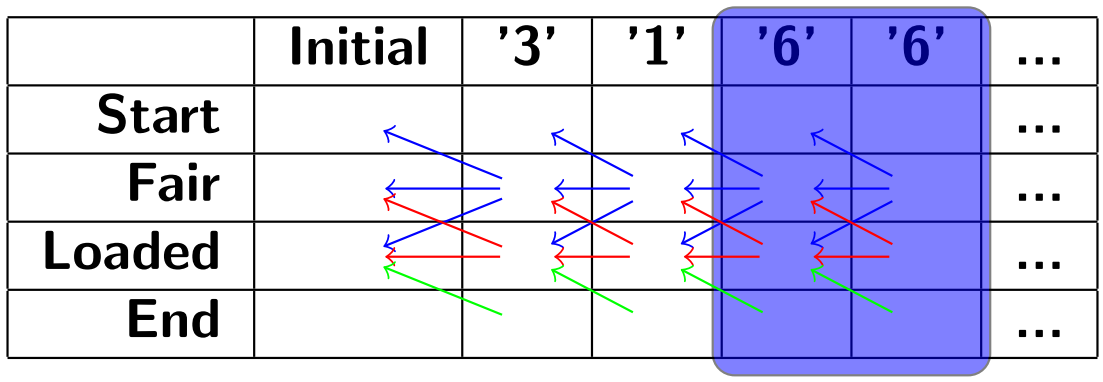
Sliding Window Optimisation



Sliding Window Optimisation

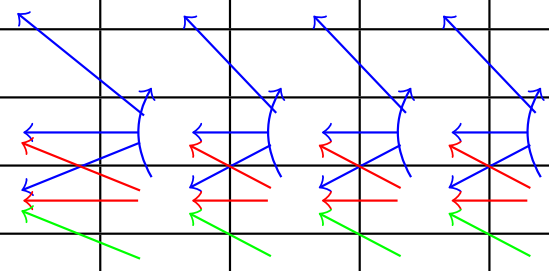


Sliding Window Optimisation

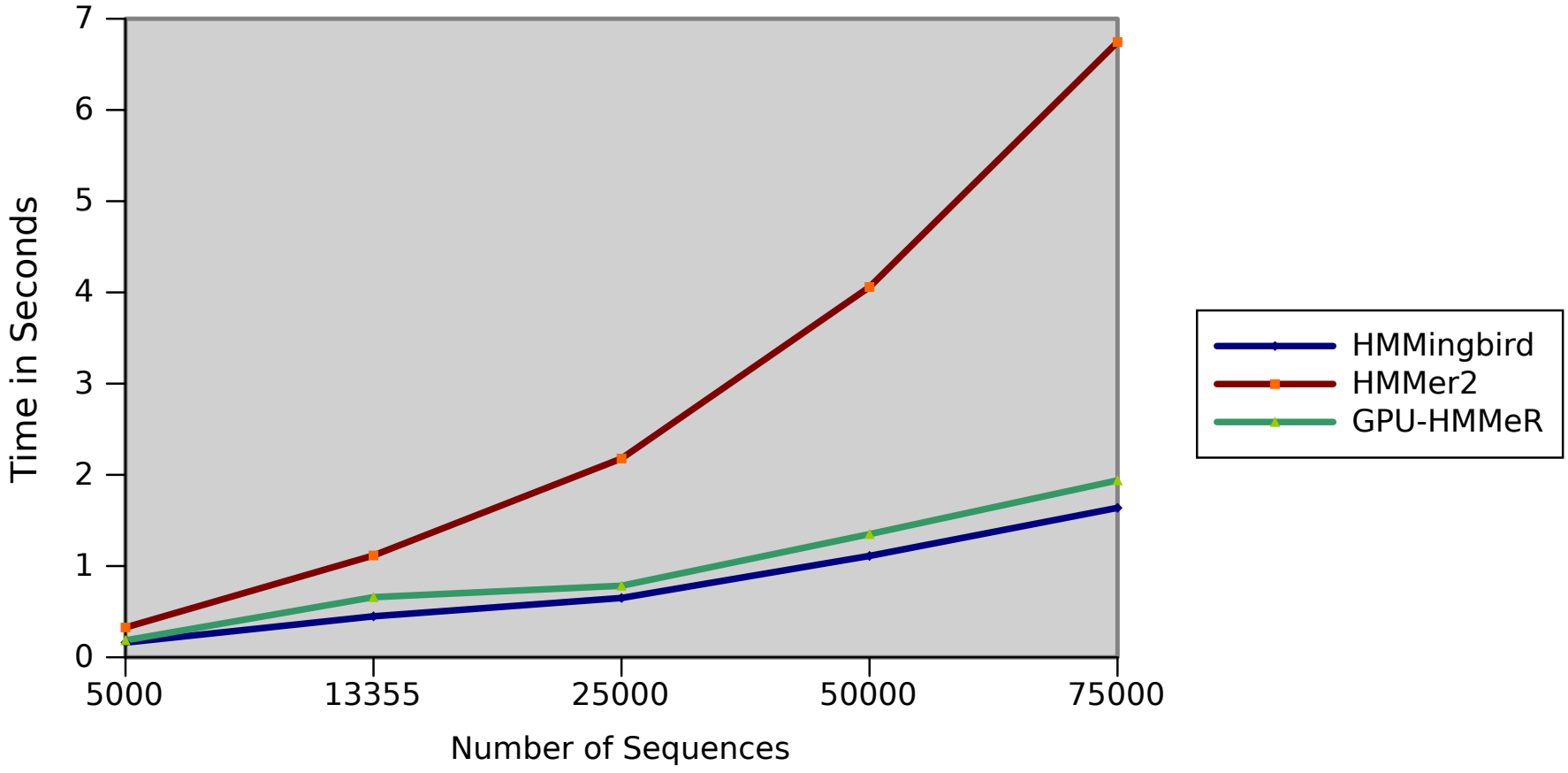


Silent States Optimisations

	Initial	'3'	'1'	'6'	'6'	...
Start						...
Silent?						...
Fair						...
Loaded						...
End						...



Results


Time in Seconds
0
1
2
3
4
5
6
7
5000
13355
25000
50000
75000
Number of Sequences
HMMingbird
HMMer2
GPU-HMMeR

Ongoing Work

Balancing transitions per thread
These are dynamic programming algorithms, so techniques should apply elsewhere.
Stochastic Context Free Grammars (CYK etc.)

Conclusion

Managing dependencies are the key to succesful parallelisation
A domain specific language can provide efficient access to parallel architectures

Further information

Information, wiki page, downloads etc.
`http://www.hmmingbird.co.uk`

Early report can be found at:
`http://www.hmmingbird.co.uk/downloads/report.pdf`