Analysis of Parallel Incremental/Decremental Graph Colouring on GPU

A Project Report

submitted by

MOHAMMED SHAMIL

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

under the guidance of

Dr. Rupesh Nasre

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

MAY 2016

THESIS CERTIFICATE

This is to certify that the thesis titled **Analysis of Parallel Incremental/Decremental Graph Colouring on GPU**, submitted by **Mohammed Shamil**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Rupesh Nasre
Research Guide
Assistant Professor
Dept. of Computer Science and Engineering
IIT Madras, 600 036

Place: Chennai

Date: 11 May, 2016

ACKNOWLEDGEMENTS

Thanks to all those who made TEX and LATEX what it is today.

ABSTRACT

KEYWORDS: Colour Quality; Compressed Sparse Row Representation; Decremental Graph Colouring; GPGPU; Graph Colouring; Incremental Graph Colouring; NP-hard; nVIDIA Cuda; Parallel Computing; Parallel Graph Algorithms; Vertex Colouring.

Graphs are a well studied and widely used data structure in the field of algorithms, programming and computing. There are a lot of interesting applications of graphs and various algorithms are built on top of the graph data structure. One of the most famous and well studied graph problems is that of graph colouring. There are a lot of different versions of graph colouring problem of which the most common ones are that of vertex colouring and edge colouring. The problem is seemingly simple, to allocate a colour to every vertex/edge of a graph so that adjacent vertices/edges don't share the same colour minimizing the number of colours used. Graph colouring is a very important and yet very challenging graph problem with ongoing active research. Graph colouring finds application in a varied range of problems including various scheduling problems like job scheduling on distributed computing systems, register allocation in compilers, pattern matching problems and solving Sudoku boards.

Though the problem is seemingly simple, it is computationally hard. The graph colouring problem we are exploring in this work, that of vertex colouring, is an NP-hard problem. The sequential approaches like greedy colouring are simply not fast enough whereas advanced approximate/randomized solutions either produce colourings of bad colour quality or aren't fast enough. Thus came the parallel approaches to Graph Colouring. Most of the parallel versions of Graph Colouring algorithms were designed with either multi-core CPUs or heavy duty super computers in mind. With the advent of General-Purpose computing on GPUs (GPGPU), we have access to cheap heavy multi-threaded parallel computing power. Our work is based on parallel computing on nVidia GPUs using Cuda programming language.

We explore different parallel graph colouring algorithms on nVIdia GPUs in this work and try to adapt them to support addition of edges, called incremental graph colouring, and deletion of edges, called decremental graph colouring. In the first section, we explore different parallel graph algorithms and adapt a couple of them, one based on *speculation* and *conflict resolution* and the other on *Vertex Independent Sets*, to work on nVidia GPUs. In the following sections, we adapt the GPU parallel colouring algorithm to support additions and deletions of edges. In the incremental part, we explore different methods to maximize parallelization while colouring newly added edges and use propagation to improve overall colour quality. In the decremental part, we explore different options to either process the vertices, on which the deleted edges were incident, on the go or to process them together and use propagation to propagate the information across the graph improving the colour quality.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS i

ABSTRACT ii

LIST OF TABLES vi

LIST OF FIGURES vii

ABBREVIATIONS viii

1 INTRODUCTION 1

1.1 Graphs and Graph Algorithms 1

1.2 Vertex Colouring 1

1.2.1 Classical Vertex Colouring Problem 2

1.2.2 Chromatic Number $\chi(G)$ 2

1.2.3 Colour Quality 2

1.2.4 Complexity 3

1.2.5 Applications 3

1.3 Parallelization 3

1.3.1 Frequency Scaling 3

1.3.2 Why Parallelization? 4

1.3.3 Parallezation of Graph Colouring 6

1.4 GPGPU 6

1.4.1 Why GPUs? 6

1.4.2 nVidia CUDA 7

1.4.3 Challenges 7

1.5 Incremental/Decremental 8

2 PARALLEL GRAPH COLOURING 10

2.1 Graph Colouring Problem 10
2.2 Related Work 10
2.3 Broad Classification of Algorithms 11
2.3.1 Vertex Independent Sets and Colouring 11
2.3.2 Speculation and Conflict Resolution 12

3 PARALLEL GRAPH COLOURING: INCREMENTAL 14

4 PARALLEL GRAPH COLOURING: DECREMENTAL 15

5 EXPERIMENTAL EVALUATION 16
5.1 Experimental Setup 16
5.2 Test Data 16
5.3 Parallel Graph Colouring on GPU 16
5.4 Incremental Parallel Graph Coloruing on GPU 16
5.5 Decremental Parallel Graph Colouring on GPU 16

6 CONCLUSION AND FUTURE WORK 17

LIST OF TABLES

LIST OF FIGURES

1.1 Graph showing Moore's Law in action, from Wikipedia, the free encyclopedia (2016). Each data point is a processor. 4

1.2 *Intel*'s transition from single core processors to multi-core processors around 2004-2005, from Sutter (2005). 5

1.3 nVidia GPU hardware model, from NVIDIA Corporation (2016). . 8

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
RTFM	Read the Fine Manual
GPU	Graphics Processing Unit
GPGPU	General-Purpose computing on Graphics Processing Units
CSR	Compressed Sparse Row

CHAPTER 1

INTRODUCTION

1.1 Graphs and Graph Algorithms

Graphs are really important mathematical concepts and in the area of computing, their various forms are widely used as data structures to aid various algorithms. Graphs are commonly used to denote relations between different entities and hence is a very important and integral part of many algorithms. On a practical level, we deal with graphs in the order of billions of nodes and edges on a daily basis. Especially with the advent of social networks and big data, a lot of active research is ongoing in the analysis and understanding of large graphs.

Many problems in the area of Computer Science, Biology etc. are solved with the help of algorithms which are based on graphs. Shortest path problem, Traveling Salesman Problem (TSP), network flow problems, vertex cover problem, graph colouring etc. are important graph-based problems with many practical applications in the real world. Our work is on Graph Colouring which is one of the most famous and well studied graph problems.

1.2 Vertex Colouring

Graph Colouring problem entails *colouring/labeling* of the vertices/edges of a graph based on some set of conditions which are to be satisfied. In other words, its a problem in which you allocate a colour/number to every vertex/edge of a graph such that a set of constraints are satified. There are different versions of Graph Colouring and the one which is of interest to us is that of Vertex Colouring.

1.2.1 Classical Vertex Colouring Problem

Vertex Colouring is the most basic version of Graph Colouring and other Graph Colouring problems can be presented as a Vertex Colouring problem. In it's classical form, Vertex Colouring is:

> ***Vertex Colouring:*** *Colouring all the vertices of a graph such that adjacent vertices have different colours. That is, there shouldn't be an edge where the incident vertices share the same colour.*

There are other forms of vertex colouring where additional conditions than the one given above need to be considered while colouring. In our work, we are concerned only with the classical form of vertex colouring which hereinafter interchangeably referred to simply as Graph Colouring.

1.2.2 Chromatic Number $\chi(G)$

A graph *G* is said to *k*-colourable, if *G* can be coloured using *k* colours. For example, from the *Four Colour Theorem*, we have that all planar graphs are 4-colourable. Also, all bipartite graphs are 2-colourable.

The *Chromatic Number* of a graph G, denoted by $\chi(G)$, is the minimum number of colours required to colour a graph. That is, $\chi(G)$, is the minimum value of all *k* for which the graph *G* is *k*-colourable. Therefore, if a graph is *k*-colourable, we have:

$$\chi(G) \leq k$$

1.2.3 Colour Quality

Colour Quality is a term used to denote how good the colouring done by a particular algorithm is. Colour Quality is said to be better for an algorithm if the number of colours used by the algorithm to colour a graph *G* is closer to its Chromatic Number, $\chi(G)$.

Mathematically, Colour Quality of a colouring is said to be better as the fraction,

$$\frac{No.\ of\ colours\ used\ by\ the\ algorithm}{\chi(G)}$$

is closer to 1.

1.2.4 Complexity

Graph Colouring is a computationally complex problem. To decide if a Graph can be coloured using *k* colours, is an NP-complete problem. Whereas, finding the Chromatic Number of a graph ($\chi(G)$) is proved to be an NP-hard problem.

There exist many algorithms like Greedy Colouring, approximation algorithms and randomized algorithms. There also exist polynomial time algorithms for some specific family of graphs. For example, it can be decided if a graph can be coloured using 2 colours by checking if it is a bipartite graph. This can be done in polynomial time using Breadth First Search (BFS).

1.2.5 Applications

Graph Colouring problem, which started as a map colouring problem (four colour theorem), finds many important real applications including but not limited to:

- Scheduling problems like job scheduling across multiple nodes in a distributed computing environment
- Register allocation problem during compilation
- Solving Sudoku
- Pattern matching applications

1.3 Parallelization

1.3.1 Frequency Scaling

Moore's law, which observes that the number of transistors present in an integrated circuit approximately doubles every two years, still stands valid. Processors, and hence

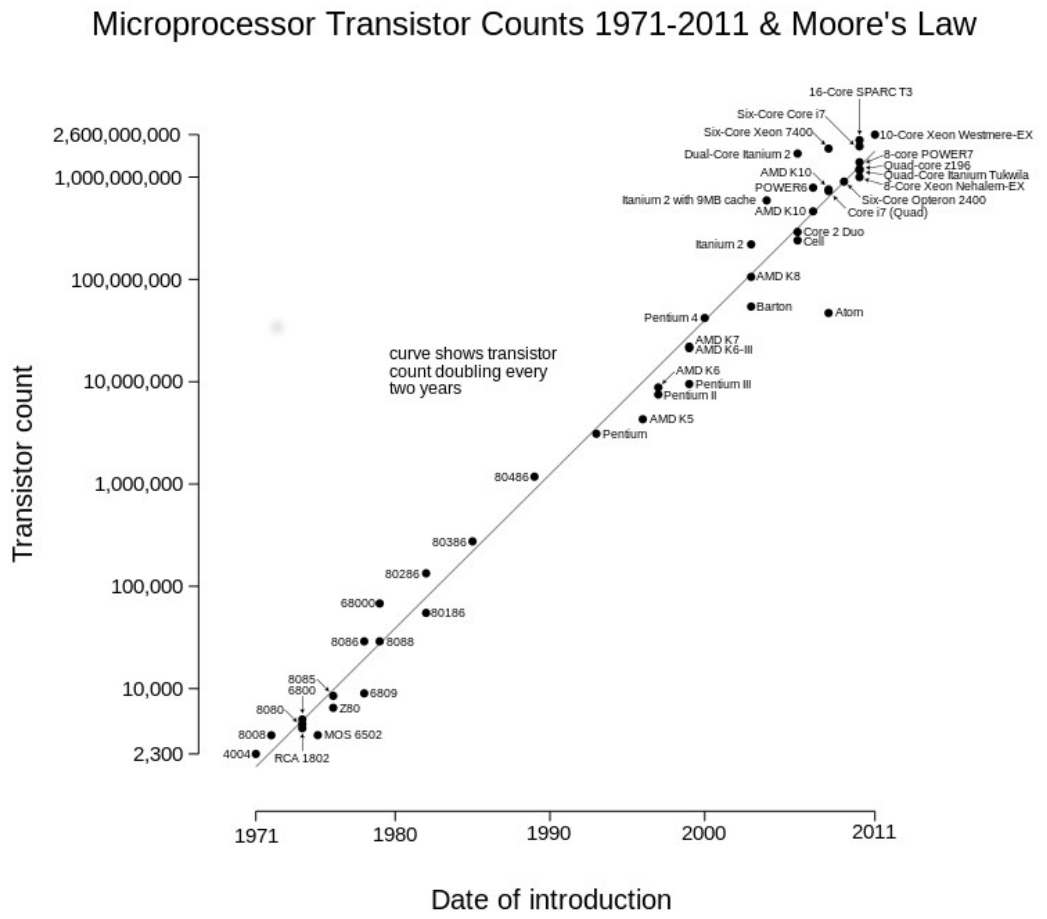

Figure 1.1: Graph showing Moore's Law in action, from Wikipedia, the free encyclopedia (2016). Each data point is a processor.

computers, have grown faster and faster over years. More and more transistors meant the processors could run faster, at a faster frequency. Processors with better and better clock speeds were introduced every year since the 1980s until around 2004 when instead of single core processors running at faster clock speeds, multi-core processors started rolling out.

1.3.2 Why Parallelization?

Around 2004, Intel and other processor manufacturing companies came to realize that frequency scaling was not practical anymore. The increase in frequency meant an increase in power consumption which inturn meant an increase in heat generation. Thus it was no longer practical to increase the clock speeds of processors. Rather, they started

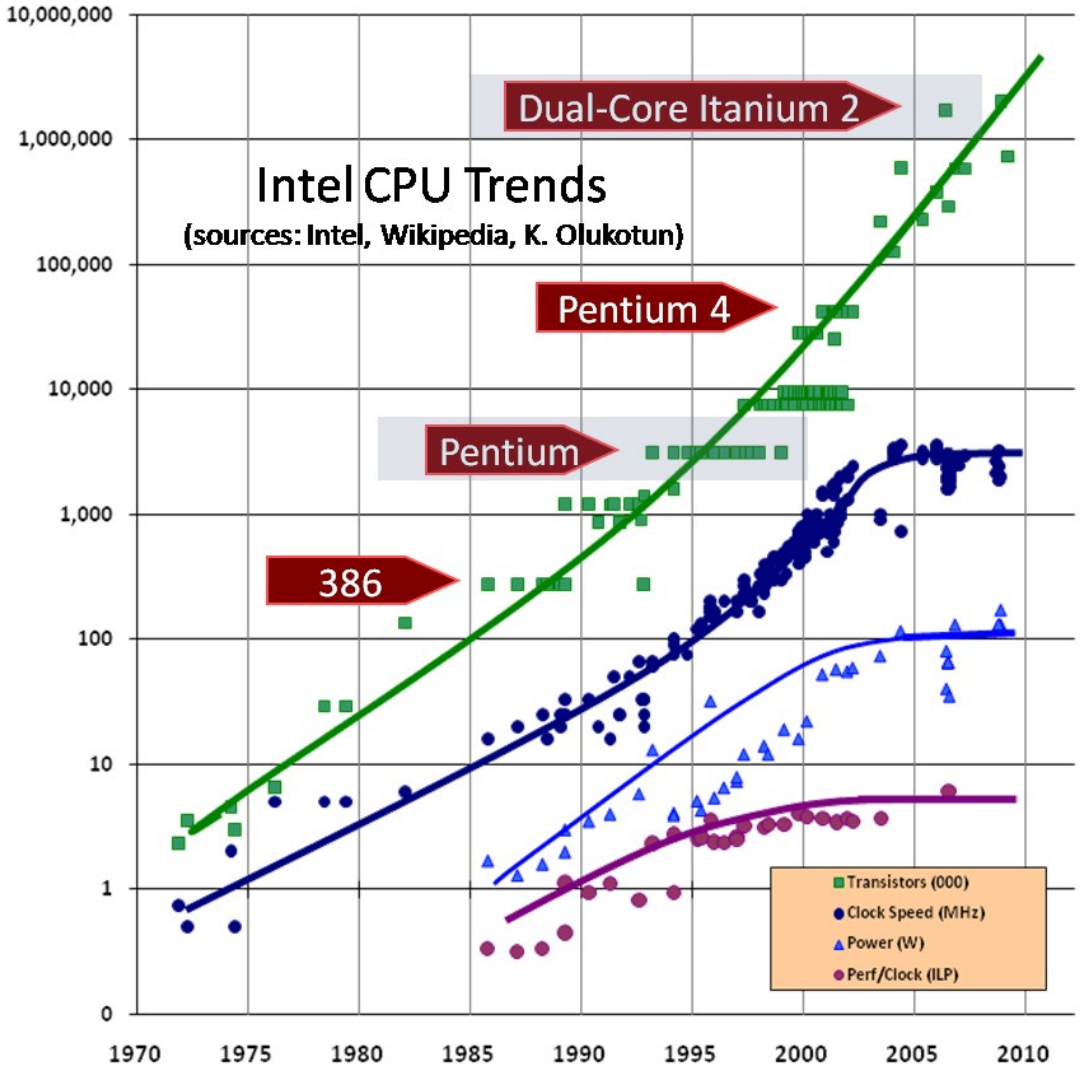

Figure 1.2: *Intel*'s transition from single core processors to multi-core processors around 2004-2005, from Sutter (2005).

making processors with the same clock speeds, but with multiple cores. Since then the computer architecture industry held fast to the paradigm of multi-core processors. This, in the case of *Intel*, is indicated in the figure 1.2.

Parallelization enables us to run programs faster by splitting the work across different cores of a processor which are ideally run in parallel. In an ideal setup, with n cores, we should see a speed up of n, which means the running time will become $1/n^{th}$ compared to the running time when run on a single core processor. Though we never really reach this ideal speedup, as stated by Amdahl's law etc., we still achieve significant speed ups.

1.3.3 Parallezation of Graph Colouring

Since the computer architecture industry made a shift to the multi-core paradigm, there had to be a shift in programming paradigm to support the newly available parallelism. Almost all the algorithms, programs etc. were designed and developed to run sequentially on a single core processor. Things have changed recently as more and more algorithms and programs are redesigned and redeveloped to make use of the newly availble parallel hardware.

As discussed earlier, Graph Colouring is a computationally complex problem. It is NP-hard to solve. Also, the approximation algorithms for colouring a graph with n vertices are also NP-hard within $n^{1-\epsilon}$ for all $\epsilon > 0$. The existing solutions are either slow or are fast but produce bad colour quality. Also, practical graphs these days are very large with billions of vertices and edges. So, since the advent of parallel programming paradigms, there have been efforts to parallelize this well celebrated graph problem though most of them were meant specifically for distributed computing setups and super computers. In our work, we focus on parallel graph colouring which can be run on parallel hardware available locally. Especially with the advent of GPGPUs, cheap massive parallelism is at a hand's reach.

1.4 GPGPU

In the domain of parallel programs and applications, one big deterrent was that the number of processor cores available for parallelism was small. Most of the multi-core processors have 32 cores at the maximum. Only super computers had a very high number of cores and they came at a price.

1.4.1 Why GPUs?

Graphics Processing Units, GPUs, have been using parallelism since their birth. They have almost always been very accessible to the normal public as they are much cheaper than super computers. They also came with thousands of cores. But they were spe-

cialized for graphics related operations. Then came the paradigm of GPGPU, General Purpose computing on Graphics Processing Units. And with that, it was now possible to run regular operations and not just graphics related operations on the GPU. GPGPU brought with it easy, cheap access to massive parallelism.

1.4.2 nVidia CUDA

nVidia, one of the biggest players in the GPUs market, introduced its famous parallel computing platform, CUDA, in 2006, thus enabling easy GPU based parallel acceleration. In our work, we use CUDA C to parallelize graph colouring. CUDA lets us harness the power of thousands of cores in the CUDA enabled nVidia GPUs.

Architecture

In an nVidia GPU, as shown in the figure 1.3, there are multiple streaming multiprocessors, SMs, and each of these multi processors have thousands of cores/processors in it. Functions to be executed on the GPU are called Kernels and Kernels, once invoked, spawn the required number of threads as blocks of threads which are then executed across SMs. All threads in a block have access to the shared memory inside the SM in which that block is executed.

1.4.3 Challenges

Though, GPUs let us access thousands of threads easily, it comes with a cost. GPU memory and CPU memory are mutually exclusive. So, we have to first copy all the data which are to be processed by the GPU threads to the GPU before invoking the kernels. As the communication between the CPU and the GPU is enabled through the PCI-Express port, which is not super fast, there is a cost for transferring data between the CPU and the GPU, the so called *memory latency*. This data transferring cost is one of the biggest overheads in GPU computing. So, as a GPU programmer, one must try to reduce data transfer between the CPU and the GPU.

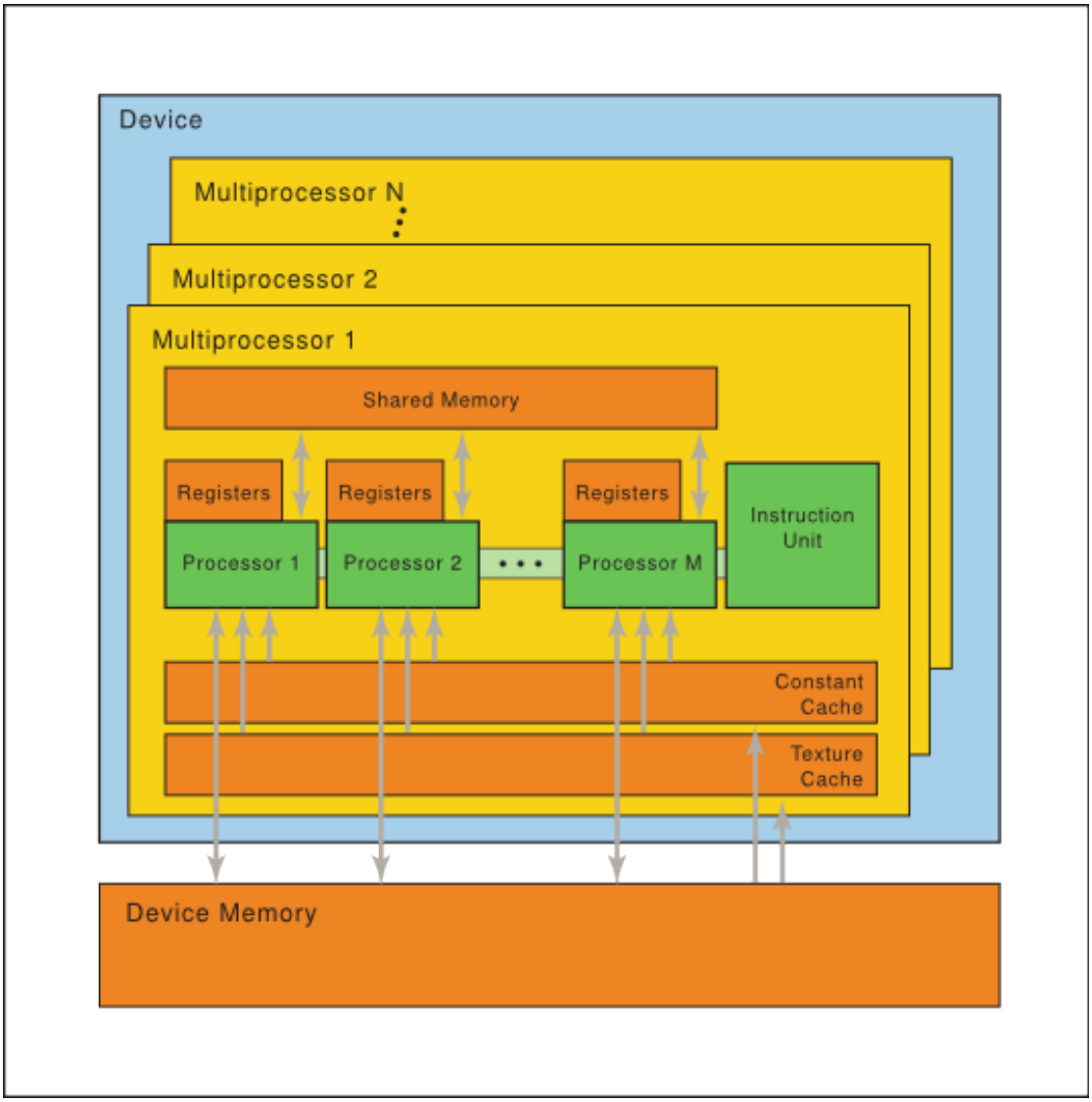

Figure 1.3: nVidia GPU hardware model, from NVIDIA Corporation (2016).

1.5 Incremental/Decremental

As we discussed already, Graph Colouring is a very important algorithm on graphs. We have so many practical applications for the same. But most of the practical graphs are dynamic in nature. Vertices and edges are added and deleted often. But the number of these changes are very small compared to the size of the full graph. So, it follows that it is not wise to rerun the graph colouring algorithm on the entire graph every time some vertices/edges are added or deleted. Our work on Incremental/Decremental graph colouring tries to take care of precisely the same.

In our work, we try to re-colour only the relevant parts of the graphs on addi-

tion/deletion of edges instead of re-colouring the entire graph. We consider only addition/deletion of edges as deletion of a vertex is considered as the deletion of all edges incident on that vertex. We consider different implications of Incremental/Decremental colouring such as the amount of time we save vs maintaining/improving the colour quality.

CHAPTER 2

PARALLEL GRAPH COLOURING

We have established why we are interested in the paradigm of parallel programming and why we want to parallelize graph colouring.

2.1 Graph Colouring Problem

The problem we are concerned with is that of *1-distance Vertex Colouring* or simply, *Vertex Colouring*.

> ***Parallel 1-distance Vertex Colouring:*** *Colouring all V vertices of a graph G(V,E) in parallel, such that vertices at a distance of 1 edge, adjacent vertices, don't share the same colour.*

2.2 Related Work

Graph Colouring is a well studied problem and there have been so many works on the same over years. But most of them were regarding sequentially solving the problem using various paradigms, like that of semi-definite programming, integer programming etc.

Recently there have been some parallel approaches to the same, but most of them like the works by ÇAtalyüRek *et al.* (2012) are based on Super Computers or other expensive hardware. There have been only a few work done regarding parallel graph colouring on GPUs like Grosset *et al.* (2011), Naumov *et al.* (2015) and Sengupta (2014) and none regarding Incremental/Decremental versions as far as the author understands.

2.3 Broad Classification of Algorithms

As we are dealing with NVIDIA GPUs, we are concerning ourselves with algorithms pertaining to shared memory architectures only. Most of the parallel colouring algorithms based on shared memory architecure can be broadly classified into two of the following categories:

2.3.1 Vertex Independent Sets and Colouring

Algorithms falling under this category work in two phases:

1. Vertex Independent Sets: Find VIS
2. Colouring: Colour the VIS found without conflicts

This set of algorithms relies upon finding Vertex Independent Sets of vertices and colouring them in parallel. Most of the earlier parallel algorithms developed for graph colouring were based on this idea.

Vertex Independent Set (VIS): A vertex independent set of a graph G is a set of vertices who don't have any edges amongst each other.

Mathematically, γ is a valid Vertex Independent Set of a graph $G(V, E)$ if

$$\gamma \subseteq V$$

and

$\forall\, v_i, v_j \in \gamma$, $e \in E$, if v_i is incident on e, then $\forall\, j \neq i$, v_j is not incident on e.

The idea is pretty straight forward. These are iterative algorithms where, in each iteration, you find a Vertex Independent Set of the given graph and colour all of the vertices in that VIS with a single colour. The process is continued with different colours until there are no more vertices to be coloured. Both the steps, finding VIS and colouring the vertices in that VIS, can be done in parallel. This category of algorithms roughly work as explained in Algorithm 1.

Algorithm 1 Vertex Independent Sets and Colouring

```
1: procedure VISPARALLELGRAPHCOLOURING(G(V,E))
2:     Initialization                ▷ Initialize all the variables and other data structures
3:     currentColour ← 1
4:     while V ≠ φ do                ▷ Run until all the vertices are coloured
5:         γ ← a VIS of G(V,E)       ▷ VIS can be found in parallel
6:         for each v ∈ γ do         ▷ This loop can be run in parallel
7:             colour[v] ← currentColour
8:         V ← V − γ
9:         currentColour ← currentColour + 1
```

2.3.2 Speculation and Conflict Resolution

Algorithms falling under this category work in two phases:

1. Speculation: Colour the graph based on some pre-existing knowledge possibly generating conficts
2. Conflict Resolution: Resolve the conflicts possibly generated in the first phase

The first category of algorithms relied upon finding Vertex Independent Sets iteratively so that we can colour the vertices in the VIS found in each step without any conflict. This second category of algorithms instead let us commit some mistakes, or rather conflicts, in our colouring. That is, it saves us from finding a VIS in each step, instead we colour the graph using some pre-existing knowledge like existing colouring of the graph or some structural information regarding the graph.

So, in the first phase, instead of finding a VIS and colouring just the vertices in that VIS without any conflict in an iteration, we speculate the colours of the entire graph with some pre-existing knowledge and possibly commit conflicts. The possible conflicts inflicted in this first phase are rectified in the second phase in which we find the conflicts and resolve them. For practical reasons, the first phase is done in parallel and the second phase is done sequentially or partially in parallel. This category of algorithms roughly work as explained in Algorithm 2.

Algorithm 2 Speculation and Conflict Resolution

```
    procedure SPECCRPARALLELGRAPHCOLOURING(G(V,E))
2:      Initialization
    speculation:
4:      for each v ∈ V do                                   ▷ Can be done in parallel
            colour[v] ← speculatedColour
6:  conflict resolution:                                    ▷ Done serially or partially in parallel
        for each v ∈ V do
8:          if colour[v] has a conflict then                ▷ Conflicts can be found in parallel
                colour[v] ← a new colour which resolves the conflict    ▷ Greedy?
```

CHAPTER 3

PARALLEL GRAPH COLOURING: INCREMENTAL

CHAPTER 4

PARALLEL GRAPH COLOURING: DECREMENTAL

CHAPTER 5

EXPERIMENTAL EVALUATION

5.1 Experimental Setup

5.2 Test Data

5.3 Parallel Graph Colouring on GPU

5.4 Incremental Parallel Graph Coloruing on GPU

5.5 Decremental Parallel Graph Colouring on GPU

CHAPTER 6

CONCLUSION AND FUTURE WORK

REFERENCES

1. **ÇAtalyüRek, ı. V.**, **J. Feo**, **A. H. Gebremedhin**, **M. Halappanavar**, and **A. Pothen** (2012). Graph coloring algorithms for multi-core and massively multithreaded architectures. *Parallel Comput.*, **38**(10-11), 576–594. ISSN 0167-8191. URL http://dx.doi.org/10.1016/j.parco.2012.07.001.

2. **Grosset, A. V. P.**, **P. Zhu**, **S. Liu**, **S. Venkatasubramanian**, and **M. Hall** (2011). Evaluating graph coloring on gpus. *SIGPLAN Not.*, **46**(8), 297–298. ISSN 0362-1340. URL http://doi.acm.org/10.1145/2038037.1941597.

3. **Naumov, M.**, **P. Castonguay**, and **J. Cohen** (2015). Parallel graph coloring with applications to the incomplete-lu factorization on the gpu. *NVIDIA Technical Report*, **001**. URL https://research.nvidia.com/publication/parallel-graph-coloring-applications-incomplete-lu-factorization-gpu.

4. **NVIDIA Corporation** (2016). Cuda toolkit documentation. URL http://docs.nvidia.com/cuda/parallel-thread-execution/#ptx-machine-model. [Online; accessed on 22 April, 2016].

5. **Sengupta, S.**, Parallel graph coloring algorithms on the gpu using opencl. *In Computing for Sustainable Global Development (INDIACom), 2014 International Conference on*. 2014.

6. **Sutter, H.** (2005). The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, **30**(3). URL http://www.gotw.ca/publications/concurrency-ddj.htm.

7. **Wikipedia, the free encyclopedia** (2016). Moore's law. URL https://en.wikipedia.org/wiki/Moore%27s_law#/media/File:Transistor_Count_and_Moore%27s_Law_-_2011.svg. [Online; accessed on 22 April, 2016].