TECH-CHRONICLE
AN INTERNATIONAL E-JOURNAL ON
EMERGING TRENDS IN SCIENCE, TECHNOLOGY AND MANAGEMENT

Force Directed Placement: GPU Implementation

Tripti Sharma
Department Of Information Technology,
Maharaja Surajmal Institute Of Technology,
New Delhi,India-110058
triptionline@yahoo.com,

Abstract:
Graph layout has had important applications in many areas of computer science. When dealing with machine generated data, we often tend to see the data to have better understanding of a structural form. Many such data can be represented in the form of graphs. By laying out a graph, we can untangle information and intuitively show relations of objects.Automated graph drawing remains a difficult placement and layout problem. This problem is difficult in part due to the complexity of formulating good algorithms to draw graphs which are aesthetically pleasing for human visualization. In this paper, we have implemented a Force-Directed Placement algorithm using Python environment that will solve the graph layout problem by using an energy minimization technique. This work aims to demonstrate the performance advantage of a GPU implementation as compared to a CPU implementation. For CPU Python is the programming platform chosen and for GPU NVIDIA PyCUDA is the platform chosen. The GPU implementation was able to achieve up to a 55-60x speed-up as compared to the CPU.

Keywords-Force directed placement, PYCUDA, PYTHON

1. INTRODUCTION

Graphs are often used to encapsulate relationship between objects.Graph drawing enables visualization of such relationships.The usefulness of this relationship is dependent on whether the drawing is aesthetic.While there are no strict criteria for aesthetics of a drawing,it is generally agreed for example,that such a drawing has minimal edge crossings with vertices evenly distributed in space,and with symmetry that may exist in the graph depicted. Graph layout algorithms use a certain characteristics and laws to determine if they are getting closer or farther from being aesthetically pleasing and use an iterative method to progress towards the final graph layout solution.The graphing problem consists of a number of *elements*(or graph nodes) which are interconnected in somemanner. The connections between elements are referredto as understand way by generating a graphical representation.Applications of graph drawing include VLSI circuitdesign, network relationship analysis, cartography, andbioinformatics [1].

PYTHON-Programming platform for CPU

python is a general-purpose, high-level programming language whose design philosophy emphasizes code readability[2].
Python's syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C, and the language provides constructs intended to enable clear programs on both a small and large scale.Python supports multiple programming paradigms, including object-oriented, imperative and functional programming styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts. Using third-party tools, Python code can be packaged into standalone executable programs. Python interpreters are available for many operating systems.

Python over other languages

Python is an interpreted high-level language and interpreters are available for Windows, Mac OS X and UNIX. Thus the same code can be executed upon these different platforms with no changes to code. By contrast, C++ must be compiled separately for each platform and all machine-specific code must be filtered out of each compilation, which puts an increased
workload upon the programmer. As an interpreted language Python is more direct and convenient, it can be used interactively or can be used to build complete programs, without any of the lengthy compilation and linking procedures involved in C++.
Python's data types are high-level and extremely flexible, unlike the strong data typing that is strictly enforced by C++. As such there is no need to declare variables before using them. However, the flexibility comes at a cost in increased memory consumption.

The flexibility of the language and high-level nature of the data types means Python is much easier to learn than C++, and therefore more accessible to non-programmers.

The main disadvantage of Python compared to C++ is that it doesn't perform well. However, the same can be said of all interpreted languages, including Java. C++ will always perform better because code is compiled to native machine code, but requires more effort on the part of the programmer. However, the results are comparable to that of low-level assembly.

ABOUT NVIDIA CUDA GPU

CUDA (an acronym for Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA. CUDA is the computing engine in NVIDIA graphics processing units (GPUs) that is accessible to software developers through variants of industry standard programming languages.

Programmers use 'C for CUDA' (C with NVIDIA extensions and certain restrictions), compiled through a Path Scale Open64 C compiler to code algorithms for execution on harnessing the power of

the GPU. With millions of CUDA-enabled GPUs sold to date, software developers, scientists and researchers are finding broad-ranging uses for CUDA, including image and video processing, computational biology and chemistry, fluid dynamics simulation, CT image reconstruction seismic analysis, ray tracing, and much more. Generally Computing is evolving from "central processing" on the CPU to "co- processing" on the CPU and GPU[3]. The CUDA SDK used is CUDA v4.2 with compute capability 2.1.S

PYCUDA-Programming platform for GPU

PyCUDA gives you easy, Pythonic access to Nvidia's CUDA parallel computation API. Several wrappers of the CUDA API already exist– so why the need for PyCUDA?[4]

- Object cleanup tied to lifetime of objects. This idiom, often called RAII in C++, makes it much easier to write correct, leak- and crash-free code. PyCUDA knows about dependencies, too, so (for example) it won't detach from a context before all memory allocated in it is also freed.
- Convenience Abstractions like the one in PyCuda's pycuda.compiler.SourceModule.
- Completeness. PyCUDA puts the full power of CUDA's driver API at your disposal, if you wish.
- Automatic Error Checking. All CUDA errors are automatically translated into Python exceptions.
- Speed. PyCUDA's base layer is written in C++, so all the niceties above are virtually free.
- Enables run-time code generation (RTCG) for flexible, fast, automatically tuned codes[5].
- Added robustness: automatic management of object lifetimes, automatic error checking[5]

2. RELATED WORK

Improved heuristics for graph layout have been developed for both force directed placement and simulated annealing approaches. Force directed placement algorithms for graph layout are primarily based on Hooke's law, but have deviated from strict physical modelling to better matchthe requirements of different graphing applications as described by Eades spring-mass equations [6] and later adapted by

Fruchterman and Reingold to emulat the GPU. CUDA is NVIDIA's parallel computing architecture. It enables dramatic increases in computing performance by particlephysics in a simulated annealing algorithm [7].Graph placement and layout remains an active area of research, with Frishman and Tal [8] [9] using GPUs to speed-up1 incremental graph layout to provide results 8x faster than a CPU, and proposing different approaches to partition the computational work to improve parallelism.For our work, we chose to use the simpler heuristics proposed by Eades which lend themselves well to parallelization on the GPU. The focus of our work was to demonstrate that a significant runtime advantage could be achieved on a GPU even for a simple force directed placement algorithm by increasing parallelism and optimizing memory bandwidth utilization.

3.ALGORITHM

Force-based/Force-directed algorithms

In this model, each element in the graph is modelled as amass, with edges between graph elements modelled assprings as shown in Figure 1. Values for the"gravitational force" and "spring constant" can be varied to produce different results. Further, additional forces(electrical charges, logarithmic springs, moments,damping effects) can be used to alter the aesthetic. Thealgorithm takes this model and chooses some arbitraryinitial state. It then iteratively simulates the movement ofeach element (or "mass") by through increments of time(steps) until the model reaches steady state. On eachiteration, the forces acting on each element in the modelare computed and the element's "position" and "velocity"are then adjusted[10].

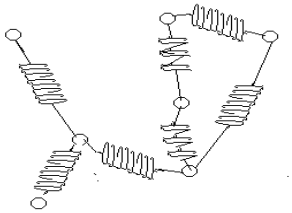


FIGURE 1:Spring-Mass system

The relationship between the force exerted by the springand the distance that the spring is stretched is describedby Hooke's Law [6] in Equation 1.

F = -kx…… (Eq. 1)

where F is the restoring force of the object, k is the spring constant, and x is the distance the spring has been extended compressed from its equilibrium position.

Combined with the damping force (friction) and the acceleration of the object, the new position of an object can be calculated [7].

Force = Fspring + Fdamping = - kx – bv…… (Eq. 2)

where b is damping constant and v is velocity.

x'' = -kx – bx'…….. (Eq. 3)

where x is acceleration, x is position of object, and x' is the velocity of the object.

4.IMPLEMENTATION

The force-directed placement algorithm described above was implemented for the CPU and GPU. Thesame framework was used for both implementations toprocess input graph files and store the node and edge information in atwo-dimensional array. The placement algorithms for

CPU and GPU implementations are run, and theframework writes out the graph information (includingplacement information).

The outer loop of our implemented algorithm is shown in .

```
Read_input_graph()
Do {
        Calculate_forces()
        update_velocity()
        update_position()
        Calculate_kinetic_energy()
        } While (kinetic energy > stablethreshold);
Write_output_graph()
```

Calculate Forces: To determine the overall force vector

for each node, the force vectors (repulsive and attractive) of all of the other nodes acting upon that node are added together. The complexity of this function is O(n2)

Update Velocities: For each node, the velocity in the x and y direction is calculated based on the force applied tothe node for the current timestep, and a damping factor isapplied. The complexity of this function is O(n).

Update Position: For each node, the position of the nodeis updated by adjusting the initial position of the nodebased on the velocity and timestep. The complexity ofthis function is O(n).

Calculate Kinetic Energy: The kinetic energy of thesystem is calculated by summing the square of thevelocities in the x and y directions. The complexity ofthis function is O(n).In our implementation, we do not strictly emulate aphysical system, but rather use a modified set of equationsformulated by Eades [2] that considers repulsive forcesbetween all nodes, but only attractive forces betweenconnected nodes.

4.2 CPU Implementation

The algorithm is implemented in a Python runtime environment. The SDK used is Python 2.7. The execution time is calculated for the CPU and shown on the Python shell.
The time is calculated in seconds. The data used in this paper was taken over from the internet.

4.3 GPU Implementation

To implement the algorithm on a parallel processing unit, the NVIDIA's CUDA GPU platform is chosen. The SDK is CUDA v4.2 with compute capability 2.1. The programming language chosen for this platform is PyCUDA. The execution time of the program on GPU is computed and is shown on the command prompt. The time calculated is in seconds. In comparison with the CPU, the time taken is reduced to 55-60% of the CPU time.

5. EVALUATION AND RESULTS

The output of the program is given below: from Figure 2 to Figure 7

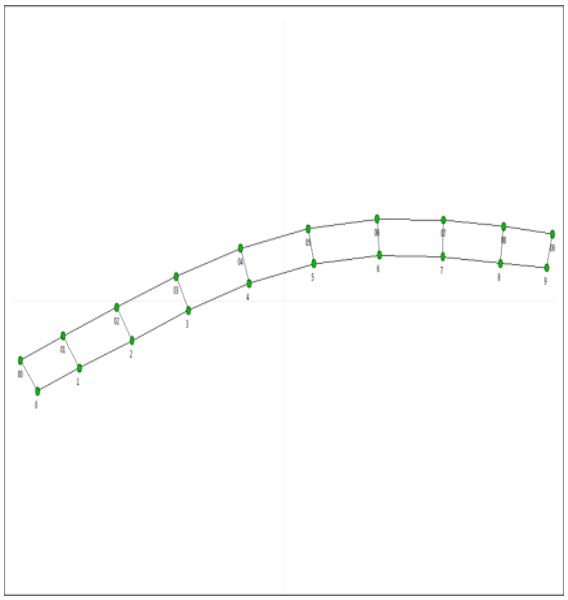


FIGURE 2: Output1

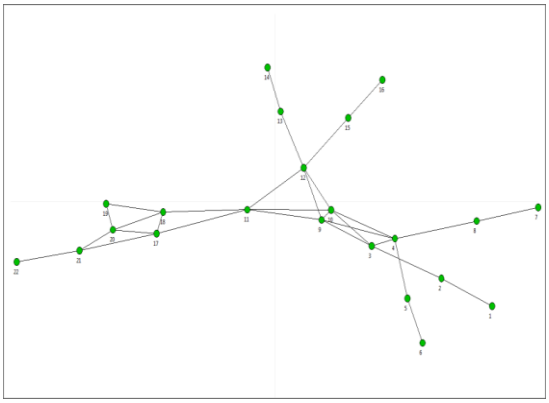
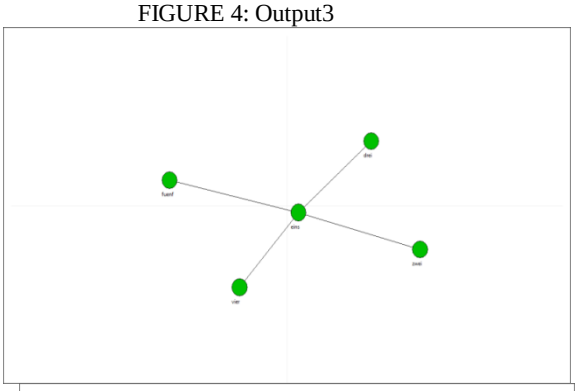


FIGURE 3: Output2

FIGURE 4: Output3



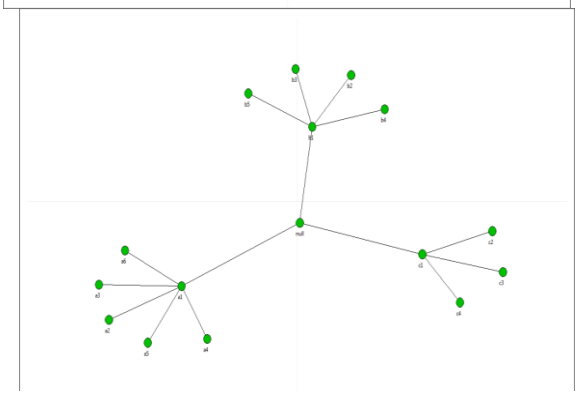


FIGURE 5: Output4

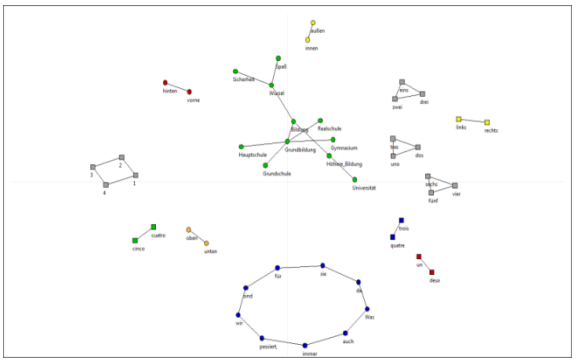


FIGURE 6: Output5

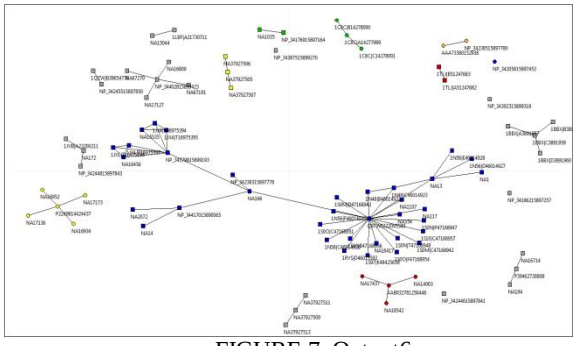


FIGURE 7: Output6

The final result for the comparison of GPU and CPU time is as shown in the table 1

Table 1: Comparison of GPU and CPU

File	CPU(in sec)	GPU(in sec)	Time Difference
Output1	0.187	0.078006488	58
Output2	0.1399	0.07999345	42.82%
Output3	0.125	0.07800646	37.56%
Output4	0.1559	0.07800689	49.93%
Output5	0.1559	0.07800365	49.96%
Output6	0.2179	0.08100045	62.28%

Graphical Analysis of the result:

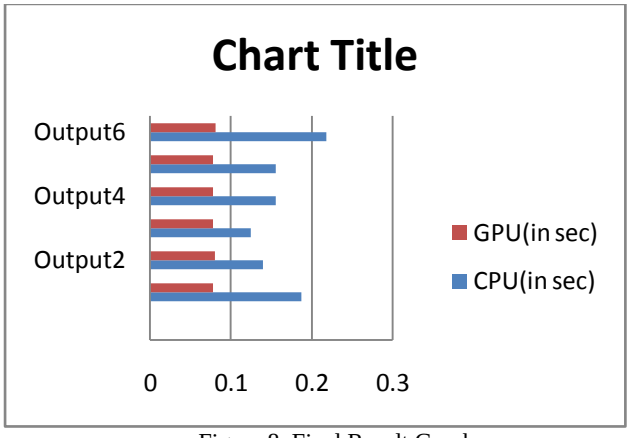


Figure 8: Final Result Graphs

6. ADVANTAGES AND DISADVANTAGES OF FORCE-DIRECTED PLACEMENT

The following are among the most important **advantages** of force-directed algorithms:

Good-quality results

At least for graphs of medium size (up to 50–100 vertices), the resul obtained have usually very good results based on the followin criteria: uniform edge length, uniform vertex distribution an showing symmetry. This last criterion is among the most important ones and is hard to achieve with any other type of algorithm.

Flexibility

Force-directed algorithms can be easily adapted and extended to fulfil additional aesthetic criteria. This makes them the most versatile class of graph drawing algorithms. Examples of existing extensions include the ones for directed graphs, 3D graph drawing,[5] cluster graph drawing, constrained graph drawing, and dynamic graph drawing.

Intuitive

Since they are based on physical analogies of common objects, like springs, the behaviour of the algorithms is relatively easy to predict and understand. This is not the case with other types of graph-drawing algorithms.

Simplicity

Typical force-directed algorithms are simple and can be implemented in a few lines of code. Other classes of graph-drawing algorithms, like the ones for orthogonal layouts, are usually much more involved.

Interactivity

Another advantage of this class of algorithm is the interactive aspect. By drawing the intermediate stages of the graph, the user can follow how the graph evolves, seeing it unfold from a tangled mess into a good-looking configuration. In some interactive graph drawing tools, the user can pull one or more nodes out of their equilibrium state and watch them migrate back into position. This makes them a preferred choice for dynamic and online graph-drawing systems.

The main **disadvantages** of force-directed algorithms include the following:

High running time

The typical force-directed algorithms are in general *considered* to have a running time equivalent to $O(n^3)$, where n is the number of nodes of the input graph. This is because the number of iterations is estimated to be O(n), and in every iteration, all pairs of nodes need to be visited and their mutual repulsive forces computed.

Poor local minima

It is easy to see that force-directed algorithms produce a graph with minimal energy, in particular one whose total energy is only a local minimum. The local minimum found can be, in many cases, considerably worse than a global minimum, which translates into a

low-quality drawing. For many algorithms, especially the ones that allow only *down-hill* moves of the vertices, the final result can be strongly influenced by the initial layout that in most cases is randomly generated. The problem of poor local minima becomes more important as the number of vertices of the graph increases.

7.REFERENCES

[1] "Graph drawing" [Online] Available: http://en.wikipedia.org/wiki/Graph_drawing

[2]http://en.wikipedia.org/wiki/Python_%programming_language%

[3]CUDA_C_Programming_Guide.pdf

[4] http://documen.tician.de/pycuda/

[5] https://developer.nvidia.com/pycuda

[6] P. Eades, 'A heuristic for graph drawing', Congressus Nutnerantiunt,42, 149–160 (1984).

[7] T. M. J. Fruchterman, E. M. Reingold: Graph drawing by forcedirected placement. Software.

[8] Y. Frishman, A. Tal 'Online Dynamic Graph Drawing' IEEE-VGTCSymposium on Visualization (2007)

[9] Y. Frishman, A. Tal 'Multi-Level Graph Layout on theGPU',Available:http://www.ee.technion.ac.il/~ayellet/Ps/FrishmanTalInfoVis07.pdf

[10]cs.brown.edu/~rt/gdhandbook/chapters/*force-directed.pdf*