Complex Systems Simulation with CUDA

Paul Richmond

Department of Computer Science
University of Sheffield
Sheffield, UK
p.richmond@sheffield.ac.uk

***Abstract*—Simulation of complex systems provides a computational challenge due to the amount of computational performance required to simulate all individuals within a large population. The Graphics Processing Unit (GPU) presents a potential solution. Using the CUDA programming language GPUs can be programmed for general purpose use. Unfortunately the translation of a complex systems model to CUDA code is a non-trivial task requiring specialist knowledge of the architecture to obtain good performance. This paper reviews FLAME GPU a framework which provides transparent mapping and simulation of complex systems models to a CUDA enabled GPU. The framework has considerable performance benefits over its CPU counterpart and provides real-time visualisation for interactive steering of simulations in domains as diverse as computational biology and pedestrian dynamics.**

I. INTRODUCTION *AND BACKGROUND*

A complex system is a system consisting of a large number of entities which interact to produce behaviour which demonstrates intelligence or self-organisation beyond that of the individuals within the system (emergent behaviour). Within biology complex systems are prevalent at many biological scales; from molecular systems to, cellular systems through to ecological systems of entire populations. Beyond the biological domain, complex systems represent phenomenon such as economic markets where the term 'complexity economics' has been derived to describe the application of complexity science to the field.

In order to understand a complex system, simulation can be applied to gain insight into the processes which result in system level behaviour. Simulation allows the behaviours of the entities comprising a complex system to be described as a set of rules, or *agents*, and their interaction to be simulated in order to observe and predict. This form of *agent based* simulation is advantageous as it can be used to explore the impact of changing an agent's behaviour or its environment. For example the impact of a drug on a particular chemical pathway or a building design with respect to evacuation safety.

The challenge of simulating large systems as interacting entities presents a computational challenge. For example, when compared to alternative methods of simulating population dynamics (such as top down equation based modelling), larger demands are placed both on computational performance and memory for storage of the systems state. Parallel computing presents a possible solution to providing increased levels of computing performance, in particular the Graphics Processing Unit (GPU) is commodity hardware boasting very high theoretical performance within a small power window. Applications which have been ported to the GPU often report speedups of orders of magnitude beyond their serial counterparts [1]. The transfer of a complex systems model to the GPU is however non-trivial. Whilst it can be observed that agents within a complex system 'may' be simulated independently, the simulation does not constitute an embarrassingly parallel problem (a problem where the task can be separated into tasks with no interdependencies). Within a complex systems simulation, communication (and hence synchronisation) between individuals must be carefully managed, agent creation and deletion (life and death) must be handled defiantly and code divergence must be minimised (see section III). In order to solve the challenge of mapping a complex systems model to the GPU therefore requires knowledge of the underlying architecture. This is particularly important on GPUs where much of the intelligence offered in traditional computing architectures (such as large data caches, out of order execution and long pipelines) is sacrificed for additional throughput.

II. FLAME GPU FRAMEWORK

The Flexible Large Scale Agent Modelling Environment for the GPU (FLAME GPU) is a framework which aims to vastly improve the performance of complex systems simulations whilst simultaneously providing a level of abstraction which hides the underlying complexities of the architecture. Ultimately the objective of the software is to provide a high level interface for describing the behaviour and interactions of agents (the model) and then automatically translating this to high performance parallel GPU code [2].

The basis of a FLAME GPU model is an XML description of the individual agents and the information (in the form of messages) shared between them. Agents are described using a high level representation based on a communicating stream X-Machine, a form of state machine. During the transition between two states a user defined *agent function* is defined to capture the translation of internal memory into a new state. During the agent function a message (the only form of communication between agents) may be input or output (but never both) via a global list. This indirect message communication allows functional dependencies between agents to be observed, allowing increased parallelism within the model and minimising synchronisation points between agent functions.

Generation of GPU code within FLAME GPU is handled through the use of templating in XSLT. An XLST template is a document which describes the translation of an XML document to another form (usually to another XML document but not exclusively). XSLT is a Turing complete functional programming languages, allowing complex queries of the model to be made in order to recognise aspects of the model where particular performance optimisations may be applied. The advantage of using code generation in this way is that very efficient GPU code can be generated without any overheads which may be introduced by producing code in more general form. As part of the code generation process (Figure 1) the user defined agent functions are linked with the dynamically generated simulation code to produce a simulation program. The program can be executed as either a console only simulation or via an interactive visualisation window. In each case the intermediate storage format for agent data (such as the initial and final states) are loaded from an XML file passed to the simulation executable as an argument.

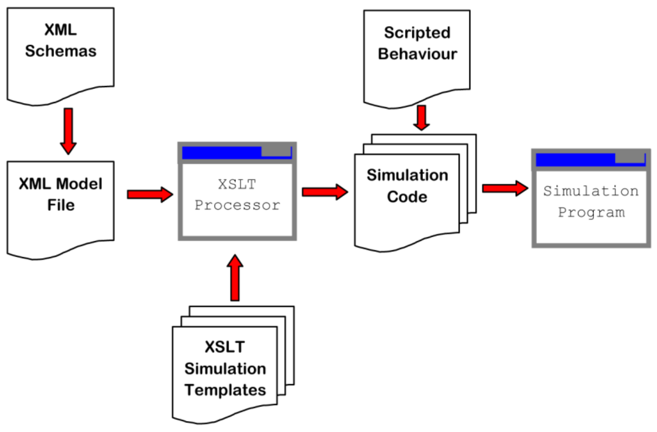

Figure 1- The FLAME GPU tool-chain showing how a model is translated to simulation code.

Visualisation is provided through instancing of a simple spherical geometry with colours used to represent agents in differing states. The visualisation code interfaces with the simulation in such a way as to allow the visualisation of agents to be provided as a user defined rendering module. In the FLAME GPU SDK a number of examples demonstrate this technique. For example rendering using levels of detail (using different geometric models depending on viewing distance) and animation of agents for visualising simulations of pedestrian dynamics (Figure 4).

III. GPU SIMULATION

The FLAME GPU code generation process outputs the NVIDIA Compute Unified Device Architecture (CUDA) language. CUDA is a language based on C which provides a number of extensions (with a dedicated compiler) allowing the specification of data parallel program *kernels*. Each *kernel* gives the high-level impression of the GPU device as a Single Program Multiple Data (SPMD) architecture; i.e. the same program is executed by a large number of simultaneous threads operating on different data. Translation of threads to the GPUs multiprocessors is handled by the runtime system, small (32 wide) thread groups (*warps*) are transparently scheduled and switched to hide memory bandwidth latencies. Within a warp the threads follow a Single Instruction Multiple Data (SIMD) model however different warps are free to follow different code paths depending on conditional statements within the program. Given the above execution model it is important that threads within a warp do not diverge (branch through different code paths). Where divergence is present within a warp, each code path must be serially evaluated requiring threads to remain idle causing considerable performance implications.

One of the major hindrances of mapping an agent based model to the GPU architecture is that as the complexity of an agent increases, so too does the likelihood of heterogeneity within the population. This heterogeneity introduces divergent behaviours across the population and ultimately divergent paths within the simulation code. FLAME GPU utilises a number of mechanisms to reduce divergent behaviours as well as improving performance through intelligent uses of device caches which are as follows;

State Based Representation: FLAME GPU utilises a one to one mapping of agents to CUDA threads. To ensure that code divergence is minimised agents are grouped with state lists (derived from the original state based representation). This ensures that agents performing similar behaviours (i.e. the same agent functions) will be within the same warp. The use of states can also be used to minimise divergent branches within agent functions as branches can instead be handled by separating a group of agents into different states (based on some condition). Recent advances in CUDA functionality allow simultaneous kernel launches which ensures that many small groups of state based agents can simultaneously execute agent functions (assuming there are no message communication dependencies between them) ensuring good hardware utilisation even for smaller state lists of agents.

Avoiding Sparse Data: Sparse data occurs when agent or message data is selectively removed or created producing non-contiguous data layouts within memory. For example, when agents change state, are born or die or when messages are selectively output. Sparse data creates a problem for GPU memory accesses as memory bandwidth is highest when adjacent threads access similarly aligned memory locations facilitating small numbers of wide memory requests to be issued. To ensure that sparse data is avoided in FLAME GPU parallel compaction via the Thrust library [3] is heavily used to ensure all data lists are compact.

Message Access Acceleration: A number of caches are available to CUDA programmers. Using these effectively to reduce the total number of slow global memory reads can increase performance considerably. FLAME GPU allows users to distinguish between specific message reading mechanisms including; brute force where each agent reads every available message, spatially distributed message space where agents within 2D or 3D continuous space read messages within a fixed radius and discrete message space where each message occupied a single space within a 2D discrete grid. Each message access mechanism has a differing implementation and cache usage which enables messages to be read efficiently for each case. For example, brute force messaging utilises the per-multiprocessor shared memory by batch loading a number of messages which are then sequentially accessed by agents in the thread group. Spatially partitioned messaging builds a spatial data structure through parallel (atomic or radix depending on the architecture) sorting and utilises the texture cache to vastly reduce the number of messages read by each agent. Discrete agent communication uses either shared memory or the texture cache to load a 2D grid of messages in discrete locations avoiding the necessity of sorting or building data structures during runtime.

IV. Benchmarking Results

To demonstrate the performance befits of FLAME GPU a benchmarking model is considered. The model called 'circles' is a force resolution model in which a *circle* agent iteratively attempts to move away from neighbours by considering their location and calculating an average repulsive force. The model consists of three agent functions and a single message type shown in Figure 2. During a single iteration of the simulation agents will move from the starting state to the end state completing each function in turn.

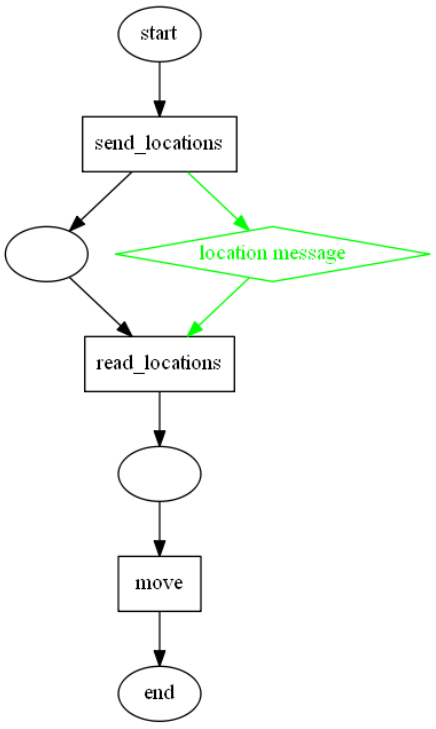

Figure 2 - The circles benchmark model for FLAME GPU showing the three distinct agent transition functions and the communication dependency based on location messages (green).

Computational performance of the circles model in FLAME GPU is considered by comparing simulation performance with FLAME for the CPU (www.flame.ac.uk) which runs on a traditional single threaded CPU architecture. The FLAME GPU model has been implemented with both brute force message (which tests location distances and disregards some messages) and with spatially partitioned messaging by pre-filter messages within a defined interaction radius. FLAME GPU version 1.4 was used for the performance benchmarking on an NVIDIA Tesla K40 GPU. All models are available for download within the FLAME GPU SDK (www.flamegpu.com). The benchmarking results are shown in Figure 3. The performance of the GPU brute force method is up to 250X that of the single threaded CPU equivalent for 131072 agents. Spatially distributed messaging performance improves performance considerably, however, direct comparisons cannot be made due to the use of spatial data structures in the GPU version (where as FLAME CPU uses brute force iteration on a single CPU node). A fairer comparison of the spatially partitioned technique can therefore be made by considering FLAME for the CPU on a distributed HPC system. In a HPC environment a FLAME CPU model can be spatially partitioned across multiple CPU cores communicating via MPI (albeit without dynamic load balancing). FLAME II [4] represents a highly optimised version of HPC FLAME which uses vector instructions and load balancing however the GPU is still roughly 700X faster than the performance reported.

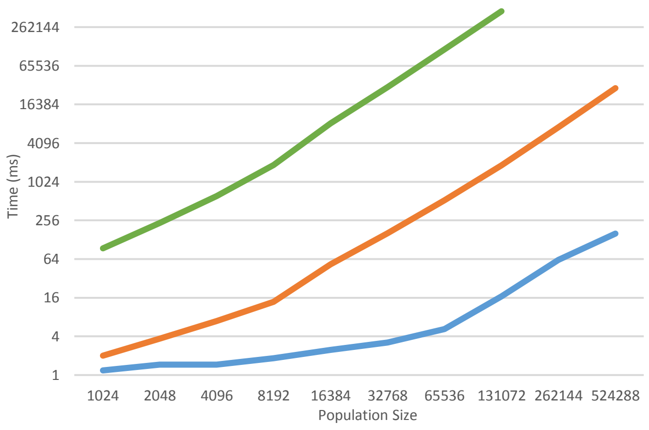

Figure 3- Performance scaling of the circles benchmark model with logarithmic y-axis. Showing FLAME CPU (green), FLAME GPU brute force messaging (orange), FLAME GPU (spatially partitioned messaging (blue).

V. Application Areas

Due to the granularity of parallelism (i.e. the mapping of agents the GPU threads), FLAME GPU is most suitable for simulating systems consisting of large number of relatively simple agents. For simulations with low number of highly complex agents (for example agents with very complex transition functions or requiring large amounts of agent storage) the GPU can often be underutilised as typically a large number of threads are required to provide context switching capable of hiding memory latencies. As such FLAME GPU is particularly well suited to application areas within computational biology either at the molecular or cellular level which require large number of agents. In the case of cellular systems FLAME GPU has been demonstrated to be highly effective at simulating epithelium tissue formation [5]. In this case agent behaviour divergence is addressed by modelling cells as differing cell types. A large amount of performance is also attributed to parallelisation of the resolution of intra-cellular forces to within an acceptable tolerance.

In addition to molecular and cellular level modelling, FLAME GPU is highly suitable to the simulation of large populations of individuals. Swarm based modelling and modelling of pedestrian dynamics are particularly well suited. In the case of pedestrian dynamics the model can incorporate the behaviour of individuals as well representing information about the environment in order to demonstrate goal driven navigation [6]. When combined with interactive visualisation pedestrian dynamics models can be used to provide real-time interactive simulations (Figure 4) which can be steered to observe the effects of external inputs or parameter changes (e.g. closing an exit route of a busy urban street).

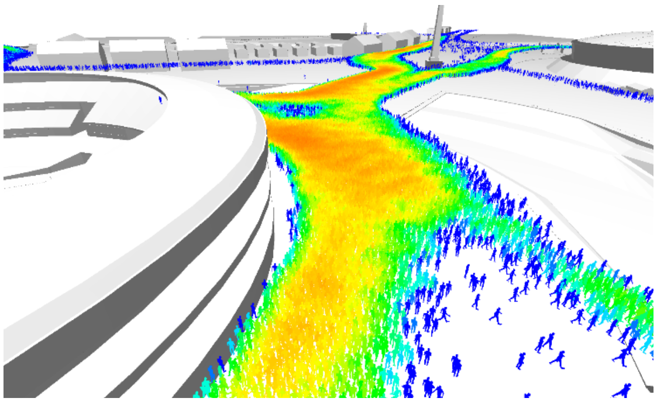

Figure 4 - A real-time interactive simulation of pedestrian behaviour within a large scale urban environment. The simulation runs at over 60fps with over 100k agents.

VI. DISCUSSION

The FLAME GPU framework has been reviewed giving details of how the software allows a high level representation of a complex systems model to be automatically translated to optimised GPU code. As part of this processes the challenge of minimising agent diversity is resolved through the use of a state based representation facilitating good utilisation of the GPU architecture.

Calculating the speedup of GPU implementations against CPU versions of the same model is challenging as each version must perform the same implementation and be optimised to ensure a fair comparison can be made. In the case of benchmarking for FLAME GPU, comparisons have been drawn with the CPU version of FLAME (and HPC versions of FLAME II). The CPU implementation is single threaded and by no means optimised, however, it does represent a fair indication of a CPU framework with the same modelling capabilities and flexibility, a key criteria for modellers not wanting to burden themselves with the details of the architecture on which their model is executed.

ACKNOWLEDGEMENT

The work in this paper has been supported by an EPSRC student development fellowship and the University of Sheffield Vice Chancellors Fellowship Scheme.

REFERENCES

[1] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips, "GPU computing," in *Proceedings of the IEEE*, 2008.

[2] P. Richmond, S. Coakley and D. Romano, "A high performance agent based modelling framework on graphics card hardware with CUDA," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, 2009.

[3] J. Hoberock and N. Bell, *Thrust: A Parallel Template Library*, http://thrust.github.io/, 2010.

[4] S. Coakley, M. Gheorge, M. Holcombe, S. Chin, D. Worth and C. Greenough, "Exploitation of high performance computing in the FLAME agent-based simulation framework," in *In High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS)*, 2012.

[5] P. Richmond, D. Walker, S. Coakley and D. Romano, "High performance cellular level agent-based simulation with FLAME for the GPU," *Briefings in bioinformatics*, vol. 11, no. 3, pp. 334-347, 2010.

[6] T. Karmakharm, P. Richmond and D. Romano, "Agent-based Large Scale Simulation of Pedestrians With Adaptive Realistic Navigation Vector Fields," in *TPCG 10*, 2010.