Parallel Agent systems on a GPU for use with Simulations and Games

Contribution

* Descriptions of parallel agent based computing system
* Agents are placed on GPU memory and executed in parallel on the GPU
* Difficulties in creating parallel agent based computing system and provide solutions
* Not define what agent is, but what agent does
* What is General Purpose GPU (GPGPU) and its memory structure
* Main advantage of performing agent calculations on the GPU is the minimise of data transfers, reducing the amount of processing time overall.

Framework

* Agents can perform of independent actions working in order to achieve a goal on someone’s (possibly their own) behalf.
* Agents can of interact with other agents or humans, cooperating and negotiating as and when required.
* Some agents can learn from experience
* Some agents influence other agents in the environment (system) which might change relationships
* Agents are not Artificial Intelligence (AI) (!), agents give us a way of accessing AI (!)
* Each agent in a multi-agent system is self-interested and will behave accordingly
* General Purpose GPU (GPGPU) – combination of pixel shaders and vertex processors
* Agents workings can be thought of as a continuous loop made up of the following steps:

1. Observation - Observe the world and update any beliefs the agent have about the world.
2. Deliberation - Decide what we need to do with this information, i.e. what should the agent goal be after absorbing this information ? This step is important and needs to be reviewed frequently by the agent as the environment might change. On the other hand, deliberation must not take place too often as otherwise no decisions will ever take place (agent can be considered reactive or proactive depending on how much time it spends deliberating about the changes in its environment).
3. Planning - With a set intention, how can I (the agent) accomplish this goal ? Generate a plan to achieve goal.
4. Execution - Execute the plan generated in the previous step.

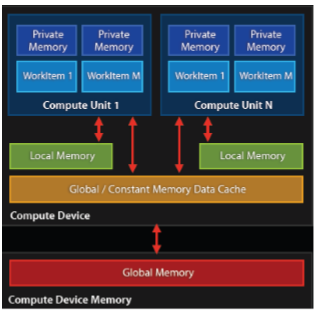
Use-Case

N/A

Tools / Techniques

* GPGPU is built up of lots of Multi Processors (MPs)
* Each MP has many Stream Processors (SPs)
* Each of the SPs executes in SIMD (single instruction, multiple data) mode.
* Multi-Processor executes the same instructions at the same time on different parts of the memory.
* GPU the memory consists of :-

1. Global memory - accessible by all processors, but due to high latency, making access to it slow
2. Shared memory – memory speciﬁc to a single MP and only accessible by the SPs on that MP. Shared memory has low latency, thus fast access to it.
3. Private memory - only accessible by the SP



* Use GPU Global memory similarly to a traditional multithreaded CPU based program that would make use of its shared memory.
* In GPU, we write messages to the GPU Global memory and then have the relevant agent read that message and act accordingly.
* In order to deal with synchronisation of reads and writes to avoid data corruption problems, GPU architectures are designed to use to 2 kernels :

1. Write data to memory
2. Contain all the actions that perform reads from memory.

* Code Level Optimisations - Threads (or in our case agents) that intend to follow the same path of execution are grouped up and executed together to increase performance.
* Branch Distribution – extracting similar code sections from different branches of code and merge this code to reduce the overall processing time.

Theories

* Agents work independently and cooperate with one another to achieve their goals.
* GPUs are excellent at processing large amounts of data in parallel.
* The combination of agents giving us a way of accessing AI along with GPU performance, games experience to the player improves. Also, online, pedestrian and trafﬁc dynamics simulation improves for the user.
* Agent data is stored in the memory in the form of arrays which track the different properties of each agent, including details such as the position of the agent in a three dimensional world.
* Agent communication done using global memory – check if target already busy and if so, wait for the next iteration. Otherwise, the communicating agent will write data into the target’s agent’s memory. The target agent will identify the communicator and write the data into the communicating agent memory.
* Data transfer between CPU and GPU especially in Games as keyboard user input is required.
* Multiple runs of data processing on the GPU before moving it back to the CPU not valid if wishing to present real-time output to the user (and in Games or simulations, we do) .
* Option is to leave results on the GPU the whole time and draw the data directly from the GPU memory.

Critic

* Cannot expect GPU to be answer to all problems. There are difficulties with GPU.
* How do we make sure agents communicate with one another on the GPU ?
* How can a GPU efﬁciently handle agents making decisions ?
* How can we minimise the overhead of moving data to and from the CPU ?
* How do we keep our agents synchronised ?
* How do we deal with slow access to Global memory on the GPU ?
* Message passing architectures designed for distributed systems (and our system is not)
* GPU not always faster than CPU
* If Code Level Optimisations and Branch Distribution unused, CPU showed on occasions to be faster

New Terminology

Spheres of inﬂuence

Multi Processors (MPs)

Stream Processors (SPs)

SIMD (single instruction, multiple data)

Citations

This paper had only 3 citations. However, I found it very beneficial for my understanding of the GPU. There could be a number of other papers which may be relevant :-

[3] J. Hagelb¨ack and S. J. Johansson. Using multi-agent potential ﬁelds in real-time strategy games. In Proceedings of the 7th international joint conference on Autonomous agents and multi agent systems - Volume 2, AAMAS ’08, pages 631–638, Richland, SC, 2008. International Foundation for Autonomous Agents and Multi agent Systems. **67 citations**

[11] G. Yamamoto and H. Tai. Performance evaluation of an agent server capable of hosting large numbers of agents. In Proceedings of the ﬁfth international conference on Autonomous agents, AGENTS ’01, pages 363–369, New York, NY, USA, 2001. ACM. **27** **citations**

[4] T. D. Han and T. S. Abdelrahman. Reducing branch divergence in gpu programs. In Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, GPGPU-4, pages 3:1–3:8, New York, NY, USA, 2011. ACM. **88** **citations**

[5] O. Maitre, L. A. Baumes, N. Lachiche, A. Corma, and P. Collet. Coarse grain parallelization of evolutionary algorithms on gpgpu cards with easea. In Proceedings of the 11th Annual conference on Genetic and evolutionary computation, GECCO ’09, pages 1403–1410, New York, NY, USA, 2009. ACM. **80** **citations**

[6] E. Manisterski, R. Lin, and S. Kraus. Understanding how people design trading agents over time. In Proceedings of the 7th international joint conference on Autonomous agents and multi agent systems - Volume 3, AAMAS ’08, pages 1593–1596, Richland, SC, 2008. International Foundation for Autonomous Agents and Multi agent Systems. **21** **citations**

[8] D. W. Roeh, V. V. Kindratenko, and R. J. Brunner. Accelerating cosmological data analysis with graphics processors. In Proceedings of 2nd Workshop on General Purpose Processing on GraphicsProcessingUnits,GPGPU-2,pages1– 8, New York, NY, USA, 2009. ACM. **20** **citations**