A Standardised Benchmark for Assessing the Performance of Fixed Radius Near Neighbours

Robert Chisholm, Paul Richmond, and Steve Maddock

Department of Computer Science, The University of Sheffield, UK
{r.chisholm,p.richmond,s.maddock}@sheffield.ac.uk

Abstract Fixed radius near neighbours (FRNNs) lies at the centre of many agent based models, whereby agents require awareness of their local peers. Due to its central role, handling of FRNNs is often a limiting factor of performance. However without a standardised metric to assess the handling of FRNNs, contributions to the field lack the rigorous appraisal necessary to expose their relative benefits.
This paper presents a standardised specification of a multi agent based benchmark model. The benchmark model provides a means for the objective assessment of FRNNs performance, through the comparison of implementations. Results collected from implementations of the benchmark model under three agent based modelling frameworks show the CPU bound performance to scale linearly with agent population, in contrast the GPU accelerated framework only became linear after maximal device utilisation around 150,000 agents. The performance of each of the assessed frameworks was also found unaffected by changes to the rate of agent movement.

Keywords: Parallel Agent Based Simulation, OpenAB, Benchmarking, Fixed Radius Near Neighbours, FLAMEGPU, MASON, REPAST

1 Introduction

Many complex systems have mobile entities located within a continuous space such as: particles, people or vehicles. Typically these represented via Agent Based Simulation (ABS) where entities are agents. In order for these mobile agents to decide actions, they must be aware of their neighbouring agents. This awareness is typically provided by fixed radius near neighbours (FRNNs) search, whereby each agent considers the properties of every other agent located within a spatial radial area about their simulated position. This searched area can be considered the agent's neighbourhood and must be searched every timestep of a simulations, ensuring the agent has access to the most recent information about their neighbourhood. In many cases such as flocking, pedestrian interaction and cellular systems, the majority of time is spent performing this neighbourhood search, as opposed to agent logic. It is hence often the primary performance limitation.

The most common technique utilised for accelerating FRNNs is one of uniform spatial partitioning. Within uniform spatial partitioning, the environment is decomposed into a regular grid, partitioned according to the interaction radius. Agents are then stored or sorted according to the grid cell they are located

within. Agents consider their neighbourhood by performing a distance test on all agents within their own grid partition and any directly adjacent neighbouring grid cells. This has caused researchers to seek to improve the efficiency of FRNNs handling, primarily by approaching more efficient memory access patterns [2,4,8]. However without a rigorous standard to compare implementations, exposing their relative benefits is greatly complicated.

With ABSs reliance on FRNNs, there are many capable available frameworks, providing initial FRNNs implementations for assessment. The Open Agent Benchmark Project (OpenAB)[1] exists for the wider assessment of ABSs and to pool the research community's ABS knowledge and resources. This paper uses the OpenAB's process of publishing a simulator independent benchmark model in a format which allows the performance of implementations across multiple ABS frameworks to be compared. By unifying the process of benchmarking ABSs it is hoped that the OpenAB project will foster the necessary transparency and standards among the ABS community, ensuring that rigorous benchmarking standards are adhered to.

This paper formalises and standardises a benchmark model named circles, previously implemented by frameworks such as FLAMEGPU [7]. The model is specifically standardised and designed to assess the performance of FRNNs implementations. A formal specification of the benchmark and it's applications is provided alongside a preliminary comparison of results obtained from FLAMEGPU, MASON and REPAST. This work has been published to the OpenAB website[2] and provides a foundation for the future assessment of ABSs frameworks, providing a motivation for the consideration of ABS framework performance.

The results within this paper assess each frameworks FRNNs implementation against the metrics of problem size, neighbourhood size, and entropy, which can be measured using the circles benchmark. Most apparent from these results is how the runtime scales linearly with problem size after maximal hardware utilisation. However, a much larger problem size is required to fully utilise Graphics Processing Unit (GPU) hardware.

The remainder of this paper is organised as follows: Section 2 provides an overview of related research; Section 5 lays out a clear specification of the circles benchmark model and how it can be utilised effectively; Section 4 details the frameworks which have been assessed using the benchmark; Section 5 discusses the results obtained from the application of the circles benchmark to each framework; Finally Section 6 presents the concluding remarks and directions for further research.

2 Related Research

FRNNs searches are most often found within agent-based models. They have also been used alongside similar algorithms within the fields of Smoothed-Particle Hydrodynamics (SPH) and collision detection. FRNNs is the process whereby each agent considers the properties of every other agent located within a radial

[1] http://www.openab.org
[2] http://www.openab.org/benchmarks/models/submit/circles/

area about their location. This searched area can be considered the agents neighbourhood and must be searched every timestep of a simulation to ensure agents have live information. Whilst various spatial data-structures such as kd-trees and R-trees are capable of providing efficient access to spatial neighbourhoods, their expensive constructions make them unsuitable for the large dynamic agent populations found within agent-based models.

The naive technique for carrying out a neighbourhood search is via a brute-force technique, individually considering whether each agent is located within the target neighbourhood. This technique may be suitable for small agent populations, however the overhead quickly becomes significant as agent populations increase, reducing the proportional volume of the neighbourhoods with respect to the volume of the environment.

The most common technique that is used to reduce the overhead of FRNNs handling is that of uniform spatial partitioning (Figure 1), whereby the environment is partitioned into a uniform grid, whereby grid cells have dimensions equal to the interaction radius. Agents are then (sorted and) stored according to the ID of their containing cell within the grid. Serial implementations are likely to utilise linked list's to store the agents within each bin. Parallel implementations in contrast are likely to store agents within a single compact array which is sorted in a distinct step after agent locations have been updated, following which an index to provide direct access to the storage of each cells agents is produced. This allows the Moore neighbourhood of an agents cell to be accessed, ignoring agents within cells outside of the desired neighbourhood. This method is particularly suitable for parallel implementations [3] and several advances have been suggested to further improve their performance: Goswami et al proposed the use of Z-order curves to improve memory locality [2]; Hoetzlein considered the effect of changing the partition cell dimensions [4]; and Sun et al proposed the use of a parallel ordered sort to improve sorting efficiency [8].

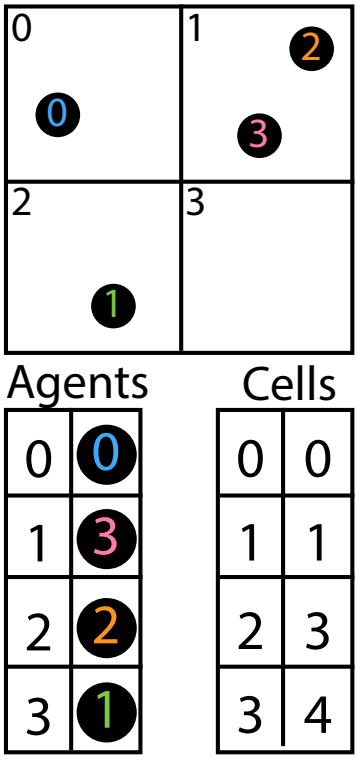

Figure 1. A representation of a data structure that can be used for uniform spatial partitioning.

Recent FRNNs publications have either provided no comparative performance results, or simply compared with their prior implementation lacking the published innovation [2,4,8]. With numerous potential innovations which may interact and overlap it becomes necessary to

standardise the methodology by which these advances can be compared both independently and in combination. When assessing the performance of High Performance Computation (HPC) algorithms there are various approaches which must be taken and considered to ensure fair results.

When comparing the performance of algorithms there are a plethora of recommendations to be followed to ensure that results are not misleading[1]. The general trend among these guidelines is the requirement of explicit detailing of experimental conditions and ensuring uniformity between test cases such that results can be reproduced. Furthermore, if comparing algorithm performance across different architectures it is important to ensure that appropriate optimisations for each architecture have been implemented when discussing results. Historically there have been numerous cases whereby comparisons between CPU and GPU have shown speedups as high as 100x which have later been debunked due to flawed methodology [5].

3 Benchmark Model

The circles benchmark model is designed to utilise neighbourhood search in a manner analogous to a simplified particle simulation in two or three dimensions (although it could easily be extended to higher levels of dimensionality if required). Within the model each agent represents a particle whose location is clamped within the environment bounds. Each particle's motion is driven by forces applied from other particles within their local neighbourhood.

The parameters (explained below) of the circles benchmark allow it to be used to assess how the performance of FRNNs search implementations are affected by changes to factors such as problem size and neighbourhood size. This assessment can then be utilised in the research of FRNNs ensuring comparisons against existing work and to advise design decisions when requiring FRNNs during the implementation of ABS.

3.1 Model Specification

The benchmark model is configured using the parameters in Table 1. In addition to these parameters the dimensionality of the environment (E_{dim}) must be decided, which in most cases will be 2 or 3. The value of E_{dim} is not considered a model parameter as changes to this value are likely to require implementation changes. The results presented later in this paper are all from 3D implementations of the benchmark model.

Initialisation Each agent is solely represented by their location. The total number of agents A_{pop} is calculated using Equation 1.[3] Initially the particle agents are randomly positioned within the environment of diameter W and E_{dim} dimensions.

$$A_{pop} = \lfloor W^{E_{dim}} \rho \rfloor \tag{1}$$

[3] $\lfloor \rfloor$ represents the mathematical operation floor.

Parameter	Description	Fig. 2	Fig. 3
k_{rep}	The repulsion dampening argument. Increasing this value encourages agents to repel.	1×10^{-5}	1×10^{-5}
k_{att}	The attraction dampening argument. Increasing this value encourages agents to attract.	1×10^{-5}	1×10^{-5}
r	The interaction radius. Increasing this value increases the radius of the neighbourhoods searched, thus increasing agent communication.	5	1-15
ρ	The density of agents within the environment.	0.01	0.01
W	The diameter of the environment. This value is shared by each dimension therefore in a two dimensional environment it represents the width and height. Increasing this value is equivalent to increasing the scale of the problem (e.g. the number of agents) assuming ρ remains unchanged.	50-300	100

Table 1. The parameters for configuring the circles benchmark model.

Single Iteration For each timestep of the benchmark model, every agent's location must be updated. The position x of an agent i at the discrete timestep $t+1$ is given by Equation 2, whereby F_i denotes the force exerted on the agent i as calculated by Equation 3.[4] Within Equation 3 F_{ij}^{rep} and F_{ij}^{att} represent the respective attraction and repulsion forces between agents i and j and d_{ij} represents the scalar distance between locations x_i & x_j. The values of F_{ij}^{att} and F_{ij}^{rep} are calculated using Equations 4 and 5, respectively.

$$x_{i(t+1)} = x_{i(t)} + F_i \tag{2}$$

$$F_i = \sum_{i \neq j} F_{ij}^{rep}[d_{ij} < r] + F_{ij}^{att}[r <= d_{ij} < 2r] \tag{3}$$

$$F_{ij}^{att} = \frac{k_{att}(d_{ij} - r)(x_i - x_j)}{d_{ij}} \tag{4}$$

$$F_{ij}^{rep} = \frac{k_{rep} d_{ij}(x_i - x_j)}{d_{ij}} \tag{5}$$

Algorithm 1 provides a pseudo-code implementation of the calculation of a single particles new location, whereby each agent only iterates their agent neighbours rather than the global agent population.

[4] The square Iversion bracket notation [ ] denotes a conditional statement; when the statement evaluates to true a value of 1 is returned otherwise 0

Algorithm 1 Pseudo-code for the calculation of a single particle's new location.

```
vec myOldLoc;
vec myNewLoc = myOldLoc;
foreach neighbourLoc
{
  vec locDiff = myOldLoc-neighbourLoc;
  float locDist = length(locDiff);
  float separation = locDist - INTERACTION_RAD;
  if(separation < INTERACTION_RAD)
  {
    float k;
    if(separation > 0)
      k = ATTRACTION_FORCE;
    else
      k = -REPULSION_FORCE;
    myNewLoc += k * separation * locDiff / INTERACTION_RAD;
  }
}
myNewLoc = clamp(myNewLoc, envMin, envMax);
```

Validation There are several checks that can be carried out to ensure that the benchmark has been implemented correctly. Most significantly it should be confirmed that agent locations remain within the environmental bounds. The absence of this clamping under certain model parameters allows agent density to decrease, which artificially benefits performance. It is intended that the benchmark model is able to reach a steady state over a number of iterations, however in some cases high-magnitude forces F_{att}F & F_{rep} may instead cause the agents to vibrate. When the cumulative force applied to each agent is within a reasonable range it is possible to predict the eventual steady state. *TODO: clarify param combinations:final state.*

3.2 Effective Usage

The metrics which may affect the performance of neighbourhood search implementations are agent quantity, neighbourhood size, agent speed and location uniformity. Whilst it is not possible to directly parametrise all of these metrics within the circles benchmark, a significant number can be controlled to provide understanding of how the performance of different implementations is affected.

To modify the scale of the problem, the environment width W can be changed. This directly adjusts the agent population size, according to the formula in Equation 1, whilst leaving the density unaffected. Modulating the scale of the population is used to benchmark how well implementations scale with increased problem sizes. In multi-core and GPU implementations this may also allow the point of maximal hardware utilisation to be identified, whereby lesser population sizes do not fully utilise the available hardware.

Modifying either the density ρ or the interaction radius r can be used to affect the number of agents found within each neighbourhood. The number of agents within a neighbourhood of radius r can be estimated using Equation 6, this value assumes that agents are uniformly distributed and will vary slightly between agents.

$$N_{size} = \rho\pi(2r)^{E_{dim}} \tag{6}$$

Modifying the speed of the agent's motion affects the rate at which the data structure holding the neighbourhood data must change (referred to as changing the entropy, the energy within the system). Most implementations are unaffected by changes to this value. However optimisations such as those by Sun et al [8] should see performance improvements at lower speeds, due to a reduced number of agents transitioning between cells within the environment per timestep. The speed of an agent within the circles model is calculated using Equation 3. There are many variables which impact this speed, but the most significant modifiers are those of attractive force (k_{att}) and repulsive force (k_{rep}). The closer these two forces are to 0 the slower that the agents move. If these forces are particularly high, agents are likely to oscillate rather than reaching a steady state as described in the previous subsection.

The final metric location uniformity, refers to how uniformly distributed the agents are within the environment. When agents are distributed non-uniformly, as may be found within many natural scenarios, the size of agent neighbourhoods are likely to vary more significantly. This can be detrimental to the performance of implementations which parallelise the neighbourhood search such that each agents search is carried out in a separate thread via single instruction multiple data (SIMD) execution. This is caused by sparse neighbourhoods spending large amounts of time idling whilst larger neighbourhoods executed simultaneously are searched. It is not currently possible to suitably affect the location uniformity within the circles model.

Independent of model parameters, the circles benchmark is also capable of assessing the performance of FRNNs when scaled across distributed systems, however that is outside the scope of the results presented within this paper.

4 Assessed Frameworks

The benchmark implementations assessed within this paper all target execution on a single machine. Care has been taken to follow best practices as expressed in the relevant documentation and examples provided with each framework to ensure that the optimisation of model implementations is appropriate. The associated model implementations are publicly available on this projects repository[5] and further details regarding the frameworks can be found on the OpenAB website[6]. The frameworks targeted within this research are:

[5] https://github.com/Robadob/circles-benchmark
[6] http://www.openab.org/benchmarks/simulators/

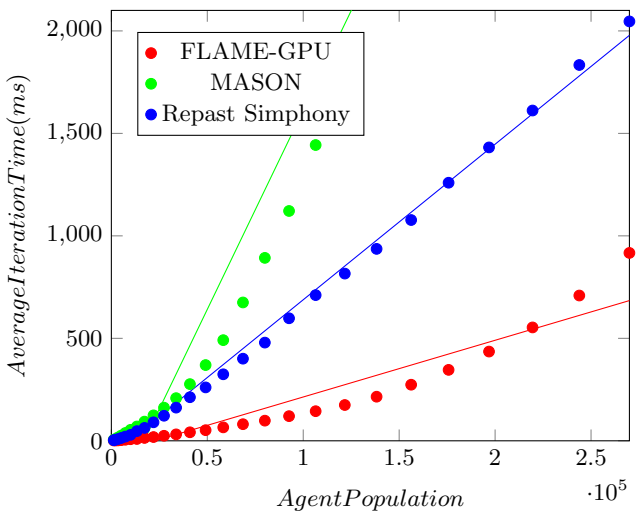

Figure 2. The average iteration time of each framework against the agent population.

- Inspired by the FLAME agent-based modelling framework, FLAMEGPU was developed to utilise GPU computation via a combination of XML and CUDA [7].
- MASON is a Java multiagent simulation toolkit capable of executing models with a large numbers of agents on a single machine, providing an additional suite of visualisation tools [6].
- The Repast collective of modelling tools has now been under development for over 15 years. Repast symphony targets computation on individual computers and small clusters, facilitating the development of agent-based models using Java and Relogo [**?**].

5 Results

Results presented within this section were collected on a single machine running Windows 7 x64 with a Quad core Intel Xeon E3-1230 v3 running at 3.3GHz[7]. Additionally the FLAME-GPU framework utilised an Nvidia GeForce GTX 750 Ti GPU which has 640 CUDA cores running at 1GHz.

Each of the parameter sets utilised targeted a different performance metric identified in Section . Results were collected by monitoring the total runtime of 1000 iterations of 3D implementations of the benchmark (executed without visualisation) and are presented as the per iteration mean. Initialisation timings are excluded as the benchmarks focal point is the performance of the nearest neighbours search carried out within each iteration. The results in Figure 2 present the variation in performance as the scale of the problem increases. This is achieved by increasing the parameter W, which increases the volume of the environment and hence the agent population. Most apparent from these results

[7] The processor supports hyper-threading, enabling 4 additional concurrent logical threads.

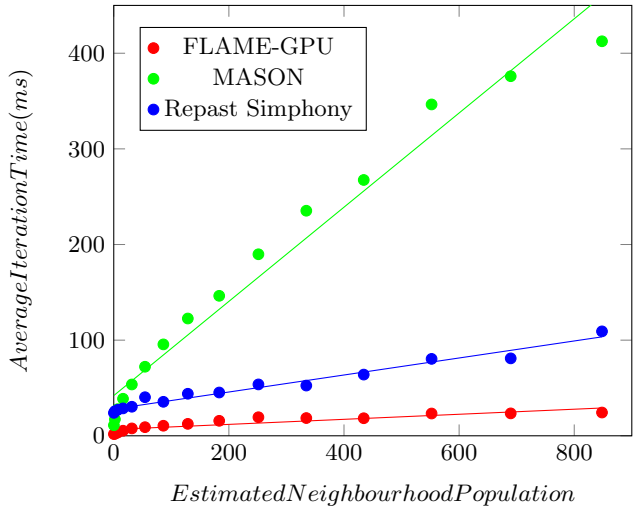

Figure 3. The average iteration time of each framework against the estimated neighbourhood population.

is that FLAMEGPU, which utilises GPU computation, as opposed to the other frameworks, which utilise a multi-threaded CPU approach, consistently outperforms the best multi-core framework by a margin of 2-5 times. This is in line with the expectations of GPU accelerated computation[5]. Although MASON and Repast Simphony are both Java based frameworks, MASON's performance trailed that of Repast by around 2-3x. Observation during runtime showed MASON with much lower multi-core utilisation than Repast, which likely explains this disparity.

The next parameter set, shown in Figure 3, assessed the performance of each framework in response to increases in the agent populations within each neighbourhood. The purpose of this benchmark set was to assess how each framework performed when agents were presented with a greater number of neighbours to survey. This was achieved by increasing the parameter r, hence increasing the volume of each agent's radial neighbourhood. The results show a fairly linear relationship for each framework. MASON initially outperformed Repast Simphony, however the effect of increased neighbourhood sizes saw Repast Simphony perform quicker when neighbourhoods were greater than 6 agents. The final parameter set assessed variation in performance in response to increased entropy. This is achieved by adjusting the parameters k_{att} and k_{rep}, causing the force exerted on the agents to increase, subsequently causing them to move faster. The purpose of this benchmark was to assess whether any of the frameworks benefited from reduced numbers of agents transitioning between spatial partitions. The results however showed that performance for each framework remained stable throughout all parameters as expected (See Section 3.2).

6 Conclusion

The work within this paper has provided a formal and standardised specification for the circles benchmark. This benchmark is beneficial for assessing the performance of FRNNs search implementations in response to changes to problem size, neighbourhood size and entropy. The results within this paper have shown the linear performance relationships of the tested ABS frameworks in response to changing agent populations and neighbourhoods. This provides a guide for those looking to implement ABS reliant on FRNNs and a metric to improve FRNNs search implementations. The next stage is further evaluation of standalone FRNNs implementations utilising the most recent research advances.

References

1. Bailey, D.H.: Misleading performance in the supercomputing field. In: Proceedings of the 1992 ACM/IEEE conference on Supercomputing. pp. 155–158. IEEE Computer Society Press (1992)
2. Goswami, P., Schlegel, P., Solenthaler, B., Pajarola, R.: Interactive sph simulation and rendering on the gpu. In: Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 55–64. Eurographics Association (2010)
3. Green, S.: Nvidia whitepaper: Particle simulation using cuda. `http://docs.nvidia.com/cuda/samples/5_Simulations/particles/doc/particles.pdf` (2010)
4. Hoetzlein, R.: Fast fixed-radius nearest neighbors: Interactive million–particle fluids. In: GPU Technology Conference (2014), `http://on-demand.gputechconf.com/gtc/2014/presentations/S4117-fast-fixed-radius-nearest-neighbor-gpu.pdf`
5. Lee, V.W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A.D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., et al.: Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. In: ACM SIGARCH Computer Architecture News. vol. 38, pp. 451–460. ACM (2010)
6. Liu, J., Chandrasekaran, B., Yu, W., Wu, J., Buntinas, D., Kini, S., Panda, D.K., Wyckoff, P.: Microbenchmark performance comparison of high-speed cluster interconnects. Micro, IEEE 24(1), 42–51 (2004)
7. Richmond, P., Romano, D.: A high performance framework for agent based pedestrian dynamics on gpu hardware. European Simulation and Modelling 3 (2008)
8. Sun, H., Tian, Y., Zhang, Y., Wu, J., Wang, S., Yang, Q., Zhou, Q.: A special sorting method for neighbor search procedure in smoothed particle hydrodynamics on gpus. In: Parallel Processing Workshops (ICPPW), 44th International Conference on. pp. 81–85. IEEE (2015)