

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 Agent based models are concerned with the interaction of independent agents. Central to this interaction is the concept of fixed radius near neighbours (FRNNs), providing agents an 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 the circles benchmark, providing a means for the objective assessment of FRNNs performance, to enable the contrasting of implementations. Supplementary results collected from implementations of 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 Nearest Neighbours, FLAMEGPU, MASON, REPAST

1 Introduction

Many Agent Based Simulations (ABSs) contain mobile agents located within a continuous space. These agents may represent entities such as particles, people and vehicles. 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 radial area about their location. This searched area can be considered the agent's neighbourhood and must be searched every timestep of a simulation, ensuring the agent has access to the most recent information about their neighbourhood. Consequently this causes the neighbourhood search to often occupy the majority of execution time within affected models.

The most common technique utilised for FRNNs is one of uniform spatial partitioning. The environment is decomposed into a regular grid, agents are then stored according to the grid cell they are located in a compact array. An

index can be used to provide fast access to the agents contained within each cell, however the nature of the problem leads to many scattered memory accesses harming performance. This has caused researchers to seek to improve the efficiency of FRNNs handling[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 an initial slate for assessment. The Open Agent Benchmark Project (OpenAB) exists for the wider assessment of ABSs and to pool the research community’s ABS knowledge and resources¹. *Something, something. This platform provides a public repository for the presentation of agent benchmark models and results collected across a wide range of simulation frameworks and hardware architectures* 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 standardises a benchmark model named circles, previously used by frameworks such as FLAMEGPU[7], which has been 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, REPAST. This work provides a foundation for the future assessment of ABS frameworks and a *something about choice of framework/motivating framework maintainers to improve perf?*.

The results within this paper assess each framework’s FRNNs implementation against the metrics of: problem size; neighbourhood size; and entropy, which can be levied from 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 area about their location. This searched area can be considered the agent’s neighbourhood and nyst be searched every timestep of a simulation to ensure agents

¹ <http://www.openab.org>

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 however, 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 size of the neighbourhoods. 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. Agents are then (sorted and) stored according to the ID of their containing cell within the grid. A separate index is then produced, providing fast access the storage of each cell’s agents. This allows the Moore neighbourhood of an agent’s 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 the performance[2,4,8].

Recent FRNNs publications have either provided no comparative performance results, or simply compared with an iteration 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, 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

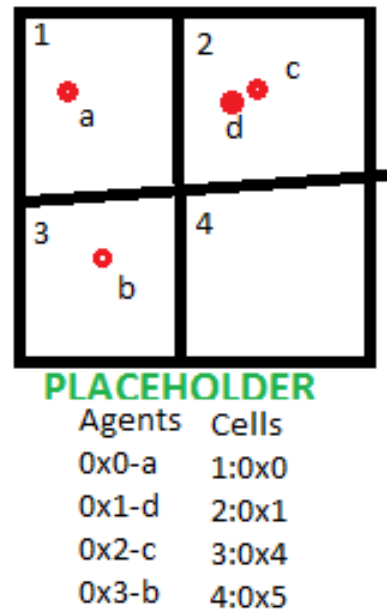


Figure 1. A representation of the data-structure used by uniform spatial partitioning.

each architecture have been carried out and that the cost of relevant hardware is considered 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 particles 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 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, in most cases this 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.

Initialisation Each agent is solely represented by their location. The total number of agents A_{pop} is calculated using Equation 1.² Initially the particle agents are randomly positioned within the environment of diameter W and E_{dim} dimensions.

$$A_{pop} = \lfloor W^{E_{dim}} \rho \rfloor \quad (1)$$

Single Iteration Each timestep of the benchmark model, every particle agent's location must be updated. The position x of a particle 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 3.³ Within Equation 3 F_{ij}^{rep} and F_{ij}^{att} represent the respective repulsive and attractive forces between agents i and j . The value of F_{ij}^{att} and F_{ij}^{rep} are calculated using Equations 4 and 5 respectively.

² $\lfloor \cdot \rfloor$ represents the mathematical operation floor.

³ The square Iversion bracket notation $\lfloor \cdot \rfloor$ denotes a conditional statement, when the statement evaluates to true a value of 1 is returned otherwise 0

Parameter	Description	Fig. 2	Fig. 3
k_{rep}	The repulsive dampening argument. Increasing this value encourages agents to repel.	0.00001	0.00001
k_{att}	The attractive dampening argument. Increasing this value encourages agents to attract.	0.00001	0.00001
r	The interaction radius. Increasing this value increases the radius of the neighbourhoods searched, subsequently 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.

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

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

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

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

Algorithm 1 provides a pseudo-code implementation of the calculation of a single agent’s new location, whereby each agent only iterates their agent neighbours rather than the global agent population. The benchmark is to be executed for the stated number of model iterations. This may result in multiple iterations in a steady state, however this does not diminish the validity of the measured metrics.

3.2 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 has a significant effect on the agent distributions over time, which artificially benefit performance. It is intended that the benchmark model is able to reach a

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);
```

steady state over a number of iterations, however in some cases high magnitudes forces F_{att} & F_{rep} may instead cause the agents to vibrate. When the cumulative force applied to each agent is however within a reasonable range it is possible to predict the eventual steady state. *TODO: clarify param combinations: final state.*

3.3 Effective Usage

There are various metrics which may affect the performance of neighbourhood search implementations: agent quantity, neighbourhood size, agent speed, 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 then 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 the density ρ and interaction radius r can both 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 calculation assumes that agents are uniformly distributed and in practise is likely to vary slightly between agents.

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

Modifying the speed of the agents motion, affects the rate at which the data-structure holding the neighbourhood data must change (for which reason it is 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] whereby sorting of agents is optimised for pre-sorted data 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, however the most significant modifiers are those of attractive force (k_{att}) and repulsive force (k_{rep}). 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. *TODO: why do we care?*

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.

Paul had comment about mentioning OpenAB here, not sure if meant end of this section or start of next

4 Assessed Frameworks

The benchmark implementations assessed within this paper all target execution on a single machine. Care has been taken to follow best-practises 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 repository⁴. The below frameworks have been targeted within this research.

Inspired by the FLAME agent-based modelling framework, FLAME-GPU was developed to utilise GPU computation via a combination of XML and CUDA. FLAME-GPU was first published by Dr Paul Richmond et al at the University of Sheffield[7]. MASON is a Java multiagent simulation toolkit capable of executing models with a large numbers of agents on a single machine across multiple machines[?]. The Repast collective of modelling tools has now

⁴ <https://github.com/Robadob/circles-benchmark>

been under development for over 15 years. The symphony product targets computation on individual computers and small clusters, facilitating the development of agent-based models using Java and Relogo[6].

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 3.3GHz⁵. 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 sans 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 is that FLAME-GPU 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 inline with the expectations of GPU accelerated computation[5], whereby the additional hardware cost must be factored in. Although MASON and Repast Symphony 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 agents radial neighbourhood. The results show a fairly linear relationship for each framework. MASON initially outperformed Repast Symphony, however the effect of increased neighbourhood sizes saw Repast Symphony 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 particles to increase, subsequently causing them to move faster. The purpose of this benchmark was to assess whether any of the frameworks benefited from particles reduced particle relocations. The results

⁵ The processor supports hyper-threading, enabling 4 additional concurrent logical threads.

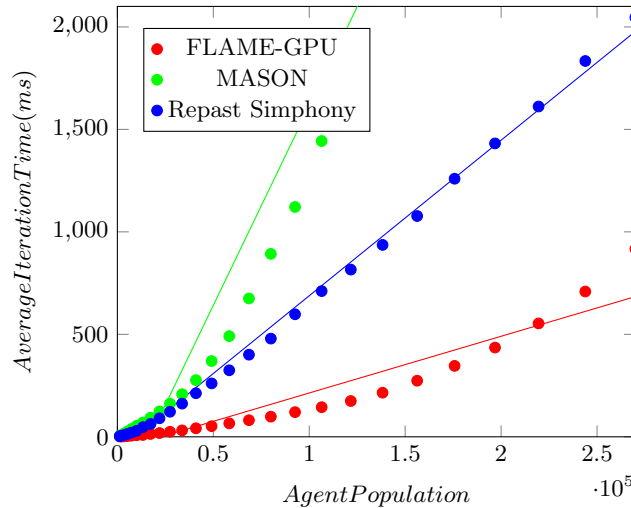


Figure 2. Graph plotting average iteration time of each framework against the agent population. Full parameters used are provided in Table 1.

however showed that performance for each framework remained stable throughout all parameters, with no discernible pattern.

6 Conclusion

The work within this paper has provided 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 3 tested ABS frameworks in response to changing agent populations and neighbourhoods. This provides a basis for those looking to implement ABS reliant on FRNNs, however this work would benefit from expansion to include more frameworks and standalone FRNNs search 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)

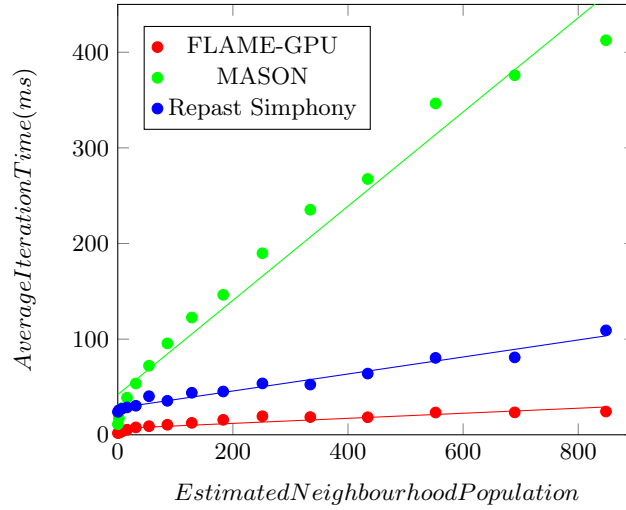


Figure 3. Graph plotting average iteration time of each framework against the expected neighbourhood population. Full parameters used are provided in Table 1.

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. North, M.J., Collier, N.T., Ozik, J., Tatara, E.R., Macal, C.M., Bragen, M., Sydelko, P.: Complex adaptive systems modeling with repast symphony. *Complex adaptive systems modeling* 1(1), 1–26 (2013)
7. Richmond, P., Romano, D.: A high performance framework for agent based pedestrian dynamics on gpu hardware. *Proceedings of EUROSIS ESM* (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), 2015 44th International Conference on*. pp. 81–85. IEEE (2015)