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, Sheffield, UK
{r.chisholm,p.richmond,s.maddock}@sheffield.ac.uk

Abstract. Many agent based models require agents to have an awareness of their local peers. The handling of these fixed radius near neighbours (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 64-bit floating point performance of each framework to scale linearly with agent population, in contrast the GPU accelerated framework's 32-bit floating point performance only became linear after maximal device utilisation around 100,000 agents.

Keywords: Parallel agent based simulation · OpenAB · Benchmarking · Fixed radius near neighbours · FLAMEGPU · MASON · Repast simphony

1 Introduction

Many complex systems have mobile entities located within a continuous space such as: particles, people or vehicles. Typically these systems are represented via Agent Based Simulations (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 simulation, 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.

F. Desprez et al. (Eds.): Euro-Par 2016 Workshops, LNCS 10104, pp. 311–321, 2017.
DOI: 10.1007/978-3-319-58943-5_25

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 [3,5,11]. However without a rigorous standard to compare implementations, exposing their relative benefits is greatly complicated.

With ABS 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 ABS 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 ABS 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 [10]. 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 the single node agent modelling frameworks: FLAMEGPU, MASON and REPAST Simphony. Single machine frameworks have been targeted as they provide a simpler and more accessible platform than distributed for initial development. This work has been published to the OpenAB website[2] and provides a foundation for the future assessment of ABS frameworks.

The results within this paper assess each framework's FRNNs implementation against the metrics of problem size and neighbourhood size, 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 when working with 32-bit floating point data.

The remainder of this paper is organised as follows: Sect. 2 provides an overview of related research; Sect. 3 lays out a clear specification of the circles benchmark model and how it can be utilised effectively; Sect. 4 details the frameworks which have been assessed using the benchmark; Sect. 5 discusses the results obtained from the application of the circles benchmark to each framework; Finally Sect. 6 presents the concluding remarks and directions for further research.

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

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 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, in order to achieve high performance in a problem as general as FRNNs they must sacrifice accuracy [6].

The naive approach 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 (Fig. 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 cell's agents is produced. This allows the Moore neighbourhood[3] 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 [4] 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

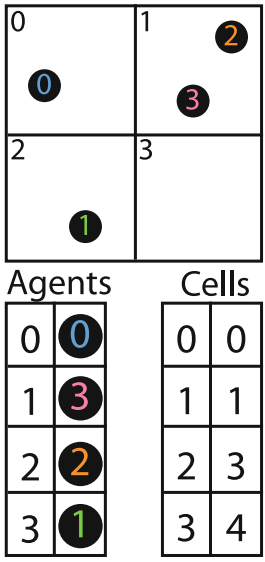

Fig. 1. A representation of a data structure that can be used for uniform spatial partitioning. The Cells table denotes the index within the agents table that data for the corresponding cell begins.

[3] The collection of cells inclusively bounded by the ring of adjacent cells surrounding the target cell.

[3]; Hoetzlein considered the effect of changing the partition cell dimensions [5]; and Sun et al. proposed the use of a parallel ordered sort to improve sorting efficiency [11].

Recent FRNNs publications have either provided no comparative performance results, or simply compared with their prior implementation lacking the published innovation [3,5,11]. 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. 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 [7].

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 between 0 and $W - 1$ in each axis.[4] Each particle's motion is driven by forces applied from other particles within their local neighbourhood, with forces applied between particles to encourage a separation of r.

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

[4] All frameworks tested utilised an environment of $0 <= x < W$, as it is not possible to cleanly clamp a floating point value within a less than bound, the nearest valid whole number was instead used to ensure the correct operation of each framework.

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

Parameter	Description	Fig. 2	Fig. 3
k_{rep}	The repulsion dampening argument. Increasing this value encourages agents to repel	1×10^{-3}	1×10^{-3}
k_{att}	The attraction dampening argument. Increasing this value encourages agents to attract	1×10^{-3}	1×10^{-3}
r	The radial distance from the particle to which other particles are attracted. Twice this value is the interaction radius	5	1–15
ρ	The density of agents within the environment	1×10^{-2}	1×10^{-2}
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

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 Eq. 1.[5] Initially the particle agents are randomly positioned within the environment of diameter W and E_{dim} dimensions.

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

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 Eq. 2, whereby F_i denotes the force exerted on the agent i as calculated by Eq. 3.[6] Within Eq. 3 F_{ij}^{rep} and F_{ij}^{att} represent the respective attraction and repulsion forces applied to agent i from agent j. The values of F_{ij}^{att} and F_{ij}^{rep} are calculated using Eqs. 4 and 5 respectively, the relevant force parameter is multiplied by the distance from the force's boundary and the unit vector from x_i to x_j in the direction of the respective force. After calculation, the agent's location is then clamped between 0 and $W - 1$ in each axis.

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

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

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

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

$$\overrightarrow{F_{ij}^{att}} = k_{att}(2r - \|\overrightarrow{x_j x_i}\|)\frac{\overrightarrow{x_j x_i}}{\|\overrightarrow{x_j x_i}\|} \tag{4}$$

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

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.

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

```
vec myOldLoc;
vec myNewLoc = myOldLoc;
float r2 = 2*RADIUS;
foreach neighbourLoc
{
  vec toVec = neighbourLoc-myOldLoc;
  float separation = length(toVec);
  if(separation < r2)
  {
    float k = (separation<RADIUS)?REP_FORCE:ATT_FORCE;
    toVec = (separation<RADIUS)?-toVec:toVec;
    separation = (separation<RADIUS)?separation:(r2-separation);
    myNewLoc += k * separation * normalize(toVec);
  }
}
myNewLoc = clamp(myNewLoc, envMin, envMax);
```

Validation. There are several checks that can be carried out to ensure that the benchmark has been implemented correctly, the initial validation techniques rely on visual assessment. During execution if the forces F_{att} & F_{rep} are both positive particles can be expected to form spherical clusters. Due to the force drop-off (switching from the maximal positive force, to the maximal negative force) when a particle crosses the force boundary, these clusters oscillate, this effect is amplified by agent density and force magnitude. If these forces are however both negative, particles will spread out, with some particles overlapping each other.

More precise validation can be carried out by seeding two independent implementations[7] with the same initial particle locations. With appropriate model parameters (such as those in Table 1), it is possible to then export agent positions after a single iteration from each implementation[8]. Comparing these exported

[7] The implementations used within this paper are available within this projects repository. https://github.com/Robadob/circles-benchmark.

[8] It is recommended to export agents in the same order that they were loaded, as sorting diverged agents may provide inaccurate pairings.

positions should show a parity to several decimal places, whilst significant differences between the initial state and the exported states. Due to the previously mentioned force fall-off and floating point arithmetic limitations, it was found that a single particle crossing a boundary between two models, snowballs after only a few iterations, causing many other particles to differ between simulation results.

The 3 agent framework implementations tested within this paper were all tested with shared initial particle locations states to ensure that their models were performing the same operations.

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 Eq. 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 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 Eq. 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). Many implementations are unaffected by changes to this value. However optimisations such as those by Sun et al. [11] 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 Eq. 3. There are many parameters which impact this speed within the circles model. As a particles motion is calculated as a result of the sum of vectors to neighbours it clear that the parameters affecting neighbourhood size (ρ & r) impact particle speed in addition to the forces F_{att} & F_{rep}.

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 thread (SIMT) execution. This is caused by sparse neighbourhood threads spending large amounts of time idling whilst waiting for larger neighbourhood threads searching simultaneously within the shared thread-group to complete. 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[9] and further details regarding the frameworks can be found on the OpenAB website[10]. The frameworks targeted within this research are:

- Inspired by the FLAME agent-based modelling framework, FLAMEGPU was developed to utilise GPU computation via a combination of XML and CUDA [10].
- 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 [8].
- The Repast collective of modelling tools has now been under development for over 15 years. Repast Simphony targets computation on individual computers and small clusters, facilitating the development of agent-based models using Java and Relogo [9].

Notably FLAMEGPU supports the usage of both 32-bit and 64-bit floating point values, whereas both MASON and Repast Simphony use 64-bit floating point values exclusively within their frameworks. This is likely influenced by the negative impact 64-bit floating point values have on GPU performance being significantly greater to that of CPUs.

5 Results

Results presented within this section were collected on a single machine running Windows 7×64 with a Quad core Intel Xeon E3-1230 v3 running at 3.3 GHz[11].

[9] https://github.com/Robadob/circles-benchmark.
[10] http://www.openab.org/benchmarks/simulators/.
[11] The processor supports hyper-threading, enabling 4 additional concurrent logical threads.

Additionally the FLAME-GPU framework utilised an Nvidia GeForce GTX 750 Ti GPU which has 640 CUDA cores running at 1 GHz.

Each of the parameter sets utilised targeted a different performance metric identified in Sect. 3.2. 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 near neighbours search carried out within each iteration.

The results in Fig. 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 both the FLAMEGPU implementations, which utilise GPU computation as opposed to the other frameworks which utilise a multi-threaded CPU approach, consistently outperform the best multi-core framework by a margin which at the largest test-case increases to greater than 6x with 64-bit floating point computation and 10x with the lower precision 32-bit floating point. This is slightly better than the expectations of GPU accelerated computation [7], suggesting their may be further room for optimisation. Although MASON and Repast Simphony are both Java based frameworks, Repast's performance trailed that of MASON by around 3x, investigating this showed Repast's separate operations for updating a particle's spatial and grid locations to be slower than that of MASON which handles both in a single operation. Notably the operation of updating a particles location could not be handled in parallel by MASON or Repast.

The MASON, Repast and 64-bit floating point FLAMEGPU results both have a Pearson correlation coefficient (PCC) [2] of 0.99. This is indicative of a linear relationship. Similarly 32-bit floating point FLAMEGPU has a PCC of 0.99 when only agent populations of 100,000 and higher are considered, this suggests that smaller agent populations did not fully utilise the GPU during 32-bit floating point computation.

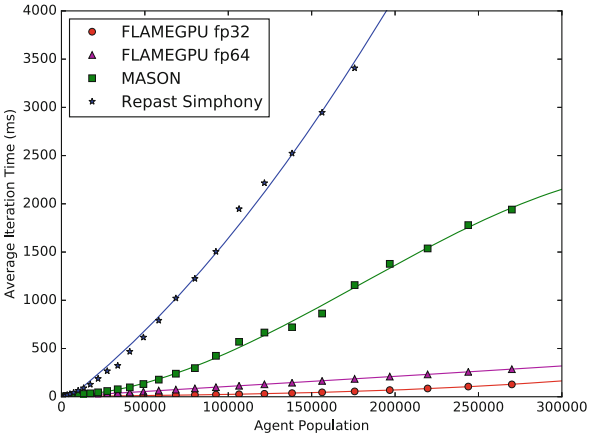

Fig. 2. The average iteration time of each framework against the agent population.

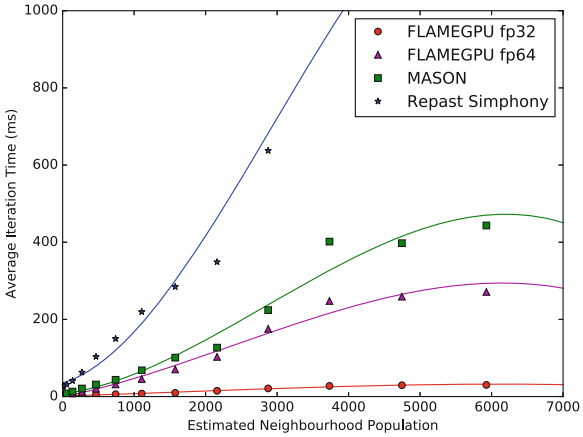

Fig. 3. The average iteration time of each framework against the estimated neighbourhood population. The estimated neighbourhood population is the calculation of agents within a neighbourhood where agents are uniformly distributed, providing a clearer interpretation of changes to the interaction radius (r).

The next parameter set, shown in Fig. 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. All results have a PCC [2] of 0.96. This is indicative of a linear relationship, albeit much weaker correlation than that seen within the prior experiment. It is likely that this weaker relationship can be explained by how the agent density becomes more non-uniform as the model progresses, causing the number of agents within each neighbourhood to grow.

The final parameter set assessed variation in performance in response to increased entropy. This is was 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 no substantial relationship between increased particle speed and performance.

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 agent entropy. The results within this paper have shown the linear performance relationships of the tested ABS frameworks in response to changing agent populations and neighbourhood sizes. This provides a guide for those looking to implement ABS reliant on FRNNs and a metric to improve FRNNs search implementations.

The next stages of this research are: further evaluation of standalone FRNNs implementations utilising the most recent research advances, improving the benchmark model to further isolate assessment criteria of FRNNs and reduce the effects of force fall-off, developing a statistical method of validating model outputs, assessing how distributed systems affect the scalability of FRNNs and considering the implications of wrapped (torodial) environments.

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. Edwards, A.: The correlation coefficient. In: An Introduction to Linear Regression and Correlation, pp. 33–46 (1976)
3. 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)
4. Green, S.: Particle simulation using cuda. Nvidia whitepaper **6**, 121–128 (2010)
5. Hoetzlein, R.: Fast fixed-radius nearest neighbors: interactive million-particle fluids. In: GPU Technology Conference (2014)
6. Kofler, K., Steinhauser, D., Cosenza, B., Grasso, I., Schindler, S., Fahringer, T.: Kd-tree based n-body simulations with volume-mass heuristic on the GPU. In: 2014 IEEE International on Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 1256–1265. IEEE (2014)
7. 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)
8. 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. IEEE Micro **24**(1), 42–51 (2004)
9. North, M.J., Collier, N.T., Ozik, J., Tatara, E.R., Macal, C.M., Bragen, M., Sydelko, P.: Complex adaptive systems modeling with repast simphony. Complex Adapt. Syst. Model. **1**(1), 1–26 (2013)
10. Richmond, P., Romano, D.: A high performance framework for agent based pedestrian dynamics on GPU hardware. In: European Simulation and Modelling, vol. 3 (2008)
11. 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: 44th International Conference on Parallel Processing Workshops (ICPPW), pp. 81–85. IEEE (2015)