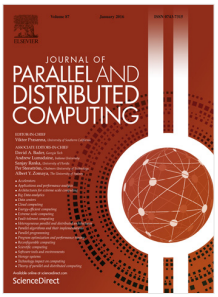
Journal Pre-proof



Improved GPU near neighbours performance for multi-agent simulations

Robert Chisholm, Steve Maddock, Paul Richmond

PII: S0743-7315(19)30134-0
DOI: https://doi.org/10.1016/j.jpdc.2019.11.002
Reference: YJPDC 4147

To appear in: *J. Parallel Distrib. Comput.*

Received date : 12 February 2019
Revised date : 14 June 2019
Accepted date : 5 November 2019

Please cite this article as: R. Chisholm, S. Maddock and P. Richmond, Improved GPU near neighbours performance for multi-agent simulations, *Journal of Parallel and Distributed Computing* (2019), doi: https://doi.org/10.1016/j.jpdc.2019.11.002.

Improved GPU Near Neighbours Performance for Multi-Agent Simulations

Robert Chisholm[a], Steve Maddock[a], Paul Richmond[a]

[a]*Department of Computer Science, The University of Sheffield, Regent Court, 211 Portobello, Sheffield, S1 4DP, UK*

Abstract

Complex systems simulations are well suited to the SIMT paradigm of GPUs, enabling millions of actors to be processed in fractions of a second. At the core of many such simulations, fixed radius near neighbours (FRRN) search provides the actors with spatial awareness of their neighbours. The FRNN search process is frequently the limiting factor of performance, due to the disproportionate level of scattered memory reads demanded by the query stage, leading to FRNN search runtimes exceeding that of simulation logic. In this paper, we propose and evaluate two novel optimisations (Strips and Proportional Bin Width) for improving the performance of uniform spatially partitioned FRNN searches and apply them in combination to demonstrate the impact on the performance of multi-agent simulations. The two approaches aim to reduce latency in search and reduce the amount of data considered (i.e. more efficient searching), respectively. When the two optimisations are combined, the peak obtained speedups observed in a benchmark model are 1.27x and 1.34x in two and three dimensional implementations, respectively. Due to additional non FRNN search computation, the peak speedup obtained when applied to complex system simulations within FLAMEGPU is 1.21x.

Highlights

- We propose two optimisations to FRNN search for complex system simulations, one reducing latency in search and one reducing the amount of data considered. The optimisations can also be combined.
- These optimisations have been applied to a standalone benchmark model, in both two and three dimensions, and to three different models within FLAMEGPU.
- The peak speedup for FRNN search for the combined optimisation applied to a range of models varies between 1.21x and 1.34x.

Keywords: GPU, CUDA, Parallel algorithms, Complex systems.

1. Introduction

Fixed Radius Near Neighbours (FRNN) search is central to many complex systems simulations, from molecular dynamics to crowd modelling. In these systems, the simulated actors often interact with their local neighbours, e.g. particles, people or vehicles, which might require awareness of the locations, velocities and/or directions of all neighbouring actors within a specific radius. FRNN search is used to perform this query across data points within a fixed radius of the target location. Within complex systems simulations on Graphics Processing Units (GPUs) the query occurs in parallel, allowing every entity to survey their neighbours simultaneously.

The technique of Uniform Spatial Partitioning (USP) is most commonly used to provide efficient FRNN searches on GPU hardware. We can consider two stages to the FRNN search process, construction and query. The USP data structure is constructed using the optimised GPU primitive operations sort and scan. The query stage is then possible with minimal branch divergence, which is optimal for the cohesive thread execution within vector units of Single Instruction Multiple Threads (SIMT) GPU architectures.

Despite USP being a highly appropriate and well adapted technique, the FRNN search is still typically one of the most expensive operations of simulation, often requiring more processing time than any compute bound model logic. Like many GPU algorithms, the query stage of FRNN search is bounded by latency, where maximal hardware utilisation is not achieved by either compute operations or memory transfers. This has led researchers to seek out techniques to improve the speed at which FRNN queries can be executed [1, 2, 3, 4].

Actors within Multi-Agent Systems (MAS), such as

Email address: {r.chisholm,p.richmond,s.maddock}@sheffield.ac.uk (Paul Richmond)

crowd modelling, are often influenced by neighbours from a greater distance than those observed in models such as Smoothed-Particle Hydrodynamics (SPH) [5]. Additionally, high density actor populations lead to more neighbours within each radial neighbourhood and thus greater numbers of memory transactions. Hence MAS performance is especially impacted by the cost of FRNN search.

This paper presents two independent techniques applicable to FRNN queries on GPUs using the USP data structure. We refer to these as *Strips* and *Proportional Bin Width*. Strips reduces the cost of bin accesses during the FRNN query and the Proportional Bin Width technique reduces redundant memory accesses within FRNN queries. These two optimisations can also be combined to enable the benefits of both techniques simultaneously.

In combination, these two techniques reduce latency within the query stage of FRNN search by optimising code performance and reducing redundant data accesses via the adjustment of bin widths. These optimisations are implementation agnostic, suitable for application to a broad range of USP problems. Hoetzlein briefly considered the trade-off of adjusting bin widths in his earlier research [3]. However, we take this further, performing both a theoretical assessment of the impact of adjusting bin widths and demonstrating the effects in practice.

Our results assess the performance according to the metrics of population size and density in a benchmark model representative of a broader class of physical (e.g. SPH, MAS) simulations in both two and three dimensions. The tested configurations show improvements across a wide collection of actor densities and distributions, as is likely to be found within complex system simulation domains. To demonstrate this wide applicability, the optimisation has been applied to three models within FLAMEGPU: Boids, a 3D flocking model; Pedestrian, a 2D crowd model; Keratinocyte, a 3D cellular biological model, representative of skin tissue. The results demonstrate that the technique is beneficial to the performance of FRNN search across actor distributions, densities and even between 2D and 3D models.

The remainder of this paper is organised as follows: Section 2 provides an overview of available techniques for performing FRNN searches, the technique of USP and prior techniques for its optimisation; Section 3.1 describes the theory and an example implementation of the Strips technique, for the reduction of compute within USP accesses; Section 3.2 describes the theory and an example implementation of the Proportional Bin Width technique, for the reduction of redundant memory accesses within FRNN search; Section 4 details how the technique has been implemented for experimentation; Section 5 explains the benchmark models that have been used for evaluation; Section 6 discusses the results obtained when comparing performance before and after the optimisations have been applied; Finally, Section 7 presents the concluding remarks and directions for further research.

2. Near Neighbours on GPU

FRNN search is primarily used by complex systems simulations to allow spatially located actors to survey their neighbours. This process provides awareness of the environment to the underlying model, which is capable of using the neighbour data to inform the behaviour of each actor. This process is typically repeated once per timestep allowing each actor, e.g. a particle, to evaluate the forces affecting their motion. FRNN search should not be confused with the technique of 'nearest neighbour', which is concerned with finding a single result closest to the search origin. This benefits from different performance considerations when compared with FRNN searches, which, for example, may return as many as 80 results per query when applied to SPH [5].

There are many techniques capable of managing spatial data [6], primarily utilising hashing [7, 8], trees [9, 10] and Voronoi diagrams [11, 12]. However, these data structures target a range of processes from nearest neighbour to intersection testing. As such they are not applicable or optimised to provide general FRNN search. Furthermore, transferring these data structures from serial implementations to the highly parallel SIMT architecture of GPUs requires consideration of an additional set of optimisations and techniques to fit within the data parallel programming model.

The naive approach to FRNN search is brute force evaluation of pairwise interactions. However, this becomes unsustainable with performance quickly declining as the number of actors increases. Whilst hierarchical spatial data structures such as kd-trees can be accelerated on GPUs, and have been used for N-Body simulations [13], they are more suited for tasks requiring greater precision of data selection, such as collision testing and ray-tracing, which are concerned with the single nearest neighbour [14].

2.1. Uniform Spatial Partitioning

Spatial partitioning has become the standard for GPU FRNN search. There are now many software frameworks which provide implementations of spatial partitioning for either general complex simulations or for simulation within a specific domain, e.g. FLAMEGPU [15] (Multi-Agent Simulation), fluids3 [16] (SPH), LAMMPS [17] (Molecular Dynamics) & AMBER [18, 19] (Molecular Dynamics).

Under GPU USP the known environment is sub-divided into a uniform grid of bins, with each bin being given a consecutive identifier (such that in 2 dimensions $i = p_Y d_X + p_X$, where p is the grid position in the corresponding axis and d is the grid's dimensions). The location of stored data is clamped to fall within the known environment bounds. As the environment has been subdivided into a regular grid of bins, this allows the location's containing bin to be identified. All of the data to be stored in the USP data structure is then sorted and stored in order of their respective bin identifiers.

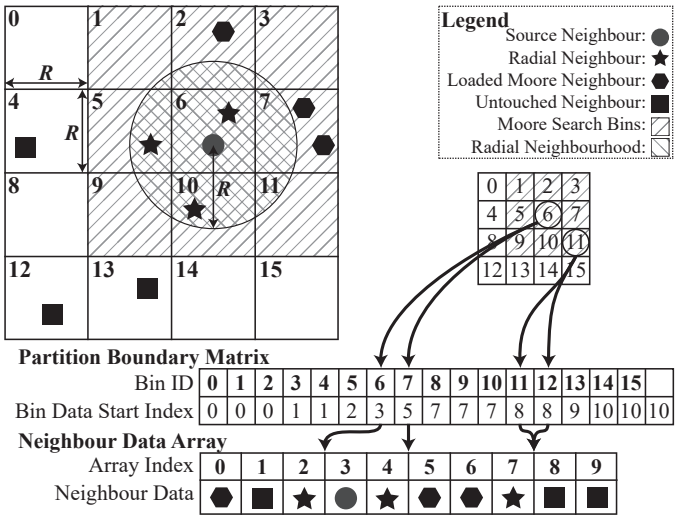


Figure 1: Visual representation of an environment and how its data is stored and accessed under GPU uniform spatial partitioning. The PBM acts as an index to each bin's portion of the neighbour data array.

So that neighbour data within a specific bin can be accessed efficiently, a Partition Boundary Matrix (PBM) is constructed. This structure provides an index to the start of each bin's data within the neighbour data array. When accessing the USP data structure to perform a FRNN query, a grid of bins encompassing the radial neighbourhood must be accessed. Accessing the contents of a bin therefore requires reading two values from the PBM, so that the memory range encompassed by the bin's data can be identified.

Figure 1 presents a two-dimensional example of the data structures used. Due to the compact nature of the data structure and the SIMT architecture of GPUs, if the containing bin of a single actor's data changes, the entire data structure must be reconstructed. However, the process for constructing both the neighbour data array and PBM is already highly data-parallel, utilising widely available parallel primitives such as sort and scan. When implemented with atomic counting sort, this produces the PBM as a by-product of the neighbour data array sort [3]. The time complexity of construction is reduced to constant-time [20, 21].

To access data located within the radial neighbourhood of a position, the position's containing environment bin is identified and all neighbour data stored within the bins of the inclusive Moore neighbourhood are then iterated. As shown in Figure 1, only those with a position inside the radial neighbourhood are considered by the simulation. In two dimensions this means that neighbour data within a spatial area 2.86x larger than required (2D neighbourhood area: πR^2, 2D Moore neighbourhood area: $(3R)^2$) are accessed. In three dimensions this increases to 6.45x (3D neighbourhood volume: $\frac{4}{3}\pi R^3$, 3D Moore neighbourhood volume: $(3R)^3$). Thus, in both 2D and 3D environments the majority of memory accesses can be assumed to be redundant.

2.2. Related Research

Research regarding the USP data structure has primarily considered improvements to the query. This can be considered as a result of the relatively low cost of construction in most applications, with respect to the query time. Other data structures such as 'neighbourhood grid' and Verlet lists have been proposed and used, however, the USP data structure remains most performant for GPU hardware.

Goswami et al were able to improve the performance of GPU FRNN queries during SPH on GPUs [1]. They adjusted the indexing of the bins, which the environment is subdivided into, so that they are indexed according to a Z-order space-filling curve (also known as a Morton code). The space-filling curve maps multi-dimensional data into a single dimension whilst preserving greater spatial locality than regular linear indexing. This creates power-of-two aligned square grids of bins, with contiguous Z-indices, such that bins containing neighbour data are stored in a more spatially coherent order. This additional locality intends to ensure that more neighbourhoods consist of contiguous blocks of memory, such that surplus data in cache lines is more likely to be required in subsequent requests, reducing additional latency caused by cache evictions.

Similarly, the AMBER GPU molecular dynamics toolkit, as described by Salomon-Ferrer et al [19], uses particle sorting within bins according to a 4x4x4 Hilbert space-filling curve in order to calculate the forces between molecules during simulation. In addition to possible benefits of spatial locality, this allows them to extend the neighbourhood cut off, reducing the quantity of redundant accesses to neighbourhood data. Their research does not address the implementation or performance impact of this optimisation independently. Subdividing bins into a further 64 sub-bins may only be viable for densities where the sub-bin occupancy remains close to 1.

Hongyu et al optimised the reconstruction of the data structure by developing a novel sorting technique that takes advantage of the knowledge that particles can only move to neighbouring bins, utilising a prefix sum of the changes to each bin's capacity [4]. They were able to improve performance in a small SPH simulation of 8192 particles within a 16^3 grid. However, overall performance was equal to that of their initial unoptimised reconstruction when applied to larger simulations. Section 5 shows how in larger simulations the construction time is a small fraction of that required for the FRNN query. As such, optimisation efforts targeting the query process are likely to have a greater overall impact.

Previously, Hoetzlein replaced Radix sort with (atomic) counting sort [3] within the construction of the USP data structure. This significantly simplified the construction of Green's original implementation [22] by reducing the number of construction kernel launches from 12 to 1. This optimisation greatly improved the performance by around

1.5-8x (for various population sizes). More recent GPU architectures have improved the performance of atomic operations at a hardware level [23], which has likely further improved the technique.

Verlet lists have also been used as a potential optimisation, allowing a list of neighbours within the radial search area to be stored per actor [24]. However, this technique relies on an additional step on top of USP and FRNN search to produce the data-structure [25]. Other construction algorithms have also been proposed [26]. Furthermore, the primary value of verlet lists lies in reducing the need to reperform the neighbour list construction, only expiring a neighbour list when the central actor moves a set distance. Therefore, the cost of using verlet lists is highly dependent on the frequency of list reconstructions required. Additionally, storage of per-agent neighbour lists will inflate memory requirements, significantly in models with the most populous neighbourhoods. This limits the applicability of verlet lists to systems of both low entropy and scale [27].

Joselli et al described a novel data structure they called neighbourhood grid, inspired by USP [28]. Their data structure, designed for SPH, instead assigns each particle to a unique bin, rather than allowing multiple particles to share each bin. This binning system instead creates an approximate spatial neighbourhood which can be queried in constant time. This approximate method has been reported to improve performance up to 9x over exact methods based on USP, by ensuring neighbourhoods are a fixed 26 particles (the Moore neighbourhood). However, they are not as generally applicable or accurate as FRNN search, due to the fixed neighbourhood volumes.

The research in this section has demonstrated an opportunity for improved query performance for a more general case of FRNN search, through improving accesses to bins and reduction of message accesses.

3. Innovations

This paper presents two independent techniques applicable to FRNN queries on GPUs using the USP data structure. We refer to these as *Strips* and *Proportional Bin Width*. Strips reduces the cost of bin accesses during the FRNN query and the Proportional Bin Width technique reduces redundant memory accesses within FRNN queries. These two optimisations can also be combined to enable the benefits of both techniques simultaneously. Section 3.1 focuses on Strips. Section 3.2 focuses on Proportional Bin Width. Section 3.3 focuses on the combined technique.

3.1. Strips

The Strips optimisation targets redundant bin changes, removing constant time operations which contribute to latency during the query's GPU kernel's execution. In 2 dimensions, the Moore neighbourhood of a query's origin's containing bin consists of a 3x3 block of bins. These nine bins exist as three strips of three bins, where each strip's storage exists contiguously in memory. In 3 dimensions, the 27 bin Moore neighbourhood, is formed of nine strips of three bins. Within a strip, only the first bin's start index and the last bin's end index need to be loaded from the PBM to identify the range within the neighbour data array that contains the neighbour data from each of the strip's bins. This optimisation only affects how bins are accessed, leaving the data structure described in Section 2 unchanged.

In the example shown in Figure 1, each row of the Moore neighbourhood would become a strip, i.e. bins 1-3, 5-7 and 9-11 would be treated as strips.

The SIMT architecture of GPUs avoids divergence between threads in each synchronous thread group (warp). Therefore it is assumed that when threads within the same warp are accessing different bins, which will most often have differing sizes, all threads within the warp must wait for the thread accessing the largest bin to complete before continuing to the next bin. By merging contiguous bins the Strips optimisation reduces the number of these implicit synchronisation points.

Additionally, the use of Strips can only reduce the total difference between threads with the least and most messages read per warp, hence reducing the quantity and duration of idle threads. It is not possible for the total difference to increase under this optimisation.

Algorithm 1 Pseudo-code showing the FRNN search's query algorithm before optimisation for a two dimensional environment.

```
vector2i absoluteBin = getBinPosition(origin)
loop -1<=relativeBin.x<=1:
  loop -1<=relativeBin.y<=1:
    selectedBin = absoluteBin + relativeBin
    if selectedBin is valid:
      hash = binToHash(selectedBin)
      loop i in range(PBM[hash],PBM[hash + 1]):
        handle(neighbourData[i])
```

Algorithm 2 Pseudo-code showing the Strips optimisation applied to the query operation for a two dimensional environment.

```
vector2i absoluteBin = getBinPosition(origin)
loop -1<=relativeStrip<=1:
  startBin = absoluteBin + vector2i(-1, relativeStrip)
  endBin = absoluteBin + vector2i(1, relativeStrip)
  startBin = clampToValid(startBin)
  endBin = clampToValid(endBin)
  startHash = binToHash(startBin)
  endHash = binToHash(endBin)
  loop i in range(PBM[startHash],PBM[endHash + 1]):
    handle(neighbourData[i])
```

This optimisation essentially removes two bin changes from every central strip and one from every boundary strip. This occurs three times per neighbourhood in two dimensions and nine times in three dimensions. Therefore the optimisation provides a near constant time speed up through removal of code, related to the problem dimensionality and number of simultaneous threads. Addition-

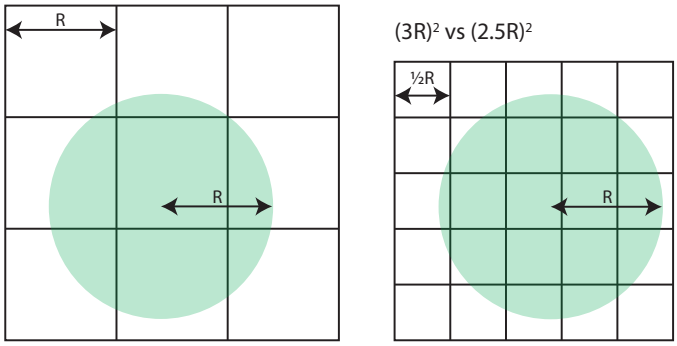


Figure 2: Visual representation of how different bin widths require different search areas to cover the whole radial neighbourhood.

ally, the removal of the bin switches, as discussed above, is likely to reduce branch divergence, whereby threads within the same warp are operating on differently sized bins and hence would change bins at different times. Algorithm 1 shows psuedo-code for the original FRNN search's query technique prior to optimisation and Algorithm 2 after the Strips optimisation has been applied, removing one of the loops and introducing an additional bounds check.

3.2. Proportional Bin Widths

Standard implementations of USP subdivide the environment into bins with a width equal to that of the radial search radius. This decision ensures that all radial neighbours will be guaranteed to be found in the same or surrounding bins of the query's origin. This is referred to as the Moore neighbourhood. The implication of this decision is that many surplus messages outside of the radial neighbourhood must be accessed. In 2D the Moore neighbourhood accessed is 2.86x that of the radial neighbourhood. In 3D this becomes 6.45x.

By adjusting the width of the bins relative to the neighbourhood radius, the number of bins that must be accessed (to ensure the entire radial neighbourhood is covered) and the volume of individual bins change. Thus the total (and surplus) area of the environment accessed also changes. Figure 2 provides an example of how alternate proportional bin widths change these values.

Hoetzlein [3] mentioned a similar optimisation in his work, however, he did not suggest an optimal proportion, only considering the trade-off between bin volume and bins per query. Our calculations give a wider consideration of the theoretical impact of adjusting the proportional bin widths.

The min (a_g) and max (a_G) grid areas of access under any proportional bin width in 2 dimensions can be calculated using equations 1 & 2, where R is the chosen search radius and W the absolute bin width.

$$a_g = \left\lceil \frac{2R}{W} \right\rceil^2 W^2 \tag{1}$$

$$a_G = \left(\left\lceil \frac{2R}{W} \right\rceil + 1 \right)^2 W^2 \tag{2}$$

Proportional Bin Width	**1**	**0.7**	**0.5**
Min Grid Area (a_g)	4	4.41	4
Max Grid Area (a_G)	9	7.94	6.25
Average Grid Area (a_a)	9	4.84	6.25
Bin Count Max ($\left(\left\lceil \frac{2R}{W} \right\rceil + 1\right)^2$)	9	16	26
Strip Count Max ($\left\lceil \frac{2R}{W} \right\rceil + 1$)	3	4	5
Max Surplus Mult. ($\frac{a_G}{a_r}$)	2.86	2.50	1.99
Average Surplus Mult. ($\frac{a_a}{a_r}$)	2.86	1.54	1.99

Table 1: This table provides an example of the variability by changing the bin width as a proportion of the search radius of 1 (with a corresponding radial search area of 3.14) in 2D. These values can be calculated using Equations 1 to 4.

By additionally calculating the proportion within the min grid area (p) and max grid area (p'), in a 1 dimensional implementation, Equation 4 can be used to calculate the average grid area access (a_a).

$$p = \frac{2R - \left\lfloor \frac{2R}{W} \right\rfloor W}{W}$$
$$p' = 1 - p_m \tag{3}$$

$$a_a = p^2 a_g + 2pp' \sqrt{a_g}\sqrt{a_G} + p'^2 a_G \tag{4}$$

Dividing by the radial neighbourhood area (a_r) then provides the surplus multiplier, which denotes the search grid's area relative to the radial neighbourhood area. This represents the level of redundant memory access where data is uniformly distributed.

$$a_r = \pi R^2 \tag{5}$$

Table 1 provides a demonstration of the above equations 1-5 applied to several differing proportional bin widths. Furthermore, these equations can be extended to 3 dimensions or more, as required, as they are constructed from the 1-dimensional mathematics surrounding the radial area (2R) divided by the bin width (W).

Some bin widths lead to different bin counts dependent on the query origin's position within its own bin (see Figure 3, where the query origin can lead to either a square or rectangular search grid). Due to the SIMT architecture of the GPU, all 32 threads of each warp will step through the maximum number of bins any individual thread within the warp requires, although this may lead to some threads remaining idle through later bins. Similarly, it is expected that threads processing surplus messages (which do not require handling by the model), will incur a runtime cost similar to that of the desired messages which are likely to incur additional compute.

Some combinations of proportional bin width and query origin do permit for corners of Moore neighbourhoods to be skipped, as they don't intersect with the radial neighbourhood (See Figure 4). Analysis suggests that in most cases the additional compute necessary to detect and skip

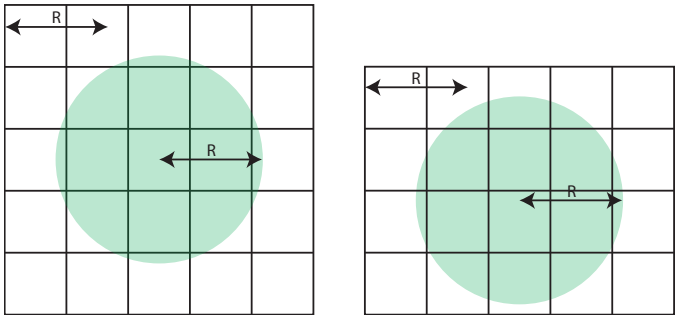


Figure 3: Visual representation of how under some bin widths the query origin affects the size of the required search area to cover the whole radial neighbourhood. R denotes the neighbourhood radius.

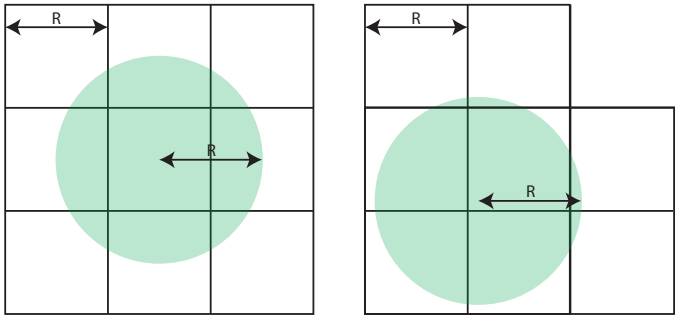


Figure 4: Visual representation of how, under some bin widths and search origins, corner bins do not always need to be accessed. R denotes the search radius.

the few redundant corner bins would incur a prohibitive runtime cost, outweighing any performance savings.

Section 3.1 demonstrated that there is a cost to individual bin accesses. The Strips technique is about optimising the trade-off between redundant area accessed and total bins accessed. The removal of W^2 from equations 1 & 2, to calculate min (a_g) and max (a_G) area can be used to calculate the number of bins. Similarly, taking the square root of these values provides the number of Strips, following the optimisation from Section 3.1.

A further consideration of increasing the number of bins is that this leads to a larger partition boundary matrix. This both requires more memory for storage and increases construction time. These impacts are considered further in Sections 5 and 6.

Algorithm 3 Pseudo-code showing the Proportional Bin Width optimisation applied to the query operation for a two dimensional environment.

```
vector2i binMin = getBinPosition(origin - radius)
vector2i binMax = getBinPosition(origin + radius)
loop binMin.x<=selectedBin.x<=binMax.x:
  loop binMin.y<=selectedBin.y<=binMax.y:
    hash = binToHash(selectedBin)
    loop i in range(PBM[hash],PBM[hash + 1]):
      handle(neighbourData[i])
```

Algorithm 3 demonstrates how the Proportional Bin Width optimisation may be applied by altering the selection of bins iterated, in contrast to that of Algorithm 1. It can be seen that in the 2D example the nested loop changes from iterating a 3x3 Moore neighbourhood to an NxM Moore neighbourhood. The exact values of N and M are dependent on the values returned by `getBinPosition()`, which clamps the continuous space coordinates to within the environment bounds and then returns the containing discrete bin coordinates.

3.3. Combined Technique

The two optimisation techniques described in sections 3.1 and 3.2 can be combined, allowing the Strips optimisation to reduce the impact of bin changes under the Proportional Bin Width's optimisation. This is important, as the act of decreasing the proportional bin width decreases the volume of bins, hence most often increasing the number of bins to be accessed.

Algorithm 4 Pseudo-code showing the combined optimisation applied to the query operation for a two dimensional environment.

```
vector2i binMin = getBinPosition(origin - radius)
vector2i binMax = getBinPosition(origin + radius)
loop binMin.y<=strip<=binMax.y:
  startBin = vector2i(binMin.x, strip)
  endBin = vector2i(binMax.x, strip)
  startHash = binToHash(startBin)
  endHash = binToHash(endBin)
  loop i in range(PBM[startHash],PBM[endHash + 1]):
    handle(neighbourData[i])
```

Algorithm 4, demonstrates how Algorithms 2 and 3 can be combined to utilise the benefits of each optimisation simultaneously during the query operation. The nested loop over `selectedBin` from Algorithm 3 has been replaced by the strip loop and validation from Algorithm 2

4. Implementation

To demonstrate the impact of the optimisation, it is first applied to a small standalone implementation which enables greater fidelity for analysis. Second, it is applied to FLAMEGPU to demonstrate how the optimisations apply to existing multi-agent simulations. The remainder of this section details the implementation of the standalone version, providing further context to figure 1 (available online[1]).

4.1. Construction

Only the Proportional Bin Width (and combined) optimisations impact construction. Their impact is limited to affecting the scale of the environment's subdivision, subsequently increasing (or decreasing) the memory footprint of the Partition Boundary Matrix (PBM).

Figure 1 highlights how the PBM consists of a single unsigned integer array, with a length one greater than the number of environmental bins ($N_{bins} + 1$). Each entry into

[1] https://github.com/Robadob/sp-2018-08

the array, except for the final entry, identifies the first index of the corresponding environmental bin's messages in the neighbour data array. The final element of the array denotes the total number of messages. The layout of the PBM allows the bounds of an environmental bin's data in the neighbour data array to be identified by accessing both the index of the desired bin and the subsequent index within the PBM.

4.1.1. Algorithm

First, an array of equal length to the neighbour data array is created. In this array, each (spatially located) message from the neighbour data array has its containing environmental bin's index stored. Equation 6 can be used to transform a spatial coordinate located within the environment to its corresponding environmental bin's coordinate (*gridPos*). This can then be transformed to the bin's 1-dimensional index using either equation 7 for a 2D environment, or equation 8 for a 3D environment.

$$gridPos = \left\lfloor gridWidth \frac{envPos}{envWidth} \right\rfloor \tag{6}$$

$$gridId_{2D} = (gridPos_y * gridWidth_x) + gridPos_x \tag{7}$$

$$gridId_{3D} = (gridPos_z * gridWidth_y * gridWidth_x) + gridId_{2D} \tag{8}$$

Figure 5 illustrates how the PBM is constructed from the neighbour data and neighbour bin index arrays. The neighbour bin indices must be used to calculate the offset of each bin's storage. This can be achieved by first producing a histogram representing the number of messages within each bin. Both implementations utilise atomic operations to produce this histogram, as proposed by Hoetzlein [3]. The histogram is then passed to a primitive scan using the exclusive sum operator. This functionality is available within libraries such as CUB [29]. For the PBM to be produced correctly, the PBM must have a length one greater than the total number of bins, so that the final value in the PBM denotes the total number of neighbour data.

Finally, the neighbour data array is sorted, so that neighbour data are stored in order of their environmental bin indices. This can be achieved using a primitive pair sort, such as that found within CUB [29], and a kernel to subsequently reorder the neighbour data.

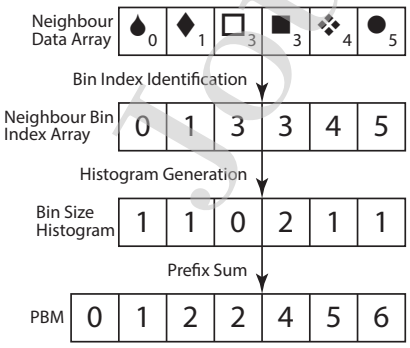


Figure 5: The arrays used to generate a PBM.

4.2. Query

Whilst the neighbour data from a single bin can be identified with two accesses to the PBM, to perform a query a contiguous block of bins must be accessed. This contiguous block of bins must include all bins which may intersect the search area. The number of bins accessed can be calculated using equation 9. When $PBW = 1.0$, a 3x3(x3) block of bins must be accessed, and when $PBW = 0.5$, a 5x5(x5) block of bins must be accessed.

$$bins = (1 + 2\left\lceil \frac{1}{PBW} \right\rceil)^{Dims} \tag{9}$$

4.2.1. Algorithm

The FRNN search's query algorithm can be described using the psuedocode in algorithm 5, which combines the earlier algorithms 2 and 3. Each GPU thread operates independently, concerned with its own unique FRNN query. In this two dimensional example we loop over `relativeStrip` which represents the Y axis, the rows of the block of bins. In three dimensions this would be a nested loop over the Y and Z axes. The absolute bin coordinate (`absoluteBin`) is then combined with `relativeStrip` and offset positively and negatively in the X axis by `RAD`. This provides the start and end coordinates of the current strip of bins to be accessed. These values must be clamped within a valid range and transformed into 1-dimensional bin indices (using equations 7 or 8). After this, they can be used to access the PBM to identify the range of elements within the storage array that must be accessed.

Algorithm 5 Pseudo-code showing the FRNN search algorithm before optimisation for a two dimensional environment.

```
vector2i absoluteBin = getBinPosition(origin)
uint RAD = ceil(1.0/PBW)
loop -RAD<=relativeStrip<=RAD:
  startBin = absoluteBin + vector2i(-RAD, relativeStrip)
  endBin = absoluteBin + vector2i(RAD, relativeStrip)
  startBin = clampToValid(startBin)
  endBin = clampToValid(endBin)
  startHash = binToHash(startBin)
  endHash = binToHash(endBin)
  loop i in range(PBM[startHash],PBM[endHash + 1]):
    handle(neighbourData[i])
```

This algorithm accesses all messages within potentially intersecting bins. The locations of messages must additionally be compared to only handle those which lie within the radial area of the search's origin.

FLAMEGPU's algorithm for handling bin iteration is implemented in a different way. The query state is tracked and updated each time a message is retrieved. This implementation abstracts the behaviour from users, allowing them to access FRNN queries within their model logic using a single while loop. Despite these differences, the actual order of message access remains consistent.

To ensure maximal performance of FRNN search queries, it is necessary to limit the kernel's block size to 64 threads. This maximises hardware utilisation, whilst reducing the number of threads remaining idle, where message distributions are not uniform (hence leading to an unbalanced workload). The optimisation is visible in the NVIDIA-distributed CUDA particles example[2], however, details of this technique do not seem to appear in general literature pertaining to FRNN search.

5. Experimental Configuration

The experiments have been designed to demonstrate six important characteristics of this research:

- Experiment 1: FRNN Search Query - That FRNN search's query is an expensive operation within a range of existing multi-agent simulation models.
- Experiment 2: Strips - That the strips optimisation provides a constant time speedup to the query operation with respect to neighbourhood size (density).
- Experiment 3: Proportional Bin Width - That the Proportional Bin Width optimisation (Section 3.2) in combination with the Strips optimisation provides further improvements to the query operation's performance by reducing surplus message accesses.
- Experiment 4: Construction - That the cost of USP construction is low relative to performing queries, thus supporting our focus on optimising queries.
- Experiment 5: Population Scaling - That the optimisations presented do not harm the linear scaling of query operations with respect to increasing population size.
- Experiment 6: Model Testing - That the optimisations presented retain their benefits when applied to existing multi-agent simulations within FLAMEGPU.

Within these experiments, construction refers to the construction of the USP data structure, which involves sorting messages and generating an index to the start of each bin's storage (PBM). Whereas, the query stage refers to the neighbourhood search performed by each actor in parallel, this includes any model specific logic that occurs per neighbour accessed.

The experimental implementation[3] follows the generalised case observed in many frameworks and examples, e.g. FLAMEGPU [15] and the NVIDIA CUDA particles sample [22][4]. The targeting of the general case ensures optimisations are more widely applicable, whilst still remaining relevant when applied to real models.

The Circles model [30](explained in Section 5.1) has been used to evaluate the impact of the optimisations presented in Sections 3.1 and 3.2 on the performance of FRNN search. The Circles model allows parameters that can affect performance scaling, such as population size and population density, to be controlled. The presented optimisations have been evaluated in both two and three dimensions.

Additionally, FLAMEGPU has been modified[5] so that the optimisations presented within this paper can be applied to a range of different models in experiments 1 and 6. The following models from FLAMEGPU have been used:

Boids (Partitioning) - this provides an implementation of Reynolds flocking model [31], which provides an example of emergent behaviour represented by independently operating birds in a 3D environment.

Pedestrian (LOD) - this provides an implementation of a random walk pedestrian model. Collision avoidance between pedestrians is handled via an implementation of the social forces model in a 2D environment, as described by Helbing and Molnar [32]. The particular environment has a low density of pedestrians.

Keratinocyte - this provides a cellular biological model representative of Keratinocyte cells, which are the predominant cell found on the surface layer of the skin. This model utilises two instances of the USP data structure, which both operate in a 3D environment. The Keratinocyte model was modified slightly to decrease the resolution of the force USP. This both allowed the model to support a higher agent population and improved performance 2x, without affecting the model's operation.[6]

5.1. Circles Model

The Circles model is an extension of that proposed by Chisholm et al [30] as a means for benchmarking FRNN search using a simple particle model analogue, typical of those seen in cellular biology and multi-agent simulation benchmarking. The extension modifies the force calculation to utilise the sine function as a means of smoothing. The addition of smoothing reduces the applied forces about the equilibrium position, such that the particle jitter about the equilibrium position from the initial model is eliminated. Additionally this has made it possible to merge the attraction and repulsion parameters.

The new force calculation formula takes the form $F_{ij} = sin(-2\pi(\frac{d_{ij}}{r}))F$, where d_{ij} is the scalar distance between

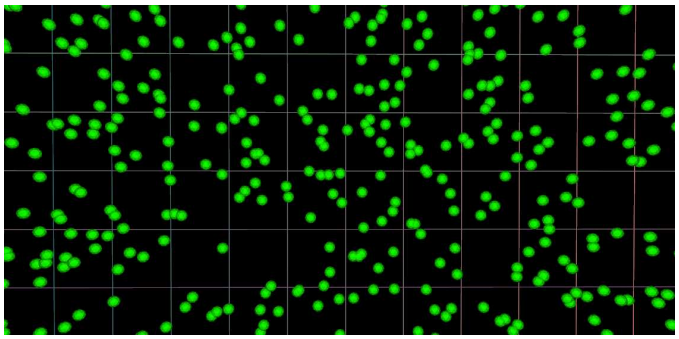
[2]The CUDA particles example is included as a sample alongside installation of the CUDA toolkit.

[3]https://github.com/Robadob/sp-2018-08

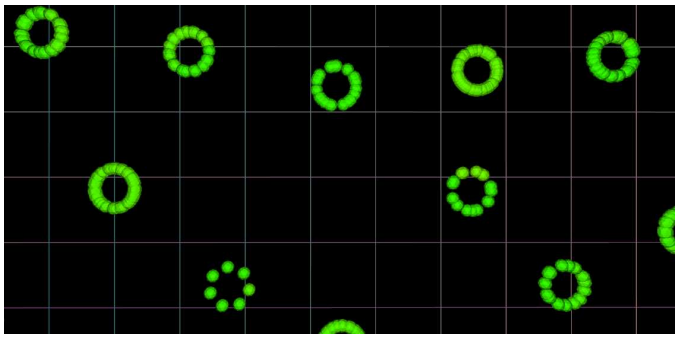
[4]The CUDA particles sample code is available on installation of the CUDA toolkit.

[5]https://github.com/Robadob/FLAMEGPU (Changes are present in the branch titled 'PBW')

[6]The original force resolution structure had around 400 bins per actor, most of which would therefore remain unused. This was reduced to around 1 bin per actor. It is possible that greater reductions would further improve performance, however, that is outside the scope of this paper.



(a) Uniform Random Initialisation State



(b) Circles End State

Figure 6: An area from the visualisation of the state of the Circles model. The spheres represent point based actors. The grid lines show the subdivision of the environment into bins. (a) The start state under uniform random initialisation whilst executing in 2-dimensions with a Moore neighbourhood volume average of 37. (b) The end state whereby the model has converged to a steady state with a Moore neighbourhood volume average of 41.

source particle i and neighbour particle j, r is the interaction radius and F is the unified force dampening argument. F_{ij} is subsequently multiplied by the normalised direction vector from i to j. As with the original model, F_{ij} is calculated for all neighbour particles with the sum of the resulting value providing the final offset that is applied to the source particle.

The particles (the actors) within this model move with each time step. Initially they are uniform randomly distributed, as shown in Figure 6a. During runtime they move to create clusters of particles arranged in circles in two dimensions (Figure 6b) and spheres in three dimensions. This has the effect of creating hotspots within the environment where bins contain many particles and dead zones where bins contain few or no particles. This structured grouping can be observed in complex systems such as pedestrian models, where a bottleneck in the environment creates a non-uniform distribution within bins. These clusters also lead to the Moore neighbourhood volumes more closely matching the radial neighbourhood volumes.

The uniform random initialisation, shown in Figure 6a, utilises the CUDA device function `curand_uniform` to position neighbours within the environment bounds, essentially producing a noisy distribution with roughly equal numbers of actors per bin. To ensure consistency when using uniform random initialisation, the same seed value was used to initialise comparative benchmarks between builds. However, the same performance trends have been observed across multiple seed values.

Figure 6b shows a visualisation of an area from the steady state at the convergence of a two-dimensional configuration. In three dimensions the arrangement of particles would form hollow spheres instead of the rings shown in the figure. Parameters are managed so that they can be controlled independently. As the density parameter is increased, the number of particles per grid square and ring increases, and the environment dimensions decrease. Changes to the actor population size simply affect the environment dimensions, so as to not change the density. During results collection visualisations were not used to avoid any impact to performance. Each configuration has been executed for 200 iterations. Visual testing showed that with a unified force dampening argument of 0.05 this number of iterations was required to consistently progress through the model to reach the steady state.

Uniform random initialisation has been used for the experiments as this provides an average of the workloads seen in many complex systems with mobile entities. Due to the diversity in distributions seen within complex systems, selecting a particular sparse/dense distribution model (such as Perlin noise) has been avoided, favouring the Circles model [30] to represent the behaviours and distributions found within complex systems, as it moves actors through many of these spatial distributions during execution.

6. Results

Benchmarks of a USP implementation, before and after the application of the optimisations presented in Sections 3.1 and 3.2, were collected using the Circles model presented in Section 5. Data in this section was collected using an NVIDIA Titan-X (Pascal) in TCC mode with CUDA 9.1 on Windows 10. Additionally, GPU boost was disabled using nvidia-smi to limit clock and memory speeds to their stock values (1417MHz and 5005MHz, respectively).

The following sections correspond to the experiments outlined in Section 5.

6.1. Experiment 1: FRNN Search Query

To evaluate the cost of FRNN search's query operation when applied to a variety of models, FLAMEGPU has been extended to output the average runtimes of all agent functions. This enables the proportion of the query as a result of the total model runtime to be assessed. To most accurately reflect processes, timings include all operations (e.g. memory copies) which occur around the agent function's kernel launch. One timing is then returned per agent function. These have then been summed according to message input (query), message output (construction) and not relevant (not included in table 2, which is why percentages do not add to 100).

Within Table 2, it is clear that FRNN search dominates the runtime, requiring over 50% of the execution in all instances (52%-91%). In contrast, the construction of the USP data structure occupies 29% and 33% in two instances, and also occupies as little as 9% under the Boids model. The Pedestrian and Keratinocyte proportions do not sum to 100% due to additional agent functions occupying runtime.

6.2. Experiment 2: Strips

To assess the impact of the Strips optimisation on the performance of the query operation, the Circles model was executed using both the original FRNN search implementation and an implementation with the Strips optimisation. With an agent population of 1 million in 2D (Figure 7a), query time increases linearly with agent density. Throughout, the queries under the Strips optimisation perform a stable 1-1.2ms faster than that without the optimisation. A similar pattern is visible in 3D (Figure 7b). Here the Strips optimisation performs a stable 2.3-3.5ms faster than that without the optimisation. This roughly three-fold increase is in line with the reduction in bin changes, which reduces from 9 to 3 in 2D and 27 to 9 in 3D, reductions of 6 and 18, respectively. Due to the stable speed up, the proportional speedup decreases significantly as the radial neighbourhood size increases. Of note, the implementation tested (detailed in Section 4) switches bins using a nested loop, with one for loop per dimension and a central loop to iterate messages within each bin. For the Strips optimisation, the x axis's for loop is removed (see Algorithms 1 and 2).

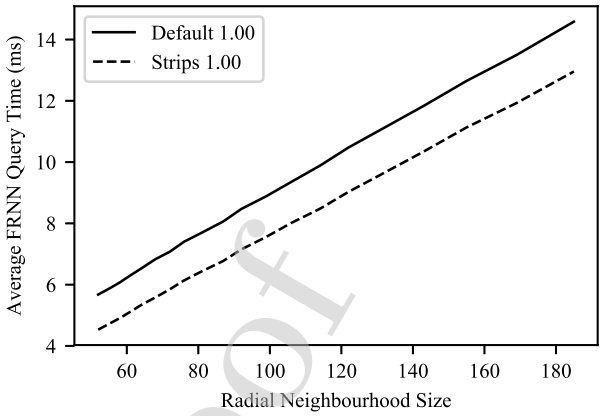
6.3. Experiment 3: Proportional Bin Width

Experiment 2 was repeated using the Strips implementation, which has a proportional bin width of 1.0, and the proportional bin widths (0.7, $\frac{2}{3}$, 0.5, 0.4). These were selected based on an extended version of Table 1, as their properties had the lowest combined maximum strip count and surplus multipliers.

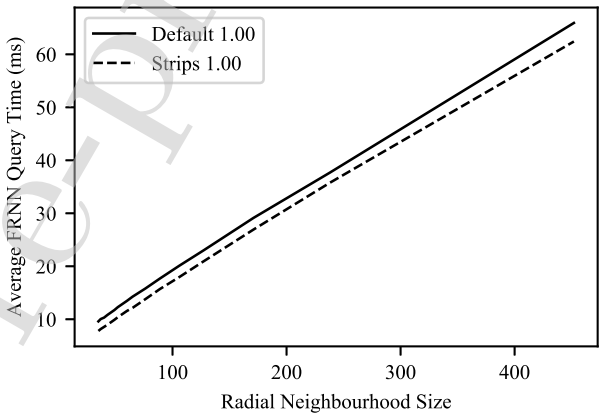
In 2D (Figure 8a), the successful proportional bin width found is 0.5. By reducing the surplus to 1.99x (equation and details available in Section 3.2), and only increasing the number of strips by one, it is able to improve on the performance of the original Strips implementation's query operation. This improvement provides a roughly stable 15% speed up over the original Strips implementation throughout all densities tested.

Model	Query %	Construction %
Boids	91%	9%
Pedestrian	52%	33%
Keratinocyte	65%	29%

Table 2: The runtime proportion of FRNN search and USP construction under 3 models within FLAMEGPU, when executed in their default configuration for 10,000 iterations.



(a) 2D

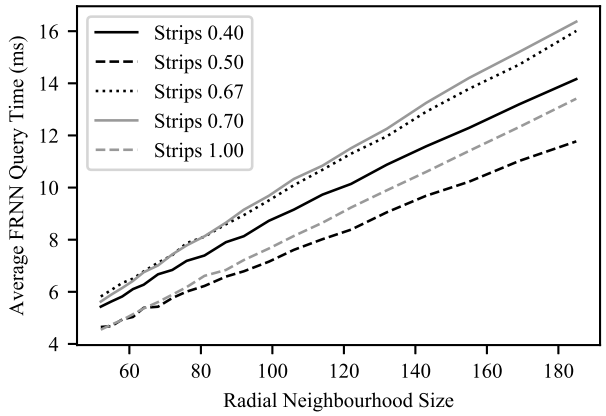


(b) 3D

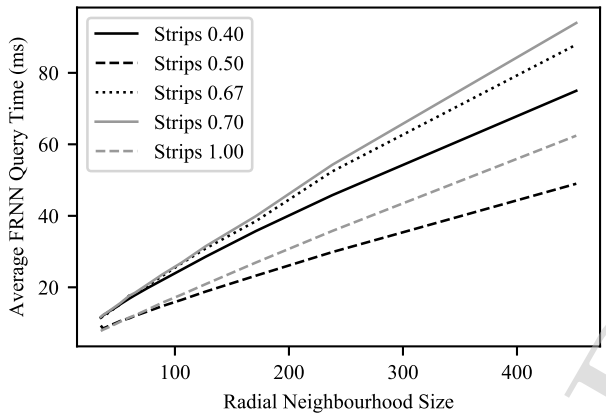
Figure 7: The performance of the query operations during execution of the Circles model with 1 million actors across a range of densities for both the original implementation and the Strips optimisation in (a) 2D and (b) 3D.

The remaining three proportional bin widths tested, 0.7, $\frac{2}{3}$ and 0.4, have worse performance than the original Strips implementation (proportional bin width of 1.0).

As expected, whilst a proportional bin width of 0.7 should have an average surplus of 1.54x (relative to the original's 2.87x), its performance is likely that of its maximum surplus of 2.50x coupled with the additional strip, due to the divergence caused by alternate search origins having varying numbers of bins to search. The proportional bin width of $\frac{2}{3}$ has an average surplus of 2.26x with no additional bin changes. Its lack of performance suggests that floating point precision limitations may have removed the primary benefit of $\frac{2}{3}$ being a factor of 2 (thus adding redundant bin changes). The proportional bin width of 0.4 also harmed performance, however, the runtime was much closer to that of the original Strips implementation. 0.4 is calculated to have average and maximum surpluses of 1.54x and 1.83x, respectively. However this comes at

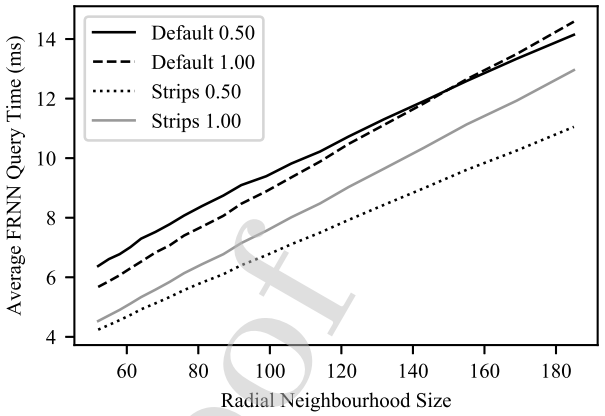


(a) 2D

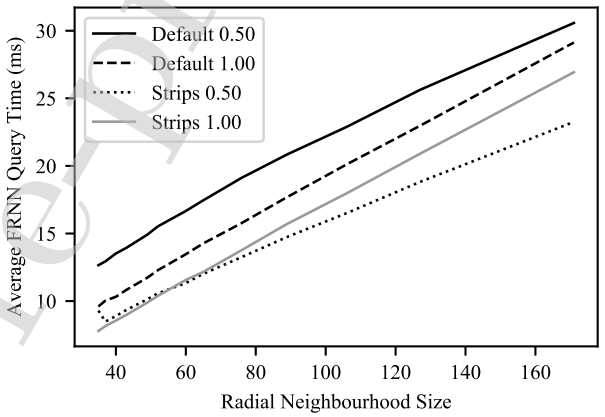


(b) 3D

Figure 8: The performance of the query operation during execution of the Circles model with 1 million actors, with the Strips optimisation applied in combination with a selected range of proportional bin widths, in (a) 2D and (b) 3D.



(a) 2D



(b) 3D

Figure 9: An extension of figure 7, to compare the performance of the query operation for the original, and the combined Strips and 0.5 proportional bin width optimised implementations during execution of the Circles model with 1 million actors across a selected range of proportional bin widths in (a) 2D and (b) 3D.

the cost of 5 strips rather than 3.

In 3D (Figure 8b), as seen in 2D, the proportional bin width 0.5 produces the best result, achieving a 27% speed up over the original implementation at the highest density.

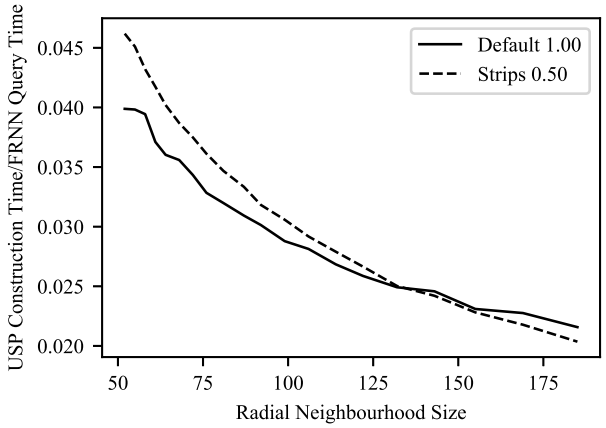
When compared with the original implementation, the Strips optimised implementation with a proportional bin width of 0.5 (Figure 9) allows the query operation to execute 27% faster at the highest density in 2D. This increases to 34% in 3D. Similarly, the graph demonstrates how the combination of Strips and a proportional bin width of 0.5 exceeds the improvement of both optimisations in isolation.

6.4. Experiment 4: Construction

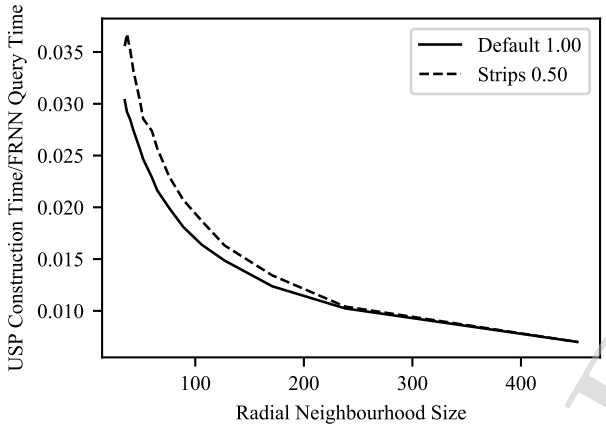
The construction execution time relative to the query operation times from the previous experiment (Figure 9) are shown in Figure 10. These demonstrate the small impact changes to construction time have versus query operation runtime. Construction accounts for 3% or less of the combined construction and query runtimes, reducing further as density and dimensionality increase. Regardless, by comparing times between Figures 9 & 10, it is shown that construction time is reduced slightly by the proportional bin width of 0.5. Due to the increased number of bins (4x in 2D, 8x in 3D), contention is reduced which appears to benefit the underlying sorting algorithm.

6.5. Experiment 5: Population Scaling

Figure 11 demonstrates how the query time of both the original and optimised FRNN search algorithms scales linearly as population size increases ($O(n)$). As discussed in section 3.2, the optimisations applied affect the number of bins accessed and the environmental scale of each bin.



(a) 2D



(b) 3D

Figure 10: A comparison of the construction time and query operation time of the original and the combined Strips and 0.5 proportional bin width optimised implementations during execution of the Circles model with 1 million actors across a selected range of proportional bin widths in (a) 2D and (b) 3D.

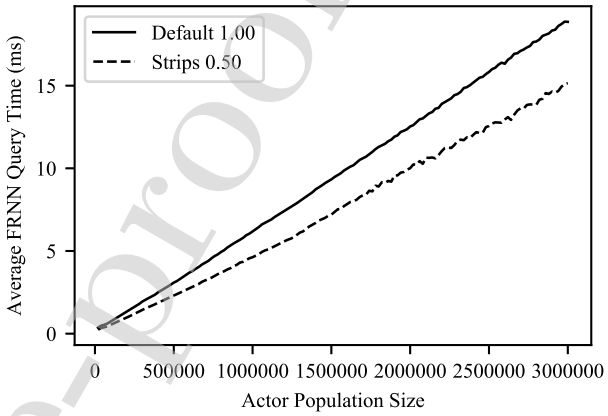
These two variables are constant with respect to a scaling population.

The jagged performance seen in Figure 11b is a consequence of the environment dimensions, locked to a multiple of bin width, (and consequently population density) not scaling as smoothly as the population size.

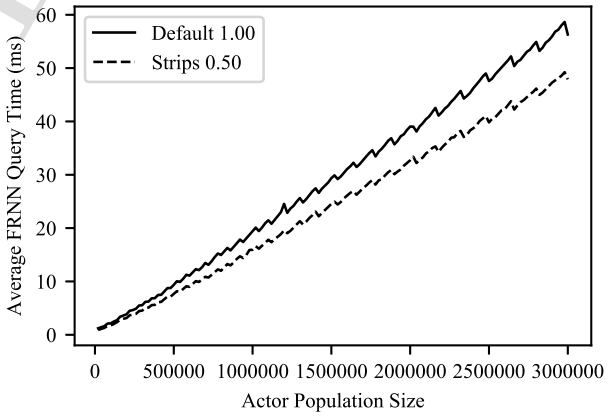
3,000,000 actors is adequate to fully utilise the computational capacity of the GPU used, such that the linear performance trend can be expected to continue until the GPU's memory capacity becomes a limiting factor [2]. In 2D the optimised version performs 1.23-1.33x faster. In 3D it is 1.18-1.22x faster.

6.6. Experiment 6: Model Testing

Three example models found within FLAMEGPU, introduced in section 5, have been utilised to demonstrate the benefit of the optimisation in practice. This applica-



(a) 2D



(b) 3D

Figure 11: Comparisons of the performance of the original and the combined Strips and 0.5 proportional bin width optimised implementations during execution of the Circles model across a range of actor population sizes in (a) 2D with radial neighbourhood volumes of ~60 and (b) 3D with radial neighbourhood volumes of ~100.

tion to differing models aims to highlight how differences between their application of FRNN search (e.g. actor density, 2D vs 3D) affect the optimisation.

The results in Table 3 show that the combined optimisation with a proportional bin width of 0.5 has allowed the query operation within the Boids model to operate 1.11x faster than the original's runtime. When combined with construction, which sees a slight increase, the overall runtime executes 1.15x faster than prior to optimisation. The Keratinocyte model sees a greater speedup of 1.32x for the query operations's runtime, reduced to 1.21x of the overall runtime. This greater speedup for the Keratinocyte model is likely to be as a consequence of the lower grid resolution applied to the model. The Keratinocyte model has several additional layers unaffected by FRNN search, reducing the proportional impact to the overall runtime.

Similarly, the only 2D model tested, Pedestrians, executed its query 1.03x faster than that of the original. This persisted to an overall speedup of 1.03x when compared with the original. Although the significance of this result is low, earlier results demonstrate that this can be attributed to the low density of actors within the model's structured environment.

7. Conclusions

This paper has presented two optimisations (Strips and Proportional Bin Width) to the USP data structure for providing FRNN searches. These optimisations can be applied individually or in tandem to improve the performance of queries to the data structure, at the relatively small cost of constructing a larger USP.

The results have shown that the Strips optimisation's performance is consistently faster than or equal to that of the original implementation of the query operation, which lacks the optimisations presented in this paper. The speedup was a constant 1-1.2ms per simulation step in 2D and 2.3-3.5ms in 3D across all densities with a population size of 1 million. When applied to low density simulations such as with ~52 neighbours in 2D, this produces a 1.22x speedup, due to the high number of bin changes (27, which Strips optimisation reduces to 9) relative to individual message accesses (~52). In line with the theory behind the optimisation, the speedup appears almost constant, relative to the number of dimensions, both due to the removal of redundant code, and its subsequent impact of reducing idle threads due to branching.

The combined optimisation of Strips and Proportional Bin Width can further improve performance in all but the cases of lowest density. The peak improvements measured were 15% and 27% in 2D and 3D, respectively. The results show how the combined optimisation scales to provide more benefit to both higher density agent populations and models with more compute demand per message processed. These improvements are due to the impact of reducing surplus message reads and thus idle threads within each warp. Our analysis and testing showed that the bin width ratio of 0.5x radius offers the best balance between surplus neighbour access reduction and total bin accesses.

This combined approach consistently outperforms the individual optimisations. Results showed the combined approach's execution to be as significant as 1.27x and 1.34x faster in 2D and 3D, respectively, when applied to the Circles benchmark model.

MAS utilise FRNN search in a wide variety of ways, with differing fidelity, actor distributions and dimensionality on top of each model's unique logic. These variables all impact the performance of FRNN search, often as a secondary consequence of the underlying GPU architecture. By demonstrating the performance improvements across 3 different models within the MAS simulation framework FLAMEGPU, we have shown how despite these differences the FRNN query executed as much as 1.32x faster. However, due to additional model logic, independent of the optimisation, the impact to overall runtime in this best case is reduced to a 1.21x speed up.

Due to the trade-off between surplus message reads and total bins accessed, it is imperative that the Strips optimisation is used alongside the Proportional Bin Width optimisation. Application of the Strips optimisation reduces the total bin access to $\sqrt{n}$ in 2D, and $\sqrt[3]{n^2}$ in 3D. This has a significant impact under the Proportional Bin Width optimisation. For example, in the case of a 0.5 proportional bin width the 2D Moore neighbourhood search grid expands from 3x3 to 5x5.

The results within this paper are confined to single GPU implementations. These are capable of representing models of millions of actors, more than necessary for many models whereby actors represent human populations. Yet there are models which can benefit from even greater populations. Current USP implementations can be transparently moved to either multi-GPU or paged off device memory platforms via the use of unified memory. However, maximising multi-GPU performance may require a manual approach tuned to optimally partition data according to a model's specific actor distribution patterns and environment.

This research has demonstrated a means for reducing the data accessed during FRNN search's query operation by decreasing the need to access messages outside of the radial neighbourhood. However, even after the Proportional Bin Width optimisation has been applied, 50% of accessed messages still remain redundant in a uniform distribution. Our future research will explore how reductions can be made to redundant memory accesses. Furthermore, specific applications of FRNN could be further improved by reducing the scope of message accesses, such as in pedestrian simulations, where a pedestrian may only have awareness of obstacles within their field of vision.

Acknowledgements

This research was funded by EPSRC grant id 1624970

		Query Time (s)			Construction Time (s)			Total Execution Time (s)		
Model	**Actors**	**Original**	**Optimised**	**Speedup**	**Orig.**	**Opt.**	**Speedup**	**Orig.**	**Opt.**	**Speedup**
Boids	262144	746.52	646.56	1.15x	5.10	7.02	0.73x	752.04	654.00	1.15x
Pedestrian	16384	1.01	0.98	1.03x	0.65	0.66	0.98x	2.25	2.18	1.03x
Keratinocyte	25000-41789	5.80	4.38	1.32x	0.94	1.87	0.50x	8.14	6.70	1.21x

Table 3: The runtime of 3 models within FLAMEGPU before and after the combined optimisation has been applied with a proportional bin width of 0.5, when executed for 10,000 iterations.

[1] P. Goswami, P. Schlegel, B. Solenthaler, R. Pajarola, Interactive sph simulation and rendering on the gpu, in: Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Eurographics Association, 2010, pp. 55–64.

[2] E. Rustico, G. Bilotta, A. Hérault, C. D. Negro, G. Gallo, Advances in multi-gpu smoothed particle hydrodynamics simulations, IEEE Transactions on Parallel and Distributed Systems 25 (1) (2014) 43–52. doi:10.1109/TPDS.2012.340.

[3] R. Hoetzlein, Fast fixed-radius nearest neighbors: Interactive million–particle fluids, in: GPU Technology Conference, 2014.

[4] H. Sun, Y. Tian, Y. Zhang, J. Wu, S. Wang, Q. Yang, Q. Zhou, A special sorting method for neighbor search procedure in smoothed particle hydrodynamics on gpus, in: Parallel Processing Workshops (ICPPW), 44th International Conference on, IEEE, 2015, pp. 81–85.

[5] T. Yang, R. R. Martin, M. C. Lin, J. Chang, S.-M. Hu, Pairwise force sph model for real-time multi-interaction applications, IEEE transactions on visualization and computer graphics 23 (10) (2017) 2235–2247.

[6] H. Samet, Foundations of multidimensional and metric data structures, Morgan Kaufmann, 2006.

[7] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, M. H. Gross, Optimized spatial hashing for collision detection of deformable objects., in: Vmv, Vol. 3, 2003, pp. 47–54.

[8] S. Lefebvre, H. Hoppe, Perfect spatial hashing, in: ACM Transactions on Graphics (TOG), Vol. 25, ACM, 2006, pp. 579–588.

[9] R. K. V. Kothuri, S. Ravada, D. Abugov, Quadtree and r-tree indexes in oracle spatial: a comparison using gis data, in: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, ACM, 2002, pp. 546–557.

[10] A. O. Finley, R. E. McRoberts, Efficient k-nearest neighbor searches for multi-source forest attribute mapping, Remote Sensing of Environment 112 (5) (2008) 2203–2211.

[11] M. T. Dickerson, R. S. Drysdale, J.-R. Sack, Simple algorithms for enumerating interpoint distances and finding k nearest neighbors, International Journal of Computational Geometry & Applications 2 (03) (1992) 221–239.

[12] D. Kim, D.-S. Kim, Region-expansion for the voronoi diagram of 3d spheres, Computer-aided design 38 (5) (2006) 417–430.

[13] R. J. Anderson, Tree data structures for n-body simulation, SIAM Journal on Computing 28 (6) (1999) 1923–1940.

[14] M. Shevtsov, A. Soupikov, A. Kapustin, Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes, in: Computer Graphics Forum, Vol. 26, Wiley Online Library, 2007, pp. 395–404.

[15] Flamegpu: Flexible large scale agent modelling environment for the gpu, http://www.flamegpu.com/ (1 2018).

[16] Fluids v.3: A large scale open source fluid simulator, http://fluids3.com/ (1 2018).

[17] Lammps molecular dynamics simulator, http://lammps.sandia.gov/ (1 2018).

[18] Amber home page, http://ambermd.org/ (1 2018).

[19] R. Salomon-Ferrer, A. W. Götz, D. Poole, S. Le Grand, R. C. Walker, Routine microsecond molecular dynamics simulations with amber on gpus. 2. explicit solvent particle mesh ewald, Journal of chemical theory and computation 9 (9) (2013) 3878–3888.

[20] N. Satish, C. Kim, J. Chhugani, A. D. Nguyen, V. W. Lee, D. Kim, P. Dubey, Fast sort on cpus and gpus: a case for bandwidth oblivious simd sort, in: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM, 2010, pp. 351–362.

[21] N. Leischner, V. Osipov, P. Sanders, Gpu sample sort, in: Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on, IEEE, 2010, pp. 1–10.

[22] S. Green, Particle simulation using cuda, NVIDIA whitepaper 6 (2010) 121–128.

[23] NVIDIA, Gp100 pascal whitepaper, Tech. rep., NVIDIA (2016).
URL https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

[24] L. Verlet, Computer" experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules, Physical review 159 (1) (1967) 98.

[25] J. A. Van Meel, A. Arnold, D. Frenkel, S. Portegies Zwart, R. G. Belleman, Harvesting graphics power for md simulations, Molecular Simulation 34 (3) (2008) 259–266.

[26] S. Páll, B. Hess, A flexible algorithm for calculating pair interactions on simd architectures, Computer Physics Communications 184 (12) (2013) 2641–2650.

[27] D. Winkler, M. Rezavand, W. Rauch, Neighbour lists for smoothed particle hydrodynamics on gpus, Computer Physics Communications 225 (2018) 140–148.

[28] M. Joselli, J. R. d. S. Junior, E. W. Clua, A. Montenegro, M. Lage, P. Pagliosa, Neighborhood grid: A novel data structure for fluids animation with gpu computing, Journal of Parallel and Distributed Computing 75 (2015) 20–28.

[29] Cub documentation, https://nvlabs.github.io/cub/index.html (11 2015).

[30] R. Chisholm, P. Richmond, S. Maddock, A standardised benchmark for assessing the performance of fixed radius near neighbours, in: European Conference on Parallel Processing, Springer, 2016, pp. 311–321.

[31] C. W. Reynolds, Flocks, herds and schools: A distributed behavioral model, ACM SIGGRAPH Computer Graphics 21 (4) (1987) 25–34.

[32] D. Helbing, P. Molnár, Social force model for pedestrian dynamics, Physical review E 51 (5).

Robert Chisholm received the masters degree in Computer Science from The University of Sheffield, UK, in 2014. He is currently working towards a PhD degree in the Department of Computer Science at The University of Sheffield, United Kingdom. His research interests include GPU accelerated algorithms, data structures and graphics, in particular with application to complex systems simulations.

Dr Steve Maddock received his PhD in computer science from the University of Sheffield, UK in 1999. He is Head of the Visual Computing research group in the Department of Computer Science at the University of Sheffield. His research interests include computer facial modelling and animation, AR and VR technology and applications, and sketch-based interfaces for simulation.

Dr Paul Richmond is an EPSRC Research Software Engineering Fellow and Senior Lecturer at the University of Sheffield, UK. His research focuses on facilitating the use of accelerated architectures such as Graphics Processing Units (GPUs) to accelerate scientific discovery, particularly around the development of software for complex systems simulation. He is the lead developer of the Flexible Large-scale Agent Modelling Environment for the GPU (FLAME GPU) and has collaborated internationally to apply this software to projects in cellular biology, ecology and transport simulation.

AUTHOR DECLARATION – Conflict of Interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed. We further confirm that the order of authors listed in the manuscript has been approved by all of us.

We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing we confirm that we have followed the regulations of our institutions concerning intellectual property.

We understand that the Corresponding Author is the sole contact for the Editorial process (including Editorial Manager and direct communications with the office). He/she is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs. We confirm that we have provided a current, correct email address which is accessible by the Corresponding Author and which has been configured to accept email from EviseSupport@elsevier.com .

Signed by all authors as follows:

13/6/2019

Robert Chisholm

13 June 2019.

Dr Steve Maddock

13/6/19

Dr Paul Richmond