Monte Carlo Simulation of Size-Enlargement Mechanisms in Crystallization using FLAME GPU

Abstract. In this work, we present a general method for implementing Monte Carlo simulation in both batch and continuous systems through two case studies implemented within the FLAME GPU simulation framework.

Table of Contents

Introduction

Monte Carlo simulation is a method of solving deterministic problems by repeated random sampling. The method is very useful is simulating systems with some degree of freedom such as fluids, crystallization, and etc. In order to achieve a higher accuracy, one may have to run the simulation for a long time or work with a large number of data set. The process is known to be time-consuming.

Improving the accuracy of the simulation, while accelerating the performance is achievable through utilising Graphics Processing Units (GPUs) many-core architecture. FLAMEGPU is a high performance GPU extension of FLAME framework that generates a simulation program from a model.

The purpose of this document is to show how multi-agent approach can be applied to Monte Carlo crystallization. We present a general method for implementing Monte Carlo simulation in both batch and continuous systems through two case studies (Batch and MSMPR) implemented within the FLAME GPU simulation framework. The two case studies have been replicated through parallel implementation of GPU.

1 Crystallization and Monte Carlo simulation

The term Monte Carlo was first used in 1930s in the calculation of neutron diffusion by Enrico Fermi. Monte Carlo simulation is referred to statistical approach that involves applying random numbers. It's been used in a wide range of fields such as Physics, Chemistry, Biology and etc. In this work, we will discuss Monte Carlo simulation of crystal population.

Crystals (from salt and sugar to diamonds) are familiar term to everyone. Crystallisation is an important process that may referred to building, separation, or particle formation process. Crystals or generally particles' properties are represented by internal (e.g size) and external (spatial location) coordinates. Note that the most important internal coordinate in crystallization is the size L that is the length of a crystal and external coordinate can be ignored in a mixed system. To verify and test the Monte Carlo simulation of crystallization process, any changes in particle size distribution (PSD) over time is examined.

Although there are various existing works on applying Monte Carlo approach to Crystal formulation, we are using the Monte Carlo approach described by Gooch and Hounslow [1]. To simulate a crystal population, the population has been setup within a two dimensional array where each row corresponds to one crystal, and each column represent an internal coordinate of that crystal. In other words, X_{ij} is referred to the jth internal coordinate of the ith crystal (row i, column j in the 2d array). The maximum size of the row is N_c, equal to the total number of crystals and is a fraction of the real number of crystals per unit volume, m_0.

The simulation is done over a large number of small time steps. To study the effect of three mechanisms of nucleation, growth, and aggregation, a series of array manipulation is done during each simulation step.

1.1 Nucleation

Nucleation in this concept is another term for crystal birth or formation of particles of size of 0. To simulate a nucleation, a new row is added to the 2d array and the value of the internal coordinate is set to a constant number that could be either zero, or a random number selected from a predetermined distribution.

1.2 Growth

Growth is a term used regarding the enlargement of a crystal by gradual incorporation of atom, ion, or molecules into crystal structure. Growth mechanism can be size independent, size dependant growth, or growth rate dispersion. In the first form of growth mechanism, growth rate is constant, whereas in the the second form, growth rate is a function of crystal size. In the third form, the growth rate is constant but different for each crystal.

To simulate crystal growth over a specific time, the new length (size) is calculated from the current crystal's size and growth rate [1]:

$$L_{new} = L_{old} + G\Delta t \tag{1}$$

, where Δt is a time interval, and G is a function of crystal coordinate. Note that relationship between G and the internal coordinate varies for a specific case study.

1.3 Aggregation

Aggregation term is used when two particles are to form one larger particle. Similar to the work by Gooch and Hounslaw [1], we only consider size independent aggregation. To simulate an aggregation, two crystals are randomly selected and removed to form an aggregate. Then, a new row to represent the aggregated crystal is added to the 2d array. The internal coordinate off the newly formed crystal is calculated as a function of the size of the two aggregated crystals [1]:

$$L_{agg} = (L_{a2}^3 + L_{a1}^3)^{1/3} \tag{2}$$

,where L_{agg} is the length of new crystal after aggregation of the two crystals with L_{a2} and L_{a1} sizes.

2 Multi agent formation with Monte Carlo

This work uses the base concepts similar to Gooch and Hounslow's paper [1]. However, we present a namely agent based approach to specifying the Monte Carlo simulation. Within the mult-agent approach, we use the term *agent* (with an assigned name name of *crystal*) to represent the properties of each row in the original MC 2d array. Each column/internal coordinate corresponds to an agent property of that agent. Properties therefore refer to attributes of the particle such as size (length of the crystal). The total number of agent crystals is referred to as agent population size N_c. The multi-agent formulation has a number of advantages which include; simple high level syntax to modify behaviour (equations), simple control of simulation parameters without changing the model (and therefore the simulation code) and targeting of simulation code to modern high performance architectures like the Graphics Processing Unit (via the FLAME GPU software).

3 Batch Simulation

In this document, we report performance results of batch crystallization, with constant supersaturation and no nucleation mechanism. Prior to the simulation, each row in the array is initialized in similar way to the nucleation mechanism. For example, the length (L) of the j th crystal is calculated from [1] :

$$L_j = F_L^{-1}(RND_j) \quad \text{for} \quad \text{j} = 1 \text{ to } N_c \tag{3}$$

, where $F_L(L)$ is the cumulative distribution. Note that in this work instead of sampling processes from the inverse cumulative distribution, we performed a random Monte Carlo sampling to initialize the length/size of the crystals.

After initializing the agent crystal internal coordinates, the effect of growth and aggregation mechanism on the crystal population is simulated for the time interval of Δt that is calculated from the aggregation rate β_0 [1] [1]:

$$\Delta t = \frac{2\Delta m_0}{\beta_0 m_0 (m_0 - \Delta m_0)} \tag{4}$$

, where, Δm_0 is the change in m_0 due to aggregation. The integer number of aggregation events per time step Δt is [1]:

$$AggNo = \frac{PopnNo \Delta m_0}{m_0} \tag{5}$$

, where *PopnNo* is the number of simulated crystals, initially being equal to N_c.

In this work, we do not have the exact values of some of the parameters (e.g volume). So, by assuming the volume to be 1, Δm_0 will be equal to *AggNo*, and m_0 will be equal to the total number of crystals. We simplify Eq. 4 and Eq. 5 to:

$$\Delta t = \frac{2AggNo}{\beta 0 PopnNo(PopnNo - AggNo)} \tag{6}$$

, where β_0 is equal to aggregation index I_{agg} defined by Hounlow [2].

After the aggregation, depending on the growth mechanism, crystals are grown in to a new size using Eq. 1. Note that the order of the two aggregation and growth mechanism does not have effect on the accuracy of the simulation.

At every simulation step, a new time interval Δt is calculated and the two process of growth and aggregation are repeated for every crystal. The overall simulation process is shown in Fig. 1. In order to show how the properties of the population change with respect to the time, the array contents will plot in a form of histogram by binning data by size.

[1] Constant number

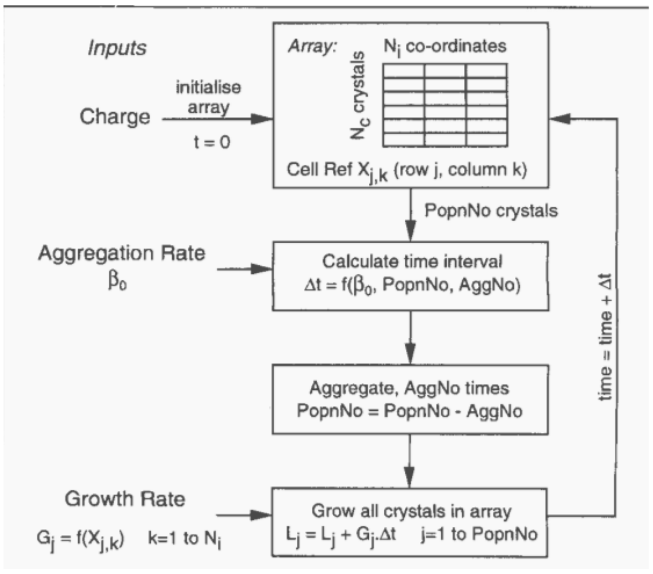

Fig. 1. Monte Carlo Batch Simulation outline, from [1]: The batch simulation processes sequentially applies a number of functions to perform aggregation and growth on the crystal structure array. The variable time-step is calculated before aggregation of growth.

4 General Batch Simulation with Multi-Agents

When formulating the Monte Carlo simulation within the constraints of an agent based simulator some functional changes to the behaviour described in Fig. 1 are required. These mostly require that the process of simulation is changed to reflect the methodology of a number of communicating agents. In doing this the problem is changed from a serial algorithm to a parallel one which facilitates simulation on highly parallel architectures like GPUs. In particular the process of aggregation must be modified to avoid serial updates whilst maintaining the correct behaviour (equivalent to that of the serial Monte Carlo approach). The overall simulation processes is summarised by Fig. 2 which shows the agent state diagram of how the model is processed with an multi-agent simulator, FLAMEGPU. The details of this process are explored further within this section.

FLAMEGPU is more normally used to output the entire population state after each iteration. Population states can then be post processed. To instead produce a histogram output, we defined a FLAMEGPU exit function *exitFunction* which counts the number of agent crystals in a particular bin. The agent bin is determined by the *simulate* function with the bin id stored as an agent variable *bin*. The *stepFunction* is utilised to calculate the Δt during each simulation run and is necessary as it varies per time-step according to 6.

In order to simulate the aggregation processes, we used ranked based (priority) method to replicate the random selection of crystals to form an aggre-

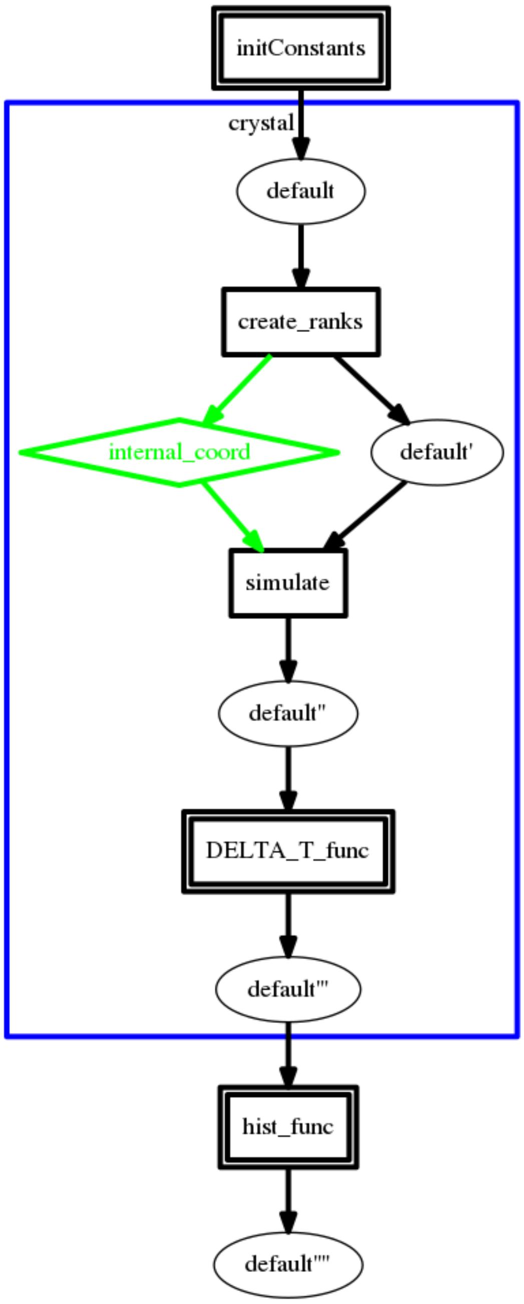

Fig. 2. Monte Carlo Batch Simulation State Diagram: Solid boxes denote a specific agent function which is applied to each agent in the population simultaneously. Circles represent states which act as synchronisation points ensuring that agent functions have completed across the entire population. Green diamond boxes represent a communication dependency in the form of messages. A simulation iteration is enclosed by the blue box, agents completing the processing of each of the functions in the state diagram will return to the *default* state after each simulation step until the specified number of simulation steps has completed. Double lined boxes represent special functions which can be applied either at the start of the simulation, before any simulation steps(i.e. *initConstants*), after each simulation step (i.e. *DELTA_ T_func*) or after completion of all simulation steps (i.e. *hist_func*).

gate. At every simulation step, each agent (crystal) generates a random rank in the form of a random number between [0.0, 1.0], 0.0 being the highest priority (and most likely the aggregate). Ranks are communicated to other agents via a *internal_ coord* message. During the *simulation* function, agents which are to perform aggregation are determined by comparing their ranks with the ranks of other agents and by tracking the number of agents with a lower rank than themselves. This allows an agent to determine its over ranked position in the entire population . If the agent counts less than `2*aggNo` [2] agents with ranks lower than itself then it has high enough priority to aggregate . Note: The *aggNo* determines the level of parallelism which can occur within the simulation. An *aggNo* of 1 is equivalent to the serial case.

The aggregation process is performed by aggregating immediately adjacent agents according to their rank. E.g. if *aggNo* is 2 then the two agents with lowest rank will aggregate to form a new crystal and the 2 agents with the 3rd and 4th lowest rank will aggregate to form a new crystal. In each case the agent with the lowest rank in the pair supersedes the other by applying the equation in Eq. 2 to calculate the new aggregated length. More generally agents with an odd rank position (which have been determined to aggregate) have highest priority. In order to implement this the rank comparison search of the *internal_ coord* messages tracks the closest lower rank and closest lower length i.e. the agent with a rank immediately lower than itself (Figure 3). Agents with an even rank position (which have been determined to aggregate) are removed from the simulation. Figure 4 illustrates the ranked based aggregation method. Rectangles with blue color are in fact agents in even rank position that are removed from the simulation.

5 Case Study 1 (aggregation only) Results

In this section, we examine a test case where only one internal coordinate (crystal length L) is considered. Moreover, we only simulate the aggregation mechanism in batch systems by considering a G_0 value of 0. The aggregation is size independent, and the *AggNo* is given at the beginning of the simulation. For this case study, we initialized the array with crystal lengths using a simplified equation based on Eq. 7:

$$n(L) = \frac{N_0 z^2}{L_0} exp(-z^3) \tag{7}$$

, where N_0 is the initial number of crystals, z is L/L_0, and L_0 is the cube root of the mean volume of the charge. In contrast to Gooch and Hounslow [1], we ignore scaling terms such as L_0 and volume. So, we assumed L_0 to be 1 and removed N_0 from the equation (the scaling terms). We do not have a way to calculate such values, and do not have experimental results to tune these parameters to. The overall effect of this is only to change the scale of the simulation,

[2] AggNo equal to 1, means two crystals will form an aggregate

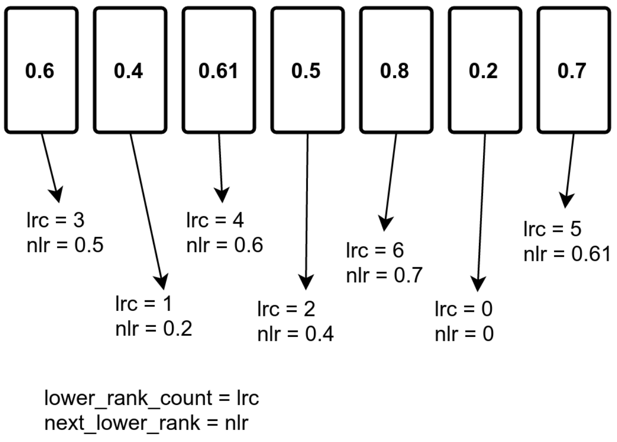

Fig. 3. Multi-agent batch simulation ranking processes: The ranking process is used to assign a priority to agents. Agents are effectively sorted by comparing the ranks of other agents in the population. The *lower_ rank_ count* value is used to store the number of agents encountered which have a lower rank. This lower rank count determines the sorted position of an agent according to the ranks. By tracking the *next_ lower_ rank* and the associated crystal length, agents are also able to know the values of the immediately preceding agent in the sorted rank list. This information can be used to then perform the aggregation as shown in Figure4.

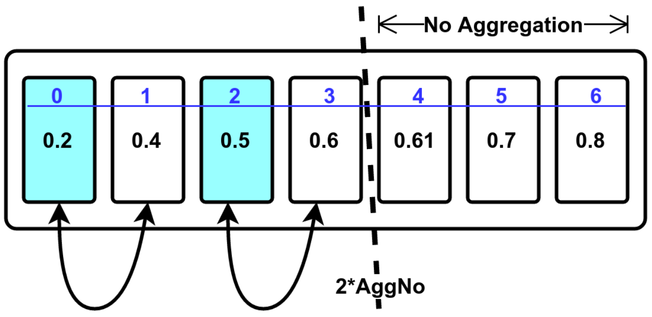

Fig. 4. Multi-agent batch simulation aggregation: For each agent in the population, the *lower_ rank_ count* from the ranking process (show in Figure 3) determines the agents sorted position by rank (shown in blue). Using this position a cut-off can be determined at the point 2*AggNo. Agents which have a sorted rank position below this cut-off will be aggregated and agents above this cut-off will not. For agents which are to aggregated, even numbered (by sort position) agents will act as the newly aggregated crystal by combining the internal co-ordinates (in this case length) of the immediately preceding (odd numbered) agent. The odd numbered agent will be flagged for removal from the simulation.

e.g. meters instead of nanometers. The evolution in time is still the same. So, the Eq. 7 is simplified to:

$$N(L) = L^2 exp(-L^3) \tag{8}$$

, where L is sampled from a of random number distribution between [0.0,2.0]. This range is chosen as below 0 is forbidden, and above 2.0 the probability to be accepted is less than 0.1%, so it can statistically be ignored. The process will be repeated until we initialize the internal coordinates of each crystal.

The value of $AggNo$ is specified at the beginning of the simulation. The parameters from all equations used for initialisation and simulation are summarised in Table 1.

Table 1. Initial Values for Case Study 1

Parameter	Value
I_{agg}	0.875
AggNo	1000
G_0	0
Δt	n/a

A GPU simulation program utilising the CUDA language generated from the model specification by applying the FLAMEGPU framework. In order to replicate the results and the plot (Figure 3) from Gooch and Hounslow paper[3], we generated an initial population of 60,000 agents and ran the simulation for 50 cycles with aggregation number of 1000. After all simulation steps, the final population was about 10,000. To validate the model we compared this histogram results for our simulation with a bin width of 0.1 with that of case study 1 from the Gooch and Hounslow paper. Our results are shown in (Fig. 5) and very closely resemble those reported by the original Monte Carlo approach. To evaluate the performance of the simulation we performed benchmarking by varying the aggregation number and hence the number of cycles to replace the same total number of aggregations. The performance results are shown in Fig. 6 which demonstrate linear speedup with relation to increases in aggregation number.

6 MSMPR Steady State Simulation

In this document, we report performance results for the steady state of MSMPR crystallization. In MSMPR simulation, crystal population is represented the same way as described in Batch simulation (Section 3). Moreover, the three mechanisms of nucleation, aggregation, and growth are simulated for the time interval of Δt. As the steady state, the mechanisms are time-independent. So, the time step Δt can be constant (will be set at the beginning of the simulation).

[3]The final population after aggregation was almost a factor 7

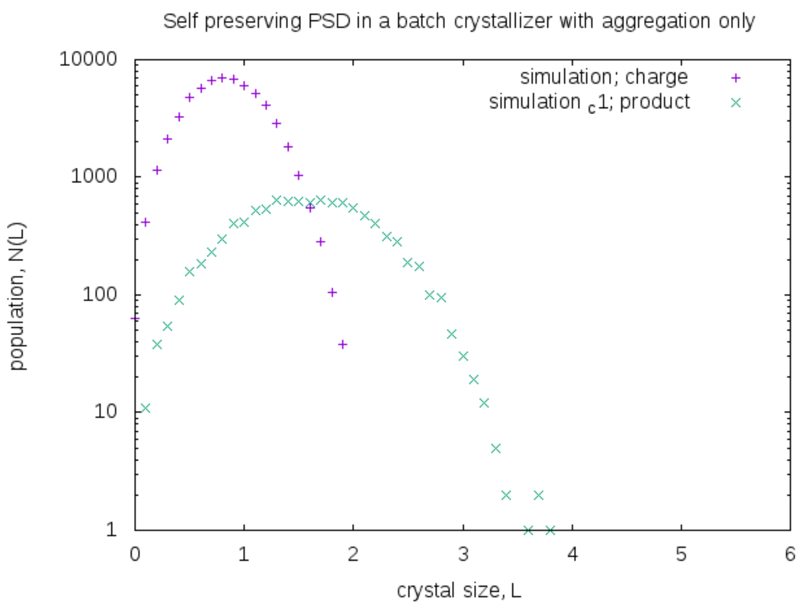

Fig. 5. Batch crystallizer with aggregation only: The figure shows the charge (blue) and product (green) from a multi-agent simulation of the batch crystallisation processes described by case study 1.

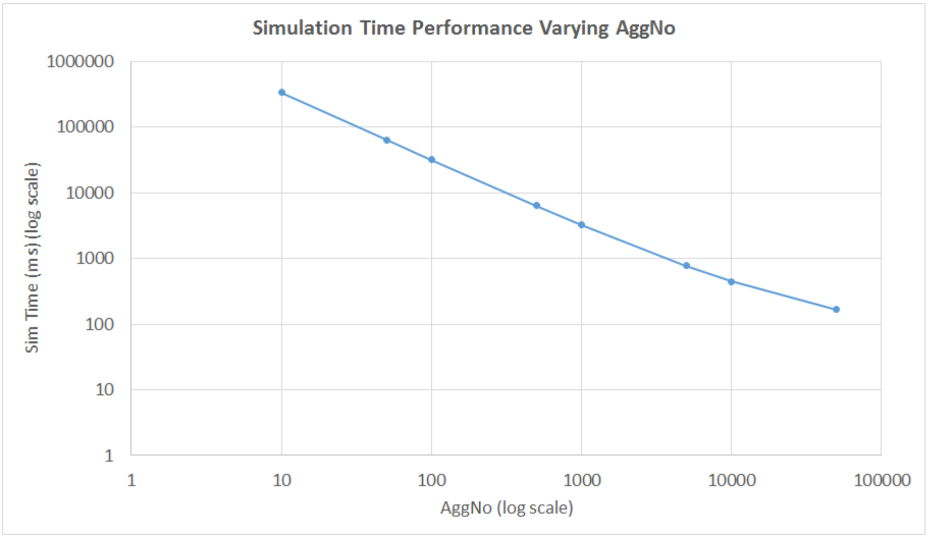

Fig. 6. Batch crystallizer simulation performance: The figure shows the effect of varying the AggNo and simulation steps/cycles to ensure a constant number of total aggregations from an initial population of 60,000. The remaining population size for all cases is 10,000 agents. Having fewer steps with larger a larger number of aggregations increases the parallelism within the model leading to a linear performance improvement.

The aggregation and growth mechanisms are applied in similar way as batch simulation. In addition to these, the nucleation mechanism and output stream are also simulated. To simulate an output stream, we simply remove *ExitNo* number of rows from the array as calculated as below [1]:

$$ExitNo = \frac{PopnNo\Delta t}{\tau} \tag{9}$$

, where τ is the mean residence time, and *PopnNo* is a constant value. The aggregation number is calculated according to

$$AggNo = \frac{PopnNo\Delta m_0}{m_0} \tag{10}$$

To simulate the nucleation, a new row is added to the 2d array. The number of rows to be added is calculated from [1]:

$$NuclNo = \frac{B_0 \Delta t PopnNo}{m_0} \tag{11}$$

, where B_0, Δt, τ, and N_c are chosen in a way that leaves the effect of the three mechanism to be neutral (i.e the result of these three shall leave the number of rows in the 2d array unchanged). The overall simulation process is shown in Fig. 7. The array is initialized with random charge as when the simulation reaches the steady state, all the crystals will be washed out.

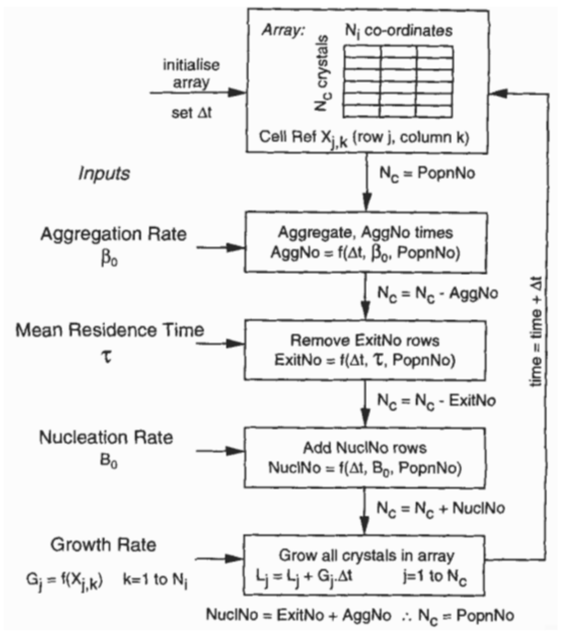

Fig. 7. Monte Carlo MSMPR Steady State Simulation outline, from [1]: The MSMPR simulation processes sequentially applies a number of functions to perform aggregation, exit (output stream), nucleation and growth on the crystal structure array. The time-step is fixed and the total population size remains constant throughout simulation (i.e. steady state).

7 MSMPR Steady State Simulation with Multi-Agents

Using multi-agent based simulator to formulate the Monte Carlo simulation for MSMPR crystallization is done in almost the same way to simulating batch crystallization. The overall simulation processes is summarised by Fig. 8 which shows the agent state diagram of how the model is processed within an multi-agent simulator, FLAMEGPU. We have already explained the graphical notation of flow chart in Section 3.

Similar to batch simulation, we defined a FLAMEGPU exit function *exitFunction* which counts the number of agent crystals in a particular bin. The agent bin is determined by the *simulate* function with the bin identifier stored as an agent variable. As the Population number is constant throughout the simulation (as it steady state) then the *AggNo* and *ExitNo* can be calculated through equations 10 and 9 respectively during initialisation (within the *initConstants* function). The *NuclNo* is the sum of these two values.

The process of aggregation is simulated in the same way as the batch simulation case by assigning a random rank to all agents. Rather than odd number agents with a lower enough rank to aggregate leaving the simulation, these are instead nucleated within the *nucleate* function which extends the batch simulations *simulate* function. Of the agents which are not aggregated, *ExitNo* subsequent agents are nucleated by considering those with the next lowest rank. Importantly this means that agents which are aggregated may not be nucleated in the same simulations step. This assumption could be removed by adding a secondary ranking and sorting stage.

8 Case Study 2 (nucleation and size dependent growth) Results

In this section, we examine a MSMPR case study where only the mechanisms of nucleation and growth are simulated with size (length L) as the only internal coordinate. I.e. Aggregation is not considered. As the growth rate is size-dependant, there is no need to add it as another internal coordinate. For this case study, we initialized the agent crystals' lengths using the same equation we used for the previous case study (Eq. 7). The reason behind this is that no matter what distribution we use to choose the crystals' lengths from, we will get the final distribution result similar to Gooch and Hounslow's paper, Figure 4.

To simulate the growth, the new size of the crystal is calculated from [1]:

$$L_{new} = L_{old} + \Delta t G_0 (1 + \gamma L_{old})^b \tag{12}$$

, where Δt is a constant value of 100 throughout the simulation and the rest of the parameters being used were taken from Table 2.

As we simulate only the two mechanisms of nucleation and growth, *AggNo* is zero and *ExitNo* and *NuclNo* are equal.

A GPU simulation program utilising the CUDA language generated from the model specification by applying the FLAMEGPU framework. In order to

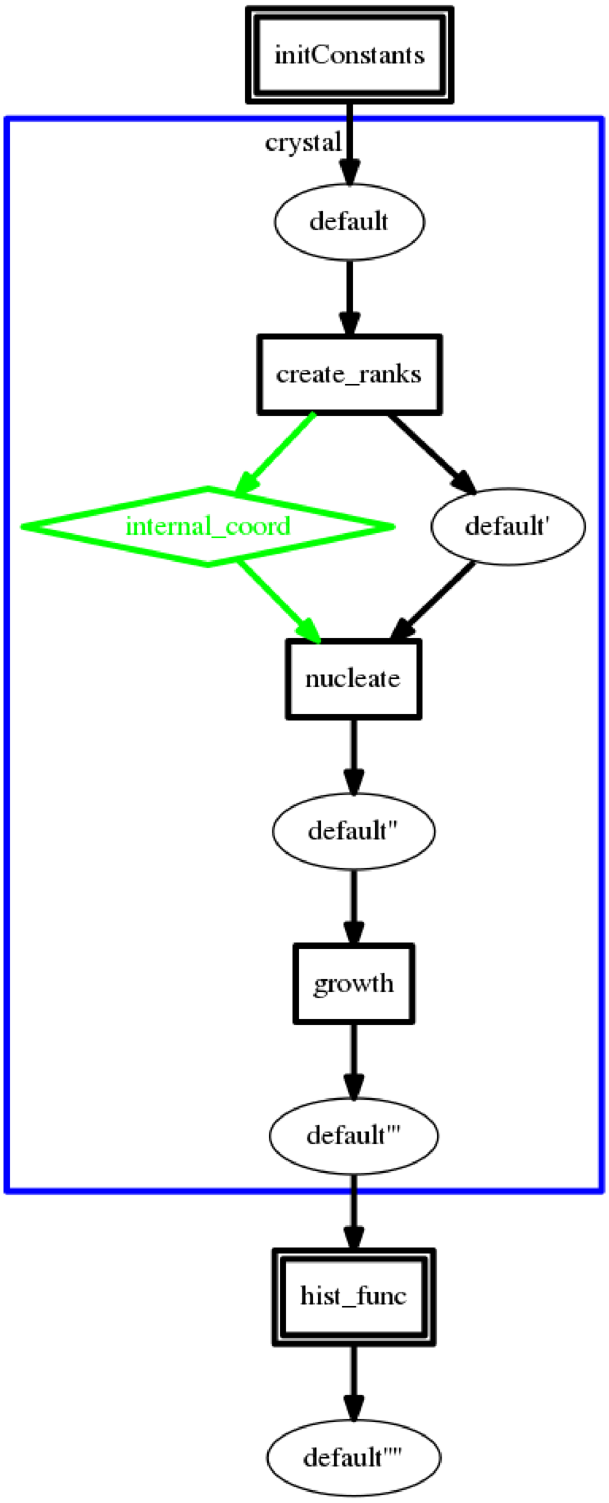

Fig. 8. Multi-Agent MSMPR Simulation State Diagram: Figure shows the state diagram of the FLAME GPU MSMPR simulation model for the general case. Figure differs from that of Figure2 by using a separate agent function to perform growth. The nucleation function also applies aggregation and exit (output stream) to maintain a steady state. A step function is not required as the time-step is constant throughout the simulation steps/cycles.

Table 2. Initial Values for Case Study 2

Parameter	Value
τ	320
B_0	$2\ 10^8$
G_0	0.00168
b	0.22615
γ	1
Δt	100

replicate the results and the plot (Figure 4) from Gooch and Hounslow paper, we generated an initial population of 10,000 agents and ran the simulation for 1,000 cycles. To validate the model we compared this histogram results of our simulation with that of case study 2 from the Gooch and Hounslow paper using bin with of 0.1. Our results are shown in (Fig. 9) and very closely resemble those reported by the original Monte Carlo approach.

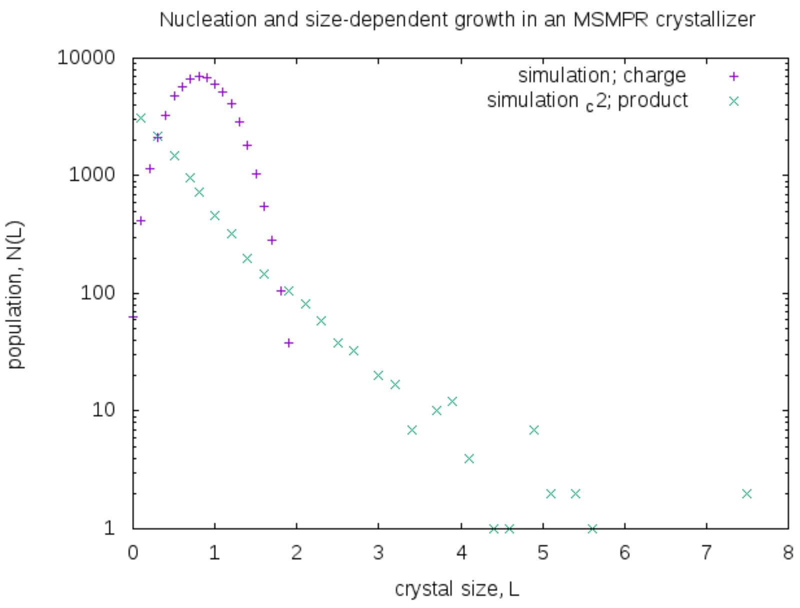

Fig. 9. Nucleation and size dependant growth in MSMPR crystallizer: The figure shows the charge (blue) and product (green) from a multi-agent simulation of the MSMPR crystallisation processes described by case study 2.

9 Software

The FLAME GPU model, code for generating the starting states and plotting scripts for the work described in this document are available from a FLAME GPU fork on GitHub under the *monteCarlo_dev* branch. `https://github.com/mozhgan-kch/FGPU_dev/tree/monteCarlo_dev`

Bibliography

[1] John R. van Peborgh Gooch and Michael J. Hounslow. Monte carlo simulation of size-enlargement mechanisms in crystallization. *AIChE Journal*, 42(7):1864–1874, 1996.

[2] Michael J. Hounslow. A discretized population balance for continuous systems at steady state. *AIChE Journal*, 36(1):106–116, 1990.