Contents lists available at SciVerse ScienceDirect

Chemical Engineering Research and Design

journal homepage: www.elsevier.com/locate/cherd

Simulation of population balance model-based particulate processes via parallel and distributed computing

*Anuj V. Prakash, Anwesha Chaudhury, Dana Barrasso, Rohit Ramachandran**

Department of Chemical and Biochemical Engineering, Rutgers, The State University of New Jersey, Piscataway, NJ 08854, USA

ABSTRACT

Population Balance Models (PBMs), a class of integro partial differential equations, are utilized for simulating dynamics of numerous particulate systems. PBMs describe the time evolutions and distributions of many particulate processes and their efficient and quick simulation are critical for enhanced process control and optimization, especially for real-time applications. However, their intensive computational resource requirement is largely a prohibitive factor in the utility of PBMs for control and optimization. This paper describes how distributed computing systems may be leveraged to execute PBM-based simulations thus achieving time savings, using MATLAB's Distributed Computing Toolbox. A parallel computing algorithm was developed for a three dimensional and four dimensional population balance model with built-in constructs such as SPMD that ran efficiently on a cluster of two quad-core machines linked via a gigabit ethernet cable. Speedup of 6.2 and 5.7 times were achieved with 8 workers, over an unparallelized code, for a 3 and 4 dimensional PBM respectively. Evaluations on work efficiency and scalability further affirm the algorithms' respectable performance on larger clusters despite significant memory transfer overheads.

Keywords: Granulation; Multi-dimensional Population Balance Model; Kernels; Parallel computing; Distributed computing; CPU; Simulation

1. Introduction

In the domain of particulate process design, computer aided modeling and simulation now form an indispensable part of research and development activities within industry and academia. It accelerates process development and design, enables equipment sizing and rating, and finally, facilitates process control and optimization (Chen and Mathias, 2002). This is due in no small part to significant advances made in computer hardware architecture over the last few decades leading to the rise of faster and more efficient CPUs with each new generation. With CPU clock speeds gradually reaching their theoretical limits, processor manufacturers are now resorting to the addition of more cores and incorporation of multi-threading in their CPUs. From a programmer's point of view, this entails rewriting sequential code to distribute data and/or tasks to be evaluated separately on different processing units, a practice known as *parallel* or *concurrent* programming (Wilkinson and Allen, 1999). Commonly termed 'parallelization', it involves the utilization of multiple processing units working together to solve a single problem. If factors such as communication between processors, load balancing and data dependency have been accounted for, parallelization yields considerable reduction in application run time. Owing to the discrete nature of granular materials, the parallel programming paradigm lends itself well to the modeling of particulate processes such as crystallization, granulation, milling and polymerization. These unit operations are fundamental to the manufacture of bulk commercial products (e.g. pharmaceuticals, detergents, fertilizers, and polymers). Research in this domain has been rising steadily over the last few decades (Cundall and Strack, 1979; Matthews et al., 1996; Ramachandran et al., 2009) notwithstanding the fact that these systems are inherently dynamic in behavior and are driven by complex micro-scale phenomena (Muzzio et al., 2002). Granulation, a particle design process, is one such area where substantial progress has been made over the years (Iveson et al., 2001). Approaches for modeling such

* *Corresponding author*. Fax: +1 732 445 2581.
E-mail address: rohit.r@rutgers.edu (R. Ramachandran).

Received 12 November 2012; Received in revised form 21 January 2013; Accepted 30 January 2013
0263-8762/$ – see front matter © 2013 The Institution of Chemical Engineers. Published by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.cherd.2013.01.017

systems are as numerous as they are varied: Discrete Element Modeling (DEM) (Gantt and Gatzke, 2006), population balance modeling (PBM) (Immanuel and Doyle III, 2005; Poon et al., 2008, 2009; Ramachandran et al., 2009, 2012; Dosta et al., 2010; Ramachandran and Barton, 2010; Ramachandran and Chaudhury, 2012), hybrid models by combining PBM with DEM (Gantt et al., 2006), PBM with Volume of Fluid (VoF) methods (Stepanek et al., 2009), PBM with computational fluid dynamics (CFD) (Rajniak et al., 2009), to name a few. Population Balance (PB) models are more suited to simulate large numbers of particles over extensive time periods due to its semi-mechanistic approach (compared to more mechanistic approaches such as DEM and VoF), which are particle-scale modeling frameworks (Freireich et al., 2011).

However, simulation of PBMs are still limited by the computational expenses it incurs in terms of run time and hardware resources. Depending on the complexity of physical phenomena modeled into the system, simulations can take anywhere from a few hours to days to reach completion. This is exacerbated by the fact that computational load increases almost polynomially on increasing dimensionality of the system, leading to even longer run times. By parallelizing a PBM simulation, a programmer is able to circumvent the restriction of running code sequentially on one core, and harness the power of multiple processors within the same machine (parallel) or a cluster of machines (distributed), saving time. Since MATLAB is a widely used high-level language for scientific computing, this article will focus on describing techniques based on toolboxes (function libraries) supported by it. For parallel programming there exists two toolboxes: the Parallel Computing Toolbox (PCT) and MATLAB Distributed Computing Server (MDCS). Specifics on each toolbox can be found in the next section. There has been some recent work on the parallelization of PBM simulations prior to this study. Gunawan et al. (2008) formulated an efficient way of parallelizing High Resolution Finite Volume (HRFV) solved PBEs by assigning the operations on particles in the first half of the size range that were more computationally intensive to processors of greater rank, and operations on the other size range half of decreasing load intensity to higher ranked processors. Their strategy enabled efficient load distribution and resulted in a near linear speedup. More recently, Ganesan and Tobiska (2011) built upon this work by developing a finite element approach of splitting the PBE dimensionally into spatial and internal coordinates, permitting the problem to be parallelized easily without the need for load balancing. This paper will present techniques to enable distributed execution on clusters with the aid of the PCT and MDCS toolboxes following a distributed shared memory approach, without the need for load balancing or explicit inter-processor communication. To illustrate the applicability of this technique, 3 and 4 dimensional PBMs of granulation are developed and parallelized to run across 8 cores (4 cores on 2 machines) utilizing the MATLAB Distributed Computing Server package.

2. Background

2.1. Population Balance Models

A general form of the population balance equation (1) highlighting temporal variation of the distribution of one or more intrinsic properties is given below (Salman et al., 2007):

$$\frac{\partial F}{\partial t}(\mathbf{x},t)+\frac{\partial}{\partial \mathbf{x}}\left[F(\mathbf{x},t)\frac{d\mathbf{x}}{dt}\right]=\Re_{formation}(\mathbf{x},t)-\Re_{depletion}(\mathbf{x},t) \tag{1}$$

where F is the particle number distribution and $\mathbf{x}$ is the vector of internal coordinates used to define the process. $\Re_{formation}$ and $\Re_{depletion}$ represent the net formation and depletion rates of particles occurring from all discrete granulation mechanisms such as aggregation, nucleation and breakage. Typically, PBMs have been described by a single intrinsic property such as particle size.

Dependence on solely particle size was soon found to be inadequate in characterizing variability inherent to the granulation process and other internal properties were required to be considered in PBMs (Annapragada and Neilly, 1996; Iveson, 2002). Therefore, in addition to granule size, binder content and granule porosity are typically selected as decisive factors in optimizing and controlling the process, as evidenced in current research on granulation (Ramachandran and Barton, 2010). This particular work first describes the development of a three-dimensional (3D) PBM framework incorporating all three aforementioned factors as separate dimensions. Verkoeijen et al. (2002) had previously described an efficient way of implementing such a framework by expressing the intrinsic properties of granules – i.e. the volume of solids s, volume of liquid l, and volume of gas g – as a vector in volume space with three coordinates, i.e. s, l and g. Particle internal coordinates (Eq. (2)) are now represented as:

$$\mathbf{x}=[s \quad l \quad g] \tag{2}$$

where each of these three co-ordinates (s, l, g) comprises unique distributions of phase volume fractions (solid, liquid or gas) of particles belonging to a pre-defined volume class, and can therefore be represented as three separate discretized domains or ‘grids’, containing the distributions. These grids are composed of ‘bins’ that represent the volume classes for each phase that constitutes a particle in the population. For instance, the first bin in the solid volume grid represents the particles that have the least solid volume, the next bin contains a fraction of particles with higher solid content and so on. This way, the individual solid volumes of the particles can be represented by allocating them in the corresponding bins. The same principle applies to the other two phase fraction grids, liquid (l) and gas(g). Henceforth, the term ‘grid size’ will be used to refer to the total number of bins in a grid. This discretized approach has two important benefits: (a) it enables decoupling of individual mesoscopic processes like aggregation, consolidation and layering; (b) it improves the numerical solution of the aggregation model due to the mutually exclusive nature of the internal coordinates (Immanuel and Doyle III, 2005). Such a 3D model can now describe changes in the volume distribution of particle volume with respect to time (Ramachandran et al., 2009), as shown below:

$$\frac{\partial}{\partial t}F(s,l,g,t)+\frac{\partial}{\partial g}\left[F(s,l,g,t)\frac{dg}{dt}\right]+\frac{\partial}{\partial s}\left[F(s,l,g,t)\frac{ds}{dt}\right]$$
$$+\frac{\partial}{\partial l}\left[F(s,l,g,t)\frac{dl}{dt}\right]=\Re_{agg}+\Re_{break}+\Re_{nuc} \tag{3}$$

where $F(s, l, g, t)$ represents the population density function such that $F(s, l, g, t)\,ds\,dl\,dg$ is the moles of granules with solid volume between s and $s+ds$, liquid volume between l and $l+dl$ and gas volume between g and $g+dg$. The partial derivative term with respect to s accounts for the layering of fines onto the granule surfaces; the term with respect to l accounts for the drying of the binder and the re-wetting of granules; the term with respect to g accounts for consolidation which, due to compaction of the granules, results in a continuous decrease in pore volume and an increase in pore saturation. $\Re_{agg}$, $\Re_{break}$ and $\Re_{nuc}$ represent the net rates of aggregation, breakage and nucleation respectively.

The $\Re_{aggregation}$ terms (Eqs. (4)–(6)) takes into account the formation/depletion of granules due to aggregation, for which the terms have been defined in literature (Ramkrishna, 2000) as:

$$\Re_{agg}(s,l,g,t) = \Re_{agg}^{formation} - \Re_{agg}^{depletion}, \tag{4}$$

$$\Re_{agg}^{formation} = \frac{1}{2}\int_{s_{nuc}}^{s-s_{nuc}}\int_0^{l_{max}}\int_0^{g_{max}} \beta(s', s-s', l', l-l', g', g-g') \times F(s', l', g', t)F(s', s-s', l', l-l', g', g-g', t)\,ds'\,dl'\,dg' \tag{5}$$

$$\Re_{agg}^{depletion} = F(s,l,g,t)\int_{s_{nuc}}^{s-s_{nuc}}\int_0^{l_{max}}\int_0^{g_{max}} \times \beta(s', s-s', l', l-l', g', g-g') \times F(s', l', g', t)\,ds'\,dl'\,dg' \tag{6}$$

where s_{nuc} is the solid volume of nuclei, $\beta(s', s-s', l', l-l', g', g-g')$ is the size-dependent aggregation kernel that signifies the rate constant for aggregation of two granules of internal coordinates (s', l', g') and $(s-s', l-l', g-g')$. Although many aggregation kernels can be found in the literature (Adetayo and Ennis, 1997; Hounslow, 1998; Sastry, 1975), we have implemented in our model, the empirical kernel proposed by Madec et al. (2003):

$$\beta = \beta_0(L_1^3 - L_2^3)\left((c_1+c_2)^\alpha\left(100 - \frac{c_1+c_2}{2}\right)^\delta\right)^\alpha, \tag{7}$$

where

$$c_i = \frac{volume\ \ of\ \ liquid}{volume\ \ of\ \ agglomerate} \times 100 \tag{8}$$

and β_0, α and δ are empirical constants. The $\Re_{breakage}$ term in the RHS comprises a breakage kernel and a breakage function. The phenomenon itself involves the disintegration of a larger particle into two or more smaller fragments and is mainly governed by attrition and impact. Considering this, the breakage term can then be divided into its corresponding birth and death terms as:

$$\Re_{break}(s,l,g) = \Re_{break}^{form} - \Re_{break}^{dep}, \tag{9}$$

such that the birth and death terms can be explained using Eqs. (10) and (11)

$$\Re_{break}^{form} = \int_0^{s_{max}}\int_0^{l_{max}}\int_0^{g_{max}} \times K_{break}(s,l,g)b(s', s-s', l', l-l', g', g-g') \times F(s', l', g', t)\,ds'\,dl'\,dg' \tag{10}$$

$$\Re_{break}^{dep} = K_{break}(s,l,g)F(s,l,g,t). \tag{11}$$

The breakage function, $b(s', s-s', l', l-l', g', g-g')$ in Eq. (10) has been obtained from literature (Pinto et al., 2007). Since one particle breaks down to form two smaller particles, there is an overall increase in the number of particles. Kernels for breakage can readily be found in the literature (Bilgili and Scarlett, 2005; Ramachandran et al., 2009). In this work we have used the kernel proposed by Pandya and Spielman (1983) to address breakage, defined as

$$K_{break}(s,l,g) = \frac{P_1 G_{shear}(D(s,l,g))^{P_2}}{2}, \tag{12}$$

where G is the shear rate, $D(s, l, g)$ is the particle diameter, and P_1 and P_2 are adjustable parameters (Pandya and Spielman, 1983). The values for G_{shear}, P_1 and P_2 are included in Table 4.

2.2. A 4 dimensional model for multi-component granulation

In a pharmaceutical granulation process, the active pharmaceutical ingredient (API) is typically granulated with one or more excipients (Lee et al., 2008). Modeling such a multi-component granulation process would require the introduction of a second solid component. Several attempts have been made in recent years to understand and subsequently model such a process (Lee et al., 2008; Marshall et al., 2011; Marshall Jr. et al., 2012). With addition of more components, inhomogeneity in granule composition becomes a serious issue as it can reduce the uniformity in the final dosage form (Iveson, 2002; Lee et al., 2008). For this reason, a model that assumes homogeneous composition is inadequate. A fourth dimension must be added to the 3D population balance model to completely account for granule composition (Iveson, 2002). For each additional solid component used, a new dimension must be added to the population balance model. This inevitably increases computational burden during simulation leading to even longer run times, which further stresses the need for parallelization. For the final case study, a four dimensional model has been parallelized to run across eight cores on two machines. The 4D population balance was developed by modifying Eq. (3):

$$\frac{\partial}{\partial t}F(s_1,s_2,l,g,t) + \frac{\partial}{\partial g}\left[F(s_1,s_2,l,g,t)\frac{dg}{dt}\right] + \frac{\partial}{\partial s}\left[F(s_1,s_2,l,g,t)\frac{ds}{dt}\right] + \frac{\partial}{\partial l}\left[F(s_1,s_2,l,g,t)\frac{dl}{dt}\right] = \Re_{aggregation} + \Re_{breakage} + \Re_{nucleation} \tag{13}$$

where s_1 and s_2 represent the volumes of the two solid components in each granule. Same kernels were utilized as per the 3D PBM formulation. In order to reduce computational burden and achieve reasonable simulation times a non-linear grid was used to define the bins in a grid. To cover the granule size ranges observed in an industrial setting, a linear grid would have required a very large number of bins. To uniformly distribute particles into bins in a non-linear grid, the cell average technique (Kumar et al., 2006) was implemented to accurately reassign newborn particles after aggregation and breakage to the adjacent bins. This method has now been extended to 3D (Chaudhury et al., 2013) and 4D (Barrasso and Ramachandran, 2012) PBMs. For the numerical method used to obtain a solution for these models, refer to Appendix A.

3. Model development and parallel programming

Developing a parallel algorithm begins with building a serial code for simulating the granulation process. Both 3D and 4D population balance models were developed from Eqs. (3) and (13) respectively with a few simplifications:

- Of the *source terms*, only aggregation and breakage are considered, eliminating nucleation ($\Re_{nuc} = 0$).
- Of the *growth terms*, layering is neglected, keeping liquid rewetting and consolidation.
- For aggregation, an empirical kernel proposed by Madec et al. (2003) is used

yielding the following PBE (Eq. (14)) for the 3D case:

$$\frac{\partial}{\partial t}F(s,l,g,t) + \frac{\partial}{\partial g}\left[F(s,l,g,t)\frac{dg}{dt}\right] + \frac{\partial}{\partial l}\left[F(s,l,g,t)\frac{dl}{dt}\right] = \Re_{aggregation} + \Re_{breakage} \tag{14}$$

and for the 4D case:

$$\frac{\partial}{\partial t}F(s_1,s_2,l,g,t) + \frac{\partial}{\partial g}\left[F(s_1,s_2,l,g,t)\frac{dg}{dt}\right] + \frac{\partial}{\partial l}\left[F(s_1,s_2,l,g,t)\frac{dl}{dt}\right] = \Re_{aggregation} + \Re_{breakage} \tag{15}$$

Once the serial version of the code has been debugged, partial *vectorization* of the code is carried out to ensure most calculations are performed as efficiently as possible. Vectorization is defined in Flynn's architecture taxonomy (Duncan, 1990) to mean a single instruction stream capable of operating on multiple data elements in parallel. In vectorized code, an operation is performed on all (or multiple) elements of the input variables in one statement, i.e. the operands are treated as single vectors. On the other hand in a non-vectorized code, operations are performed element-wise by treating each operand as a scalar, using loops to index each element of the array. After building a sequential version, the code is parallelized for execution on multiple workers. The procedure for parallel programming involves three basic steps: locating portions of the code that are most time-consuming with tools like the MATLAB profiler; applying one of the approaches for parallelism outlined previously (task, data, or message passing models) as appropriate; and finally optimizing the script for minimal variable transfer overhead. A flowchart representation of this strategy is given in Fig. 1.

The first step is to identify routines that take the most time to run and prioritize to obtain most speedup. A utility like the MATLAB profiler is indispensable in such situations, allowing the programmer to profile his code and locate with precision those statements that are called the most and time needed for their execution. Based on the profiler results, one can then carefully choose which portions of the code to be sped up to yield the best performance without sacrificing scalability. There are some calls which cannot be avoided though, such as those invoked to start a pool of workers and synchronize them at intervals, overheads from built-in functions, etc. These cannot be parallelized or circumvented and are usually not significant. As shown in the next section, the function computing for aggregation proved to be the most resource-intensive,

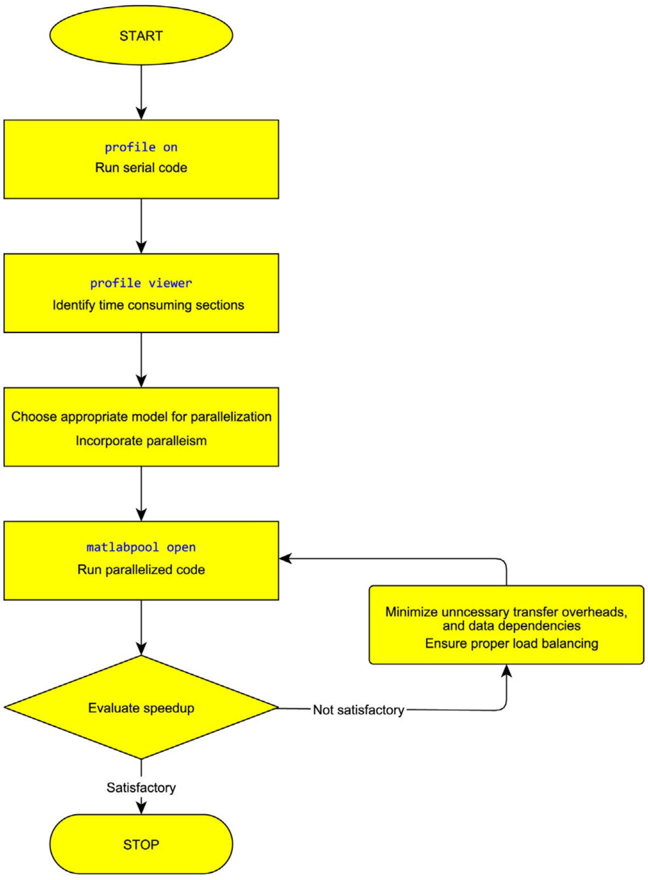
START
profile on
Run serial code
profile viewer
Identify time consuming sections
Choose appropriate model for parallelization
Incorporate paralleism
matlabpool open
Run parallelized code
Minimize unncessary transfer overheads, and data dependencies
Ensure proper load balancing
Evaluate speedup
Not satisfactory
Satisfactory
STOP

Fig. 1 – Flowchart depicting key steps in parallel programming.

followed by the calculations to relocate particle phase fractions into adjacent bins with the cell average method in both 3D and 4D PBMs. Aggregation and associated cell average calculations are therefore the main computational bottlenecks in the PBM code, rendering them prime candidates for parallelization. Their computational burden can be attributed to the presence of several 'nested `for`-loops', prominently those accounting for integral equations (5) and (6), which are performed element-wise and sequentially on a single CPU core. Consequently, broadening the range of each loop index causes individual iterations to run even slower. Increasing the number of bins in each grid while raising the dimensionality of the system also slows down the code execution considerably. This is termed the *curse of dimensionality phenomenon*. Although it is preferred to use a higher grid size for a more accurate representation of the system, the aforementioned limitations curb the degree of flexibility available to a researcher trying to simulate a system. There is therefore much potential for speed-up in parallelizing these loops to run simultaneously on all cores/processors present on the cluster.

Once potential sections have been identified for parallelization, the next step is to decide on a parallel programming paradigm most suited to the application. In our case, following the SPMD (a type of data-parallel) approach seemed most appropriate due to the embarrassingly parallel nature of the computations for aggregation and breakage. Moreover, this would scale well to a distributed system with shared memory (DSM) lending further credence to this mode of parallel programming. The aggregation/breakage kernel (which assuming either a three or four dimensional form – depending on the single or multi-component granulation) typically comprises of 6 nested `for`-loops, with two sets of three loops each for the 3D case (or 8 loops, 2 sets of 4 each for the 4D case), to account for interactions between the *s*, *l*, and *g* fractions of two colliding particles in a bin. Since each MATLAB worker is designed to operate independently of each other with all communication handled by the client instance, the best approach is to decompose the index space adequately by a process known as *loop-slicing* (Klockner et al., 2011). The process involves identifying loop axes (a range of loop index values) capable of functioning as indices for parallelism, followed by assigning these loop axes to available MATLAB workers, `numlabs` (preferably equal to the number of cores on the cluster) by means of `labindex`. `numlabs` returns the number of workers open in a given `matlabpool` session, while `labindex` returns the currently executing worker's index. Loop orders may be switched for efficient memory access patterns and axes may be further sliced if the global memory is found to be insufficient for a given loop size. Following the procedure just described, execution of the aggregation kernel (4D) can be parallelized by slicing the outermost loop:

$$\begin{aligned}\mathfrak{R}_{agg}^{formation} = & \int_0^{upper_1}\int_0^{l_{max}}\int_0^{g_{max}} form(s,l,g) \\ & + \int_{upper_1+1}^{upper_2}\int_0^{l_{max}}\int_0^{g_{max}} form(s,l,g) \\ & + \int_{upper_2+1}^{upper_3}\int_0^{l_{max}}\int_0^{g_{max}} form(s,l,g)\cdots \\ & + \int_{upper_n+1}^{s_{max}}\int_0^{l_{max}}\int_0^{g_{max}} form(s,l,g) \qquad (16)\end{aligned}$$

where

$$upper_n = \frac{s - s_{nuc}}{numlabs} \times labindex \qquad (17)$$

and

$$\begin{aligned}form(s,l,g) = & \frac{1}{2} \times \beta(s', s-s', l', l-l', g', g-g')F(s', l', g', t) \\ & \times F(s', s-s', l', l-l', g', g-g', t) \qquad (18)\end{aligned}$$

In MATLAB code, this slicing would translate as:

$$for \quad i = \frac{(labindex - 1) \times (grid\ \ size)}{numlabs} + 1 : \frac{labindex \times (grid\ \ size)}{numlabs} \qquad (19)$$

The parallel algorithm implemented herein is shown in Fig. 2.

At the start of the simulation, only the MATLAB client instance is actively processing code sequentially. On reaching an `SPMD` keyword, the code then forks off function calls onto idle workers concurrently. With every worker active, execution of the allocated serial tasks now begins asynchronously. After all the workers have completed their respective tasks, the results are summed over all workers (with `gplus`) and the sum cast to one of them. Any result on a worker is of the `Composite` type, but can readily be cast back to regular CPU `single` type on the client MATLAB instance and subsequently re-joined. The final step involves evaluating speedup obtained and fine-tuning the code for optimized performance if necessary. There is no standard protocol to follow while re-structuring the code

```
matlabpool open distributed_config nlabs
% define inputs and perform initial calculations
....
while time  < final_time do
        % send data to workers
        spmd (nlabs)
                % perform calculation on workers
                for   $\frac{(labindex\ -1)*grid\_size}{numlabs} + 1 : \frac{(labindex\ )*grid\_size}{numlabs}$
                        % calculate  $\mathfrak{R}_{agg}^{form}$  and  $\mathfrak{R}_{agg}^{dep}$
                        % calculate  $\mathfrak{R}_{break}^{form}$  and  $\mathfrak{R}_{break}^{dep}$
                end
                output = gplus(output,1) % global summation across workers
        end
        %gather results and send back to client
        %calculate output variables and update F(s,l,g)
end
```

Fig. 2 – Algorithm for distributed execution describing the loop-slicing technique in conjunction with the `SPMD` keyword.

to achieve optimal speedup. As a rule of thumb, the programmer must strive to: minimize data transfer overheads; reduce data inter-dependencies; and finally, balance computational loads across all cores. This final step is critical to ensure robust performance and scalability. To sum up, the procedure followed herein for parallelizing PBMs for distributed systems involved three steps: locating portions of the code that are most time-consuming with tools like MATLAB profiler; applying one of the aforementioned approaches for parallelism as appropriate; and lastly minimizing overheads associated with variable transfer, data dependencies and load balancing.

4. Results and discussion

4.1. *Three-dimensional PBM*

The serial version of the 3D population balance code was developed as described in the previous section. After issuing the `profile on` command to initiate the MATLAB profiling utility the script was executed on one worker. The profiler results are displayed in Fig. 3. As can be observed from the figure, the primary computationally intensive parts are those forming the core aggregation kernel: Solid, liquid and gas phase fraction relocation into adjoining bins (using the cell-average method), followed by formation then depletion. Breakage and its associated functions took relatively much lesser time to compute mainly due to the lack of integral terms. Results indicated that the formation term was the primary bottleneck in 3D PBM simulation using a linear grid. However, a non-linear grid would require much fewer bins than the linear grid to cover the same granule size range. This justifies the incorporation of a computationally burdensome cell average method for relocating particle phase fractions to appropriate adjacent bins.

Once computational bottlenecks were identified, the next step was to incorporate parallelism into the code. The data parallel approach was deemed ideal for this algorithm owing to the presence of nested `for` loops that perform numerous MAD (Multiply-ADd) operations on all elements of the same data set. More specifically, a divide and conquer strategy was executed on equally partitioned data sets which would likely scale well on distributed systems with shared memory. The loop slicing technique described in the previous section was utilized to achieve data parallelism, effectively assigning the same task to operate on different partitions of the shared array

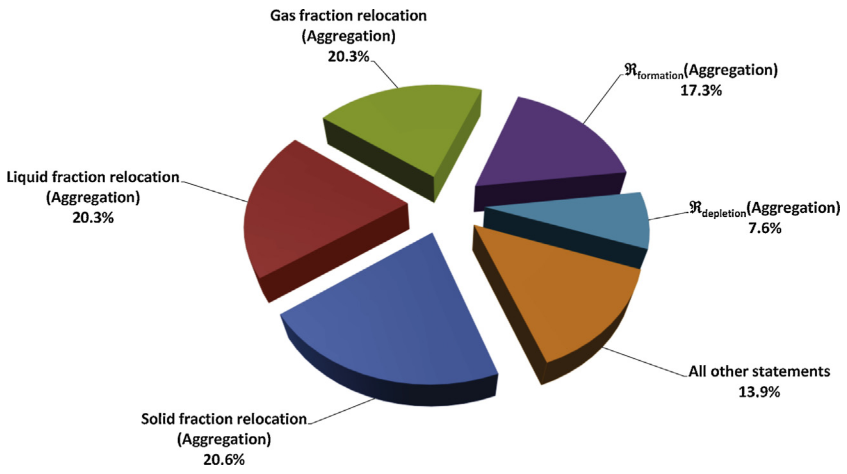

Fig. 3 – Piechart representation of MATLAB's profiler results for a serial version of the 3D granulation population balance code run on a single worker.

concurrently. The aggregation and breakage functions were consolidated into a single function and the outermost `for` loop was sliced in accordance with the number of workers available (refer Eq. (19)), and performance was assessed for five cases: one (sequential), two, four, eight local workers and 8 distributed workers. For the 8 distributed worker case the parallel code was executed in a distributed manner using 4 cores per node. The distributed system consisted of two nodes linked via a high speed gigabit ethernet cable, each node housing a quad-core Core i7-2600 CPU running at 3.4 GHz (stock), and 16GB of local memory (4 DIMMs × 4GB @ 1600 MHz).

Speedup benefits of parallelization are meaningless if the simulation results are not reproducible and reasonably accurate numerically. To verify this, the resulting data from each of the four parallel cases are superimposed to confirm numerical precision of the each parallel simulation case (see Figs. 4–7). Initial input parameters for the simulation are given in Table 3. Fig. 4 shows that the average diameter linearly rises due to the combined effects of liquid binder addition and aggregation, offsetting consolidation and breakage. Both particle number and porosity distribution (Figs. 5 and 6) is limited to the 0–100 μm size class by the end of the short simulation period (t = 20 s), while the increase in total volume (Fig. 7) at each time step is very minimal due to gradual liquid binder addition into the system. It is evident that the results from each parallel version conforms numerically to the results of the sequential version (1 worker) showing that computational accuracy was not compromised during distributed execution.

The most direct metric for measuring parallel performance is the speedup factor, and is generally represented as the ratio

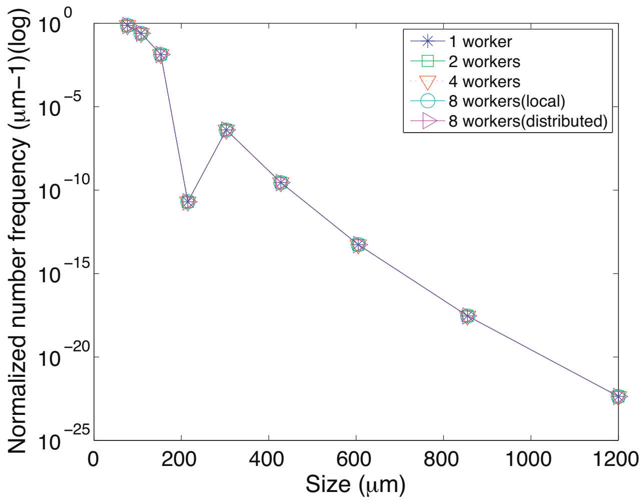

Fig. 5 – Number frequency vs particle size for the 3D population balance model.

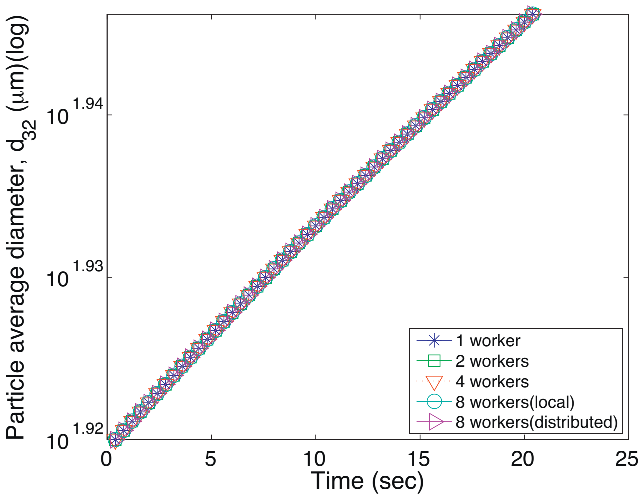

Fig. 4 – Evolution of average diameter of a particle over time for the 3D population balance model.

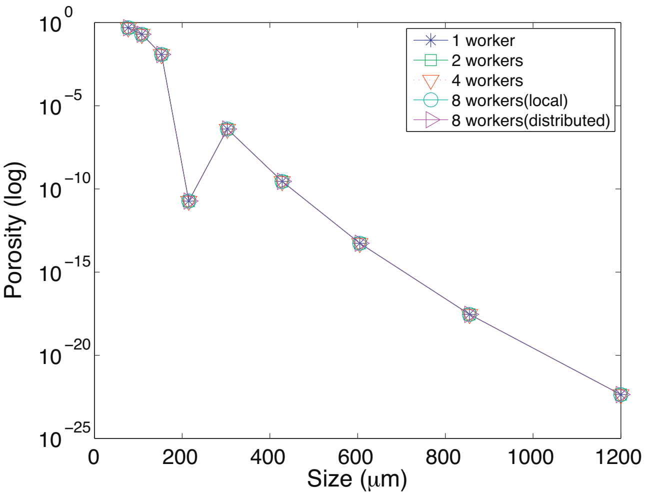

Fig. 6 – Particle porosity vs particle size for the 3D population balance model.

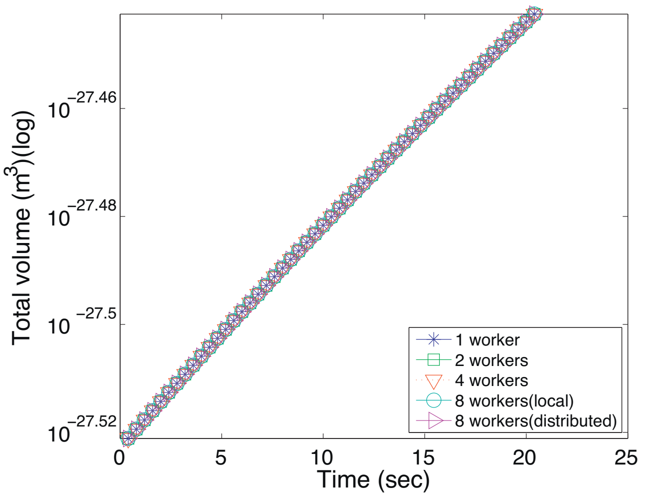

Fig. 7 – Evolution of total volume of particles over time for the 3D population balance model.

of serial execution time to parallel execution time (Eq. (20)). The speedup ratios thus calculated were plotted versus number of workers. In Fig. 8, speedup is shown for a simulation that was run on a single CPU making use of 1, 2, 4, and 8 labs. These 8 labs were initiated on 8 threads present in the quad-core CPU (4 cores × 2 threads each) with the `matlabpool open` command. Another linearly rising plot in same figure highlights the improvement in speedup after using 8 distributed workers running on 2 nodes over 8 labs on a single node. This comparison serves to prove why assigning tasks to cores (equivalent to workers) provides better performance than assigning the same number of tasks to labs (equivalent to threads). This is not unexpected since memory transfer overheads associated with inter-thread communication are much higher than for cores (Warg, 2006). A speedup of 3.79 was achieved with 8 labs on a single CPU (an improvement of 94.7%), which is very close to the theoretical maximum speedup possible, i.e. 4 times (corresponding to 4 cores). On a distributed system, this performance increase is pushed even higher with a peak

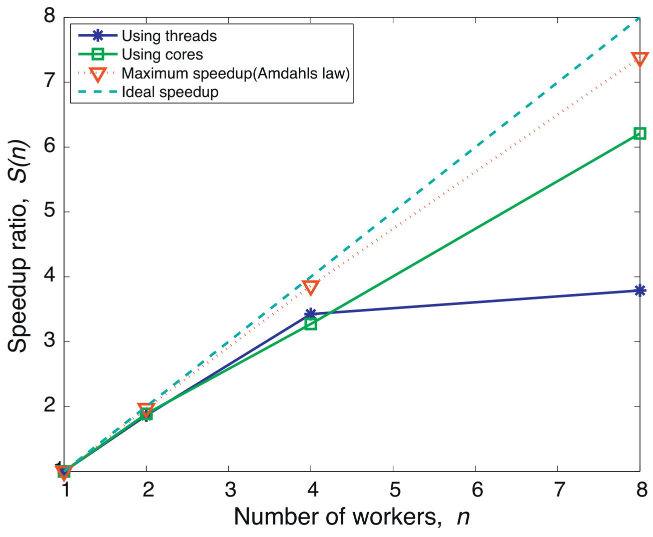

Fig. 8 – Comparison of speedup using 8 threads on a single CPU, 8 cores on 2 nodes, and maximum speedup as predicted by Amdahl's law for a 3D PBM. The dashed line represents the theoretical upper bound for speedup and is equal to the number of available workers.

speedup of 6.21 times on 8 workers (77.61% improvement). The ideal speedup line shown in both figures indicates the theoretical maximum for each worker pool set and is equal to the number of physical cores (workers) available.

$$\textit{speedup},\ S(n) = \frac{\text{Execution time on a single worker}}{\text{Execution time on} n \text{ workers}} \tag{20}$$

It must be kept in mind that there are almost always some statements in a parallel algorithm that cannot be parallelized, and have to be executed serially on one worker. This serial fraction (*f*) limits the maximum speedup attainable with a certain pool of workers (*n*), given by Eq. (21), also known as Amdahl's law (Amdahl, 1967).

$$S(n) = \frac{n}{1 + (n-1)f} \tag{21}$$

The maximum speedup according to Amdahl is calculated for each parallel case and plotted in Fig. 8. From the graph it is clear that the observed speedup factor line is initially in proximity with the maximum speedup line, but gradually diverges as the number of workers increase. This leads us to the issue of scalability, a property of cluster systems exhibiting linearly proportional increase in performance of the parallel algorithm with corresponding increase in system size (i.e. addition of more processors) (Wu and Li, 1998). Several metrics have been proposed to synthetically quantify scalability, and due to the pre-determined nature of the initial problem size, the *fixed serial work per processor* approach based on *efficiency* was adopted, and the resulting speedup is termed *scaled speedup*. It relies on problem size (number of bins in a grid) remaining constant even as the number of processors are increased. If memory overheads associated with distributed execution rise only linearly as a function of the number of workers *n*, the parallel execution time (T_n) keeps on decreasing and algorithm is considered scalable. Efficiency in its simplified form is defined as the fraction of time that workers actually take to perform computations, and is given in Eq. (22).

$$\text{Parallel efficiency},\quad E_n = \frac{T_s}{T_n \times n} = \frac{S(n)}{n} \times 100\%$$

where T_s is the serial execution time. A more descriptive definition of efficiency may be given as (Gupta et al., 1993):

$$E_n = \frac{t_c W}{n T_n} = \frac{t_c W}{t_c W + T_0(n, W)} \tag{22}$$

where E_n is efficiency of the system size n, $T_0(n, W)$ the total overhead of the distributed system, t_c the average execution time per operation in the architecture and is a constant, W is the problem size which translates to processor work, and $t_c W$ is the serial computing time (T_s) of an algorithm. $T_0(n, W)$ is calculated as:

$$T_0 = pT_n - T_s \tag{23}$$

Since t_c is a constant and depends on the underlying architecture itself, it can be calculated from processor specifications. We used a Intel Core i7 'Sandybridge' CPU which maintained a 3.512 GHz during the simulations, i.e. 3.512×10^9 cycles/s. In the Sandybridge architecture, 4 floating point operations (flops) can be computed per clock cycle, which allows us to calculate the maximum theoretical flops that can be processed in a second (υ): 1 core × 4 flops/core/cycle × 3.512×10^9 cycles/s = 14.046 gigaflops/s;

Table 1 – Performance evaluation metrics for the parallel 3D population balance model simulation.

Workers	T_0 (s)	E_n	W_n (flops)
1	–	1	2.149×10^{14} (serial work, W)
2	919.3	0.9433	2.278×10^{14}
4	3405.9	0.8179	2.627×10^{14}
8 (local)	16,992.1	0.47	4.536×10^{14}
8 (distributed)	4409.3	0.776	2.768×10^{14}

therefore the time required to compute 1 flop, t_c is 7.119×10^{-11} s. Using this information we can determine the serial work W from Eq. (22). Parallel work W_n is defined as:

$$W_n = n \times T_n \times \upsilon \tag{24}$$

See Table 1 for the calculated values of efficiency, parallel overhead and processor work. Extracting the expressions for T_s and T_n from Eqs. (22) and (24) respectively and combining them with Eq. (22) yields: $E_n = W/W_n$. It can be inferred that E_n cannot exceed 1 and addition of workers beyond n does not increase S_n if parallel work W_n exceeds the serial work W. From Table 1, it is apparent that parallel work is indeed consistently greater than serial work for increasing worker counts, supporting this observation. Even increasing the number of workers to infinity only results in bringing down efficiency closer to zero (refer Eq. (22)). Therefore, for constant-problem sized scaling, the speedup does not continue to increase with the increasing number of processors, but tends to saturate or peak at a certain system size, a principle apparent from the trajectory of the speedup curve (Fig. 8). Thus the 3D granulation population balance algorithm cannot be perfectly scalable for a fixed-size problem. According to Amdahl, even with an infinite number of processors the maximum speedup for a code is restricted to $1/f$, the inverse of its serial fraction, which yields a value of 83.33 times for our simulation. This implies that utilizing more than 84 workers will not produce any additional improvement in performance. Even this limit is virtually impossible to reach, as Amdahl's equation does not take into account memory transfer overheads, effects due to improper load balancing and synchronization, all of which become more dominant as the number of workers increase. Moreover, to remain scalable, W_n must be a function of the number of processors, n, to ensure that parallel overhead T_0 does not grow faster than n rises (Gupta et al., 1993). This is where another metric, the *overhead ratio*, proves useful for testing the performance of our parallel algorithm with additional workers. This concept is introduced in the next section, followed by an analysis of this ratio for both 3D and 4D granulation codes with a view to compare scalabilities.

Table 2 – Performance evaluation metrics for the parallel 4D population balance model simulation.

Workers	T_0 (s)	E_n	W_n (flops)
1	–	1	7.24×10^{13} (serial work, W)
2	749.9	0.865	8.36×10^{13}
4	1207.1	0.81	8.94×10^{13}
8	2131.05	0.708	1.02×10^{14}

4.2. *Four-dimensional PBM*

The serial version of the 4 dimensional population balance model code was built following the procedure described in Section 3. The initial grid size was set to 8 along each dimension and the process run time to 20 s in order to manage actual execution time within a feasible range sufficient for comparison. After debugging, some sections were vectorized to partially reduce the overall execution time and concentrate burdensome computations on aggregation and breakage. Using the MATLAB profiler utility, a breakdown of the time spent calculating each statement was obtained and the results charted (Fig. 9). Once again, it is clear that aggregation and its associated calls are the main bottlenecks in the code: solid, liquid and gas phase fraction relocation into adjoining bins using cell-average take the most time, followed by formation due to aggregation. Aggregation-induced depletion consumes nearly 10% of the total run time, while breakage and its associated calls took relatively much lesser time to compute. The second solid component ('solid 2') present in the initial distribution adds further computational complexity in our 4D PBM code as it introduces another pair of `for`-loops to cover the entire grid range. In all, there are eight nested `for`-loops accounting for the integral terms in the 4D PBM, making it the candidate of choice for parallelism via loop slicing. The outer loop enclosing the aggregation and breakage functions was sliced in accordance with Eq. (19) to ensure data-parallel execution on 8 workers.

To confirm numerical precision of the parallel simulation results for a 4D PB code, a more accurate set of initial parameters was used (see Table 3) instead of those for profiling and performance analysis. This was done to validate the effectiveness of our parallel model with realistic parameters. In addition, the results were compared for only 2 cases – 1 and 4 workers due to the trial version of the MDCS toolbox expiring. Results from each case were directly superimposed as in the case of the 3D model to check for consistency of numerical precision (see Figs. 10–13). These figures plot the average diameter and composition over time, l as well as the normalized number frequency and average porosity vs diameter at the end of granulation. The average porosity is the total

Table 3 – Process parameters and initial conditions used in 3D PBM simulation.

Parameter name	Value
ρ_{solid}	2700 kg/m^3
ρ_{liquid}	1000 kg/m^3
ρ_{gas}	1.2 kg/m^3
Granulation time	20 s
Time step size	0.4 s
Volume of first bin, solid component, $s_{1,1}$	1×10^{-13} m^3
Volume of first bin, liquid binder, l_1	1×10^{-13} m^3
Volume of first bin, gas, g_1	1×10^{-13} m^3
Total number of bins in each dimension	16
Initial particle count F in bin (1, 1, 1)	1×10^{-15} mol
Aggregation constant, β_0	8×10^{14} mol^{-1}s^{-1}
Aggregation constant, α	1
Aggregation constant, δ	1
Breakage constant, P_1	7×10^{-11} m^{-1}
Breakage constant, P_2	1.3 m^{-1}
Shear rate, G_{shear}	60 s^{-1}
Consolidation rate constant, c	1×10^{-21} s^{-1}
Minimum porosity, ϵ_{min}	0.2
c_{binder}	0.1
Liquid binder spray rate, $\dot{V}_{spray}$	5×10^{-3} m^3/s

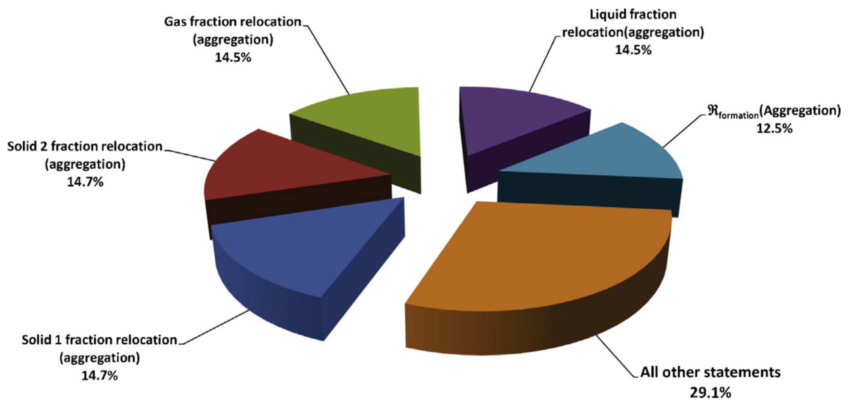

Fig. 9 – Piechart representation of MATLAB's profiler results for a serial version of the 4D granulation population balance code run on a single worker.

pore volume (liquid and gas) within each size class divided by the total volume of particles in that class. By visual inspection it is evident that the data for granule properties in the 4 worker parallel case is completely identical to the data from the serial version (1 worker) confirming that computational accuracy was not compromised during distributed execution.

Fig. 10 shows the evolution of particle diameter over time. It linearly rises due to the combined effects of liquid binder addition and aggregation, offsetting consolidation and breakage phenomena. This brought up the average diameter from under 100 μm to around 500 μm. Distribution of the average composition of solid component 1 (s_1) remains constant (Fig. 11)

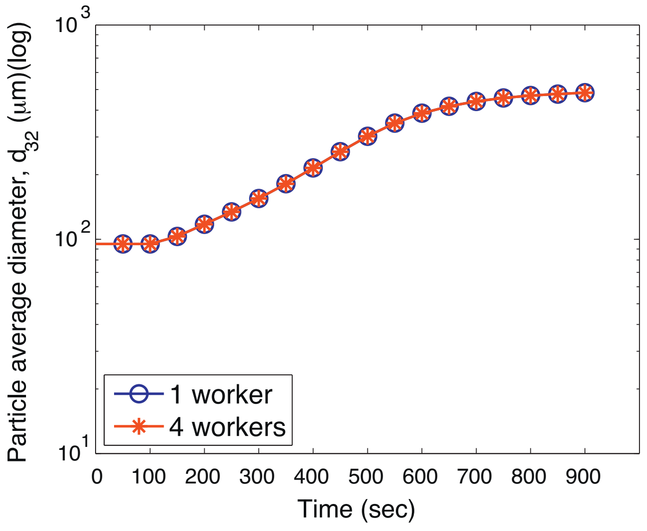

Fig. 10 – Evolution of average diameter of a particle over time for the 4D population balance model.

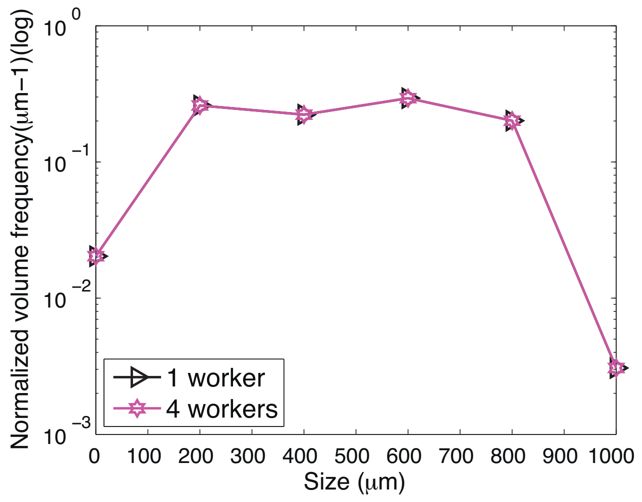

Fig. 12 – Normalized particle size distribution (weighted by volume) for the 4D population balance model.

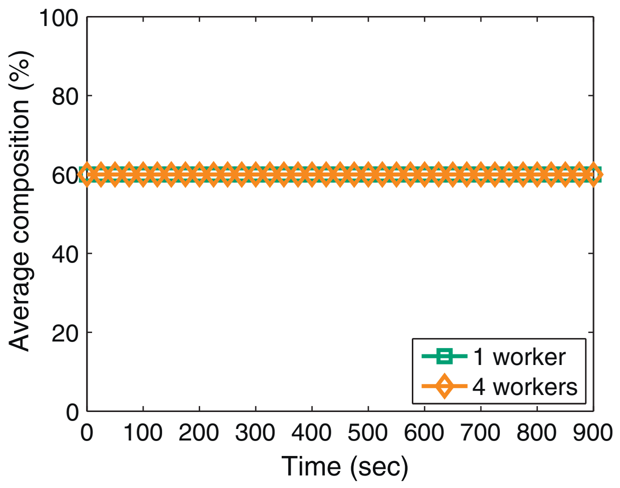

Fig. 11 – Average composition distribution of solid component 1 (s_1) in the mixture after multicomponent granulation for the 4D population balance model.

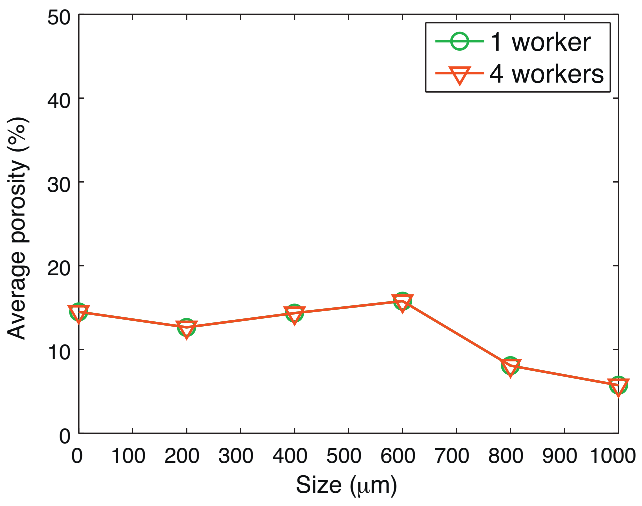

Fig. 13 – Average particle porosity distribution after multicomponent granulation for the 4D population balance model.

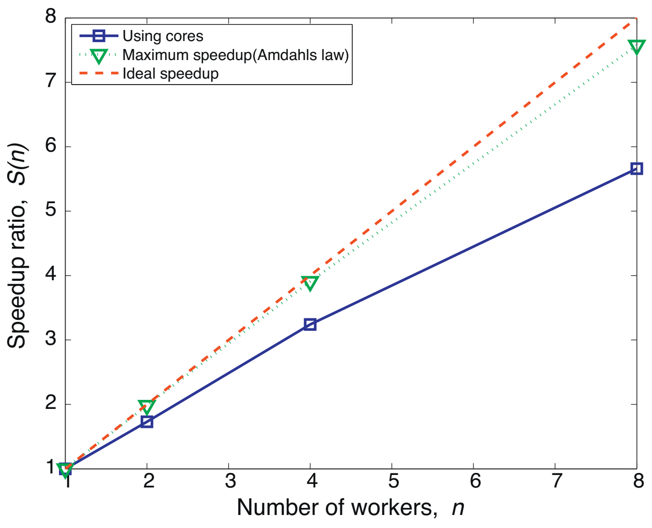

Fig. 14 – Comparison of speedup 8 cores on 2 nodes, and maximum speedup as predicted by Amdahl's law for a 4D PBM. The dashed line represents the theoretical upper bound for speedup and is equal to the number of available workers.

because of the conservation of mass with respect to s_1 in the system. Particle size distribution after granulation, weighted by volume and normalized, is presented in Fig. 12. Average particle porosity is directly affected by the volumes liquid and gas in a particle and the distribution is plotted in Fig. 13.

The speedup ratios were calculated from Eq. (20) and plotted against the number of workers. As shown for the 3D case, running on a distributed system produced better performance as opposed to execution on threads since distributed workers can create local arrays simultaneously, saving transfer time (MathWorks, 2011). Hence, there was no need to perform a speedup comparison between cores and threads for this particular case. Eight workers were initiated on 8 cores divided across two nodes (4 cores per node) with the `matlabpool open` command. Each core on the INTEL Core i7 2600 CPU has a theoretical SSE (Streaming SIMD Extensions) rate of approximately 14 gigaflops per second as computed in the previous section. Fig. 14 shows the speedup for a simulation run on the distributed system making use of 1, 2, 4, and 8 workers. Initial input parameters for the simulation (except for grid size and runtime) are given in Table 4. The linearly rising plot shows a peak speedup of 5.6 times over a single worker, using 8 workers. The maximum permissible speedup as calculated according to Amdahl's law is roughly 7.6 times considering a non-parallelizable, serial fraction of 0.8% of the total execution time. The ideal speedup line shown in both figures indicates the theoretical maximum for each worker pool set and is equal to the number of physical cores (workers) available.

The observed speedup line diverges from the theoretical ideal and Amdahls's speedup much more rapidly than for the 3D granulation simulation case, a trend evident in the speedup plot (Fig. 14). As a result, we expect the scalability of the algorithm to naturally decrease as more nodes housing CPUs of the same specifications are added. To quantify scalability we once again rely on the fixed-problem size scaled approach since the initial 4D grid size will remain the same even when the available pool of workers are increased. The algorithm is considered scalable if the overheads associated with data transfer between the job manager and workers rise only linearly as a function of the number of workers n (Heath, 2011). This will

Table 4 – Process parameters and initial conditions used in 4D PBM simulation.

Parameter name	Value
Granulation time	900 s
Time step size	0.5 s
Volume of first bin, solid component 1, $s_{1,1}$	1×10^{-13} m^3
Volume of first bin, solid component 2, $s_{2,1}$	1×10^{-13} m^3
Volume of first bin, liquid binder, l_1	2×10^{-14} m^3
Volume of first bin, gas, g_1	1×10^{-14} m^3
Total number of bins in each dimension	8
Initial particle count F in bin (1, 2, 1, 2)	3×10^{-13} mol
Initial particle count F in bin (2, 1, 1, 2)	7×10^{-13} mol
Aggregation constant, β_0	1×10^{19} mol^{-1}s^{-1}
Aggregation constant, α	1
Aggregation constant, δ	1
Breakage constant, P_1	1 m^{-1}
Breakage constant, P_2	1
Shear rate, G_{shear}	85 s^{-1}
Consolidation rate constant, c	1×10^{-7} s^{-1}
Minimum porosity, ϵ_{min}	0.1
Liquid binder spray interval	$120\,s < t < 360\,s$
Liquid binder spray rate, $\dot{V}_{spray}$	25 mL/s

cause parallel execution time (T_n) to steadily decrease. While discussing scalability, an aspect to be considered is the scalability of an algorithm with respect to underlying hardware. Algorithms scalable on one architecture need not necessarily scale well on others, and hence deploying the application on heterogeneous cluster systems may prove counter-productive in such situations. Since our distributed system is homogeneous in terms of underlying hardware, it will also scale well on additional workers of the exact same specifications. An important measure of such an algorithm's efficiency is its overhead ratio – a ratio of its communication overhead to parallel execution time (Eq. (25)). The lower the ratio, the more each worker will perform effectively. Typically, this ratio increases swiftly with increasing number of workers but decreases as the problem size grows (Wu and Li, 1998). For the specific case of the 4D granulation simulation, this intrinsic property of a parallel algorithm presents the best means of explaining the accelerated decline in speedup as worker counts increase.

$$\text{Overhead ratio,}\quad \varnothing = \frac{\text{Parallel overhead}}{\text{Parallel execution time}} = \frac{T_0}{T_n} \qquad (25)$$

Table 2 displays the calculated values for efficiency, parallel overhead in seconds and processor work in flops. From the relation $E_n = W/W_n$, it can be inferred that E_n cannot exceed 1 and addition of workers beyond n will not increase speedup if parallel work, W_n, remains greater than the serial work, W. From the table it is evident that parallel work is indeed consistently greater than serial work for increasing worker counts, proving that perfect scalability is unattainable for the 4D PBM. Moreover, W_n must remain a function of the number of processors, n, to ensure that parallel overhead T_0 does not grow faster than n rises (Gupta et al., 1993). To observe this trend, we plot the overhead ratio against the number of workers, as seen in Fig. 16. A relatively linear rise of parallel overhead with the number of processors is observed for 8 workers accompanied by the inevitable decrease in processor efficiency. On comparison with the 3D granulation simulation (Fig. 15), we find the overhead ratio is approximately 3 times lesser than for the 4D code across 8 workers. The most obvious cause for this difference is that the time required for distribution of 4 dimensional

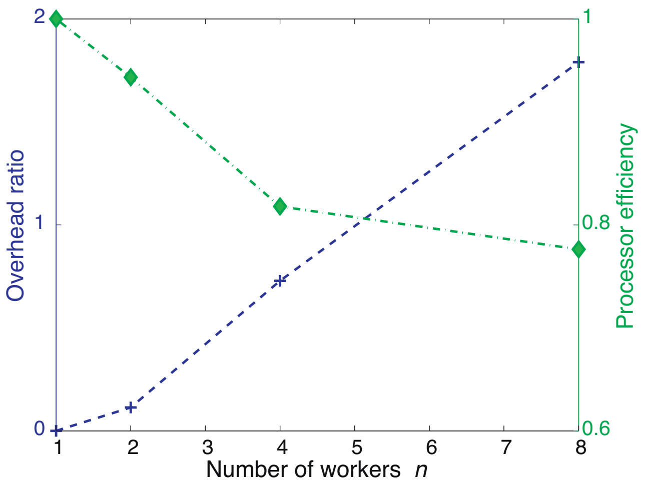

Fig. 15 – Variation of overhead ratio with processor efficiency as the number of workers are increased for the 3D population balance simulation.

arrays across workers are much higher than for 3 dimensional data arrays.

Larger, multi-dimensional arrays have a higher memory requirement during data creation and modification, entailing longer read/write times over the network. Thus, we can classify this phenomenon as a type of memory-constrained parallelism. This additional overhead alone consumes a significant portion of useful processor time especially as the input grid size increases, bringing down processor efficiency drastically. This is apparent from the nearly 10% drop in efficiency from using a 3D to 4D grid for 8 workers (see Figs. 15 and 16). Consequently, for problem size-fixed scaling, the speedup does not increase with the increasing number of processors, but it tends to saturate or peak at a certain system size, showing that the distributed system is not perfectly scalable for a fixed-grid size 4D PBM. Following Amdahl's hypothesis, an infinite number of processors will only yield a maximum speedup of $1/f$, yielding a value of 125 times for the 4D case, implying that utilizing more than 125 workers will not produce any additional improvement in performance. Nevertheless, this value is a gross overestimate as explained in the previous section, and more so for the 4D case as Amdahl's equation does not take into account memory transfer overheads, the performance-deteriorating effects of which have just been demonstrated.

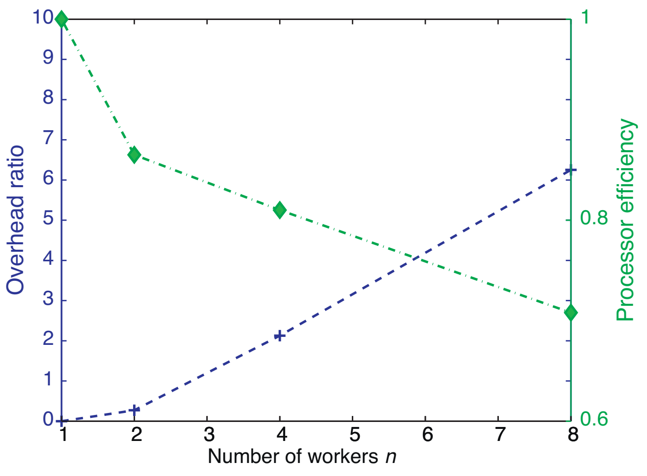

Fig. 16 – Variation of overhead ratio with processor efficiency as the number of workers are increased for the 4D population balance simulation.

5. Conclusions

This study has described the development and implementation of a parallel algorithm describing granulation behavior on a distributed computing system. Two models were built based on the population balance method: a single component granulation process represented by a three dimensional PB model; and a multi-component granulation process represented by a four dimensional model. Following a distributed shared memory approach to achieve data parallelism, the models were parallelized with the loop slicing technique. The MATLAB toolboxes utilized for this purpose were the Parallel Computing Toolbox in conjunction with the Distributed Computing Server. The results show a good speedup of 6.2 times with 8 workers for the 3D PBM code while the 4D code was slightly less efficient with a speedup factor of 5.7 times. It was shown that perfect scalability is theoretically impossible to attain for either case, since fixed-problem size scaling does not allow for 100% per processor efficiency, which in turn is due to parallel work continually exceeding serial work. By Amdahl's law, even with an infinite number of processors, the speedup is limited by the fraction of algorithm that has not been parallelized. With the aid of the overhead ratio we compared scalabilities of the two algorithms on our homogeneous distributed system. While the overhead ratio rose linearly with respect to the number of workers in both cases, the performance is clearly better for the 3D model which also displayed a higher processor efficiency with 8 workers. This is usually a good indication that the 3D algorithm will scale better than the 4D, with a lower execution time. The 3D code will be expected to yield no speedup benefit beyond 83 workers, whereas the speedup for 4D code will be restricted to less than 125 workers. Amdahl's equation does not take into account memory transfer overheads, improper load balancing and synchronization, the effects of which becomes more dominant as the number of workers increase. In all parallel cases, simulation results were confirmed to be equal to the sequential version's showing that numerical precision was not compromised during distributed execution. In future work, we will seek to further reduce transfer overheads algorithmically by memory pre-allocation using distributed data types and also MPI-based constructs like labSend and labReceive to manage inter-processor communication.

Acknowledgements

This work is supported by the National Science Foundation Engineering Research Center on Structured Organic Particulate Systems, through Grant NSF-ECC 0540855.

Appendix A. Numerical solution

Using a multidimensional population balance with an appropriate kernel ensures an improved analysis/prediction of the granulation process. But, besides developing the model, incorporating an efficient numerical technique for solution to such integro-partial differential equation is yet another difficult task. The time scale and multiple dimensions introduce various complexities to the solution technique. Hence it is very crucial to develop robust models with efficient solution techniques for such a framework. Our approach for obtaining a solution to such equations is based on a hierarchical two-tiered algorithm as proposed by Immanuel and Doyle III (2003). This involves using the finite volume approach for discretization with respect to each individual solid, liquid and gas volume, followed by integration of the population balance over the domain of these subpopulations. Neglecting layering, Eq. (3) can be expressed in the discrete form as shown in Eq. (26):

$$\frac{dF'_{i,j,k}}{dt} + \left(\frac{F'_{i,j,k}}{\Delta l_j} \frac{dl}{dt}\bigg|_{l_j} - \frac{F'_{i,j+1,k}}{\Delta l_{j+1}} \frac{dl}{dt}\bigg|_{l_{j+1}} \right) + \left(\frac{F'_{i,j,k}}{\Delta g_k} \frac{dg}{dt}\bigg|_{g_k} - \frac{F'_{i,j,k+1}}{\Delta g_{k+1}} \frac{dg}{dt}\bigg|_{g_{k+1}} \right) = \Re_{nuc}(s_i, l_j, g_k) + \Re_{agg}(s_i, l_j, g_k) + \Re_{break}(s_i, l_j, g_k) \qquad (26)$$

Here $F'_{i,j,k} = \int_{s_i}^{s_{i+1}} \int_{l_j}^{l_{j+1}} \int_{g_k}^{g_{k+1}} F(s, l, g)\, ds\, dl\, dg$, s_i is the value of the solid volume at the upper end of the ith bin along the solid volume axis, l_j is the value of the liquid volume at the upper end of the jth bin along the liquid volume axis, g_k is the value of the gas volume at the upper end of the kth bin along the gas volume axis, Δs_i, Δl_j and Δg_k are the sizes of the ith, jth and kth bin with respect to the solid, liquid and gas volume axis. Using this technique, the population balance equation is reduced to a system of ordinary differential equations in terms of the rates of nucleation ($\Re_{nuc}(s_i, l_j, g_k)$), aggregation ($\Re_{agg}(s_i, l_j, g_k)$) and breakage ($\Re_{break}(s_i, l_j, g_k)$). The triple integral for the aggregation term can thereby be evaluated by casting it into simpler addition and multiplication terms. This approach is extended for application to the 4D population balance model via inclusion of the second solid component term.

References

Adetayo, A.A., Ennis, B.J., 1997. Unifying approach to modeling granule coalescence mechanisms. AIChE J. 43 (4), 927–934.

Amdahl, G.M., 1967. Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18–20, 1967, Spring Joint Computer Conference, AFIPS '67 (Spring), ACM, New York, NY, USA, pp. 483–485.

Annapragada, A., Neilly, J., October 1996. On the modelling of granulation processes: a short note. Powder Technol. 89, 83–84.

Barrasso, D., Ramachandran, R., 2012. A comparison of model order reduction techniques for a four-dimensional population balance model describing multi-component wet granulation processes. Chem. Eng. Sci. 80, 380–392.

Bilgili, E., Scarlett, B., 2005. Population balance modeling of non-linear effects in milling processes. Powder Technol. 153 (1), 59–71.

Chaudhury, A., Kapadia, A., Prakash, A.V., Barrasso, D., Ramachandran, R., 2013. An extended cell-average technique for a multi-dimensional population balance of granulation describing aggregation and breakage. Adv. Powder Technol. http://dx.doi.org/10.1016/j.apt.2013.01.006

Chen, C.-C., Mathias, P.M., 2002. Applied thermodynamics for process modeling. AIChE J. 48 (2), 194–200.

Cundall, P.A., Strack, O.D.L., 1979. A discrete numerical model for granular assemblies. Geotechnique 29 (1), 47–65.

Dosta, M., Heinrich, S., Werther, J., 2010. Fluidized bed spray granulation: analysis of the system behaviour by means of dynamic flowsheet simulation. Powder Technol. 204, 71–82.

Duncan, R., February 1990. A survey of parallel computer architectures. Computer 23, 5–16.

Freireich, B., Li, J., Litster, J., Wassgren, C., 2011. Incorporating particle flow information from discrete element simulations in population balance models of mixer-coaters. Chem. Eng. Sci. 66 (16), 3592–3604.

Ganesan, S., Tobiska, L., 2011. An operator-splitting finite element method for the efficient parallel solution of multidimensional population balance systems. Chem. Eng. Sci 69 (1), 59–68.

Gantt, J.A., Cameron, I.T., Litster, J.D., Gatzke, E.P., 2006. Determination of coalescence kernels for high-shear granulation using DEM simulations. Powder Technol. 170 (2), 53–63.

Gantt, J.A., Gatzke, E.P., 2006. A stochastic technique for multidimensional granulation modeling. AIChE J. 52 (9), 3067–3077.

Gunawan, R., Fusman, I., Braatz, R.D., 2008. Parallel high-resolution finite volume simulation of particulate processes. AIChE J. 54 (6), 1449–1458.

Gupta, A., Gupta A., Kumar, V., 1993. Performance properties of large scale parallel systems. Tech. rep. PhD thesis, Department of Computer Science, University of Minnesota.

Heath, M.T., 2011. Parallel numerical algorithm. Parallel Performance. Lecture at University of Illinois at Urbana-Champaign (Chapter 4).

Hounslow, M., 1998. The population balance as a tool for understanding particle rate processes. KONA 16, 179–193.

Immanuel, C.D., Doyle III, F.J., 2003. Computationally efficient solution of population balance models incorporating nucleation, growth and coagulation: application to emulsion polymerization. Chem. Eng. Sci. 58 (16), 3681–3698.

Immanuel, C.D., Doyle III, F.J., 2005. Solution technique for a multi-dimensional population balance model describing granulation processes. Powder Technol. 156 (2–3), 213–225.

Iveson, S.M., 2002. Limitations of one-dimensional population balance models of wet granulation processes. Powder Technol. 124 (3), 219–229, Control of Particulate Processess IV.

Iveson, S.M., Litster, J.D.H.K., Ennis, B.J., 2001. Nucleation, growth and breakage phenomena in agitated wet granulation processes: a review. Powder Technol. 117 (1), 3–39.

Klockner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., Fasih, A., 2011. PyCUDA and PyOpenCL: a scripting-based approach to GPU run-time code generation. Parallel Comput. 911, 1–24.

Kumar, J., Peglow, M., Warnecke, G., Heinrich, S., Mrl, L., 2006. Improved accuracy and convergence of discretized population balance for aggregation: the cell average technique. Chem. Eng. Sci. 61 (10), 3327–3342.

Lee, K., Kim, T., Rajniak, P., 2008. Compositional distributions in multicomponent aggregation. Chem. Eng. Sci. 63, 1293–1303.

Madec, L., Falk, L., Plasari, E., 2003. Modelling of the agglomeration in suspension process with multidimensional kernels. Powder Technol. 130 (13), 147–153.

Marshall, C.L., Rajniak, P., Matsoukas, T., 2011. Numerical simulations of two-component granulation: comparison of three methods. Chem. Eng. Res. Des. 89 (5), 545–552.

Marshall Jr., C.L., Rajniak, P., Matsoukas, T., 2012. Multi-component population balance modeling of granulation with continuous addition of binder. Powder Technol., http://dx.doi.org/10.1016/j.powtec.2012.01.027

MathWorks, September 2011. Parallel Computing Toolbox: Product description. MathWorks Inc., 3 Apple Hill Drive, Natick, MA 01760-2098, USA. URL: http://www.mathworks.com/help/distcomp/index.html

Matthews, H., Miller, S.M., Rawlings, J.B., 1996. Model identification for crystallization: theory and experimental verification. Powder Technol. 88 (3), 227–235.

Muzzio, F.J., Shinbrot, T., Glasser, B.J., 2002. Powder technology in the pharmaceutical industry: the need to catch up fast. Powder Technol. 124 (1-2), 1–7.

Pandya, J., Spielman, L., 1983. Floc breakage in agitated suspensions: effect of agitation rate. Chem. Eng. Sci. 38 (12), 1983–1992.

Pinto, M.A., Immanuel, C.D., Doyle, F.J., 2007. A feasible solution technique for higher-dimensional population balance models. Comput. Chem. Eng. 31 (10), 1242–1256.

Poon, J.M.-H., Immanuel, C.D., Doyle III, F.J., Litster, J.D., 2008. A three-dimensional population balance model of granulation with a mechanistic representation of the nucleation and aggregation phenomena. Chem. Eng. Sci. 63 (5), 1315–1329.

Poon, J.M.H., Ramachandran, R., Sanders, C.F.W., Glaser, T., Immanuel, C.D., Doyle III, F.J., Litster, J.D., Stepanek, F., Wang, F.Y., Cameron, I.T., 2009. Experimental validation studies on a multi-scale and multi-dimensional population balance model of batch granulation. Chem. Eng. Sci. 64, 775–786.

Rajniak, P., Stepanek, F., Dhanasekharan, K., Fan, R., Mancinelli, C., Chern, R.T., 2009. A combined experimental and computational study of wet granulation in a Wurster fluid bed granulator. Powder Technol. 189, 190–201.

Ramachandran, R., Ansari, M.A., Chaudhury, A., Kapadia, A., Prakash, A.V., Stepanek, F., 2012. A quantitative assessment of the influence of primary particle size polydispersity on granule inhomogeneity. Chem. Eng. Sci. 71 (0), 104–110.

Ramachandran, R., Barton, P.I., 2010. Effective parameter estimation within a multi-dimensional population balance model framework. Chem. Eng. Sci. 65 (16), 4884–4893.

Ramachandran, R., Chaudhury, A., 2012. Model-based design and control of a continuous drum granulation process. Chem. Eng. Res. Des. 90 (8), 1063–1073.

Ramachandran, R., Immanuel, C.D., Stepanek, F., Litster, J.D., Doyle III, F.J., 2009. A mechanistic model for breakage in population balances of granulation: theoretical kernel development and experimental validation. Chem. Eng. Res. Des. 87 (4), 598–614.

Ramkrishna, D., 2000. Population Balances: Theory and Applications to Particulate Systems Engineering. Academic Press, San diego.

Salman, A.D., Seville, J.P., Hounslow, M., 2007. Granulation. In: Vol. 11 of Handbook of Powder Technology Series, 1st ed. Elsevier Science, Oxford.

Sastry, K.V., 1975. Similarity size distribution of agglomerates during their growth by coalescence in granulation or green pelletization. Int. J. Miner. Process. 2 (2), 187–203.

Stepanek, F., Rajniak, P., Mancinelli, C., Chern, R., Ramachandran, R., 2009. Distribution and accessibility of binder in wet granules. Powder Technol. 189 (2), 376–384.

Verkoeijen, D., Pouw, G.A., Meesters, G.M.H., Scarlett, B., 2002. Population balances for particulate processes – a volume approach. Chem. Eng. Sci. 57 (12), 2287–2303.

Warg, F., 2006. Techniques to reduce thread-level speculation overhead. Ph.D. Thesis. Department of Computer Science and Engineering, Chalmers University of Technology.

Wilkinson, B., Allen, M., 1999. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, 1st ed. Prentice Hall, Lebanon, IN, USA.

Wu, X., Li, W., 1998. Performance models for scalable cluster computing. J. Syst. Architect. 44 (34), 189–205.