Computers and Chemical Engineering xxx (2017) xxx–xxx

Contents lists available at ScienceDirect

Computers and Chemical Engineering

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

Parallelization methods for efficient simulation of high dimensional population balance models of granulation

Franklin E. Bettencourt, Anik Chaturbedi, Rohit Ramachandran*

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

ARTICLE INFO

Article history:
Received 3 October 2016
Received in revised form 20 February 2017
Accepted 22 February 2017
Available online xxx

Keywords:
MPI
OpenMP
Parallel computing
Population balance model
Granulation
Pharmaceutical process design

ABSTRACT

In order to solve high resolution PBMs to simulate real systems, with high accuracy and speed, a comprehensive and robust parallelization framework is needed. In this work, parallelization using just Message Passing Interface (MPI) and a more advanced method using a hybrid MPI + OpenMP (Open Multi-Processing) technique, have been applied to simulate high resolution PBMs on the computing clusters, SOEHPC and Stampede. We study the speed up and the scale up of these parallelization techniques for different system sizes and different computer architectures to come up with one of the fastest ways to solve a PBM to date. Parallel PBMs ran approximately 50–60 times faster, when using 128 cores, than the serial PBMs ran. In this work it is found that hybrid MPI + OMP methods which account for socket affinities led to the fastest PBM compute times and about 80% less memory than a purely MPI approach.

1. Introduction and objectives

Particulate processes represent half of all industrial chemical production (Seville et al., 1997). Some examples of particulate processes include crystallization (Sen et al., 2014), granulation (Barrasso et al., 2015), milling (Barrasso et al., 2013a) and biological processes (Ramakrishna and Singh, 2014). Some important products made using particulate processes are fertilizers, detergents, aerosols, and pharmaceuticals (Prakash et al., 2013b). Due to the prevalence of particulate processes there is a great demand for methods, which can be used to aid in the detailed study and design of these processes.

The accurate modeling of particulate systems is crucial to enhance process development, production, equipment design, optimization, and process control (Ramachandran and Barton, 2010). However, these systems are very difficult to model due to complex micro phenomena that take place within them (Muzzio et al., 2002). Population balance models (PBMs) have proven to be one of the most efficient ways to model these systems while still capturing much of the physics that takes place in them (Ramakrishna and Singh, 2014).

Even though PBMs are one of the most efficient (fastest) ways to model these processes, high resolution and high dimensional models can still take days to solve and optimize via the use of conventional desktop/workstations (Prakash et al., 2013b). High resolution refers to the classification of particles into a large number of size classes (i.e. bins) and the division of the granulator volume into fine segments and high dimensional implies multiple internal/external variables of high resolution grids. To optimize the design of a process, or to tune a model based on a real system, a PBM will need to be run tens or hundreds of times creating a great need for faster simulations. There is also the push to increase PBM speeds to utilize them for model predictive control (MPC) and feed forward controllers (Christofides et al., 2007). In the past when scientists have faced computationally intensive problems they would use many CPUs working together to solve the problem faster, a method referred to as parallel programming (Wilkinson and Allen, 1997). Using modern computing clusters to evaluate a PBM in parallel has a great deal of promise. Two applications used for parallel programming are Message Passing Interface (MPI) and Open Multi-processing (OMP) both of which have different strengths and weaknesses.

1.1. Objectives

The overall objective of this study is to develop more efficient parallelization methods to reduce the computational times of high-dimensional PBMs. To that effect, specific objectives are listed below:

* Corresponding author at: 98 Brett Rd, Piscataway Township, NJ 08854, USA.
E-mail address: rohitrr@rci.rutgers.edu (R. Ramachandran).

http://dx.doi.org/10.1016/j.compchemeng.2017.02.043
0098-1354/

- MPI and hybrid MPI + OMP methods were used to parallelize a PBM.
- The performance of the parallelized models was tested using different criteria such as speed up, parallel efficiency, and memory usage.
- The hybrid MPI + OMP approach was compared to the purely MPI approach and was also compared with other works in the literature.

2. Background and motivation

2.1. Population balance models

PBMs have been effectively used to study granulation with a great deal of accuracy. Granulation is a process for engineering particles via liquid or solid binders to form larger aggregate granules with desired traits such as particle size distribution (PSD) and bulk density. The mechanism of particle growth and breakage are referred to as rate processes (Barrasso et al., 2013b). From a process stand point PBMs are used to simulate how the distribution of a set of particles, with varying properties, will change due to the systems rate processes over time (Barrasso et al., 2013b). However more generally, PBMs evaluate how a population distribution of entities is effected by its environment over time (Ramakrishna and Singh, 2014). A general form of population balance model, which applies to granulation systems, is shown as Eq. (1) below (Barrasso et al., 2013b),

$$\frac{\partial F(x,z,t)}{\partial t} + \frac{\partial}{\partial x}\left[F(x,z,t)\frac{dx}{dt}(x,z,t)\right] + \frac{\partial}{\partial z}\left[F(x,z,t)\frac{dz}{dt}(x,z,t)\right] = \Re_{formation}(x,z,t) - \Re_{depletion}(x,z,t) + \dot{F}_{in}(x,z,t) - \dot{F}_{out}(x,z,t) \quad (1)$$

where x is a vector which represents some or all of the characteristic properties of the particles in the system. Here x is a vector of internal parameters and is used to represent solid (s), liquid (l), and gas (g) contents of the particles. Here, z is a vector of external coordinates and represents spatial variance in the system of interest. F represents the population densities of particles described by the vectors x and z.

The first term on the left accounts for the rate of change of the particle number density, with time. The second term from the left, represents the rate of changes of population densities as the values of x are changed due to growth terms (dx/dt). In granulation systems this term would be associated with layering, consolidation, and liquid addition. The third term describes the change in population density over the physical space of the system due to movement, dz/dt is the particle velocity. For a continuous system F_{in} and F_{out} are the rates of particle inflow and outflow respectively. For a batch process F_{in} and F_{out} are both set to zero. The $\Re_{depletion}$ and $\Re_{formation}$ describe the net changes due to the rate processes of nucleation, aggregation, and breakage (Barrasso et al., 2013b).

2.2. Parallelization and parallel computing

2.2.1. Overview

In general parallel computing is taking a problem, breaking it into parts which can be solved independently, and distributing those small independent parts to many computers/computing cores to be evaluated all at the same time, i.e. in parallel (Wilkinson and Allen, 1997).

2.2.2. Computer architecture

Computing clusters are made up of many nodes connected by means of some sort of communication network such as InfiniBand, a type of high speed wireless communication, or Ethernet. Each node is analogous to a PC (personal computer) in that it has one or more CPUs (central processing unit), RAM (random access memory), a motherboard, cooling systems, and possibly GPUs (graphics processing unit) or co-processors. Commonly CPUs are multi-core processors, which means they have multiple compute units or cores, which can carry out independent computations. CPUs come with their own built-in memory which is very fast and heavily used for computation which is called the cache. Another important type of memory is the RAM, CPUs will have a direct connection through the motherboard's CPU socket to the RAM. RAM is slower than cache memory and therefore efficient memory usage, which optimizes cache utilization and minimizes data transfers from the RAM to the CPU and from the CPU to the RAM is critical for fast computation. Large amounts of memory transfer from the RAM to the CPU can greatly limit the speed at which computation can be performed. Furthermore, data movement in general is one of the greatest limitations to the performance of a parallel application. For the previous reasons to maximize performance network message passing should be minimized and when memory is needed cache utilization should be optimized.

Computer architectures, are often classified as distributed memory machines, shared memory machines, or some sort of combination of distributed and shared. Clusters are actually distributed memory systems (nodes have separate memory from one another) with shared memory subsystems (the CPU cores on a node can operate in shared memory mode). Additional architectural features arise when a node has more than one CPU, most nodes these days support two CPUs. Even though a node can be operated in shared memory, the CPUs on the node cannot access all of the memory on that node at the same speeds. One CPU can access memory that it is connected to through its socket much faster than it can access memory stored in a memory bank associated with the other CPU socket on the node. These kinds of systems are called non-uniform memory access (NUMA) clusters (Jin et al., 2011). The compute cores of a single CPU can all potentially access the same RAM at the same speed since they are all connected to it through the same socket. Since the cores on a CPU can all draw from the same RAM they share the memory, and are referred to as a shared memory system. Two nodes are distributed machines with respect to another because the cores on one node do not share the same physical RAM as the cores on the other node. Due to this the interplay between computer architecture, software, and computational load distribution need to be considered when developing a parallel program (Adhianto and Chapman, 2007).

2.2.3. Parallelization applications and practices

Message passing interface (MPI) and open source message passing (OMP) are two application program interfaces, which have been commonly used to parallelize computations. MPI is an application which is used for distributed memory computing (Jin et al., 2011). When just MPI is used, all of the CPU cores on all of the nodes have their own private local memory, meaning core-to-core communication must be done by explicit message passing even on hardware that can utilize shared memory (Jin et al., 2011). Large amounts of message passing can lead to bottlenecking due to message passing overhead and due to overloading of the bandwidth/communication network (Jin et al., 2011). Using only MPI results in each core needing its own copy of all variables used for the computation it is performing, resulting in inefficient memory usage due to large numbers of duplicate variable storage. Even on shared memory architecture systems MPI will operate each core as a distinct unit with its own private memory.

Please cite this article in press as: Bettencourt, F.E., et al., Parallelization methods for efficient simulation of high dimensional population balance models of granulation. Computers and Chemical Engineering (2017), http://dx.doi.org/10.1016/j.compchemeng.2017.02.043

OMP is an application for parallelization on shared memory systems. This means that changes in the local memory are seen by all of the cores using that local memory, with no need for explicit message passing (Jin et al., 2011). The downside is that OMP alone cannot be used to parallelize computation over more than one shared memory machine, preventing it from being able to take advantage of the multiple nodes of a cluster (Jin et al., 2011). Therefore, there is impetus to combine the advantages of both MPI and OMP in the form of a hybrid MPI + OMP technique.

2.2.4. Previous studies on hybrid MPI + OMP

Hybrid MPI + OMP methods of parallelization are relatively new and have been studied in far less detail than either MPI or OMP separately. Most commonly, it has been studied in applications to computational fluid dynamic programs which are needed to solve systems of ODEs and to date no work has been done which uses the hybrid approach for solving PBMs and their complex integro-differential equations.

Due to the short comings of a purely MPI or purely OMP approach, the hybrid MPI + OMP approach has become increasingly studied and utilized over the years. Jin et al. (2011) found that the hybrid method was faster than purely MPI since there were less messages being sent and received, which additionally leads to reduced memory usage due to smaller message buffer needs and the ability to take advantage of shared memory storage on each node.

Gorobets et al. (2011) found that above a certain size the hybrid MPI + OMP method was slightly faster than the purely MPI method. Most importantly they found that the hybrid approach allowed for the utilization of far more cores than was possible for the MPI process, which allowed them to scale the simulation to a far greater size. Another added benefit was that the hybrid approach made the code more easily adapted to the clusters it could be used on. The hybrid approach was faster because only one MPI process had to send a message to collect the information from each node. The reason only one MPI process had to send a message per node was because all of the cores on the node were parallelized with OMP, meaning they were operating in shared memory, so all of the cores automatically updated the MPI process on the node and the one MPI process could send the results of all of the cores for that node with one message.

In another study, Wei and Yilmaz (2011) found that the utilization of the hybrid parallelization methods reduced memory usage and reduced simulation time by decreasing the total number of message passes and reduced the impact of latency on communication.

Rabenseifner et al. (2009) and Mininni et al. (2011) found that the thread per CPU ratios, MPI processes socket affinity, and thread core affinity, are major indicators of performance and need to be considered by the programmer in relation to the system hardware the program will be implemented on. They found that it may be better to have one MPI process per CPU socket rather than per node, since the two sockets have different access speeds to different memory on the same node. It was also found that the hybrid implementation allowed for better scalability due to reduced memory usage. Interestingly the hybrid approach potentially can allow for a simpler implementation of the MPI portion of the code as there is no need for added shared-memory libraries since the OMP takes care of intra-node communications.

In summary, many studies found that as computing systems trend toward having more cores per node the hybrid approach becomes more efficient than purely MPI does in many cases (Jin et al., 2011; Gorobets et al., 2011; Wei and Yilmaz, 2011; Rabenseifner et al., 2009). The general conclusion as to why this was the case, was because the hybrid implementation required far less message passing since potentially only one message is sent per node or per CPU, reducing latency significantly but still saturating the communication networks available (Jin et al., 2011; Gorobets et al., 2011; Wei and Yilmaz, 2011; Rabenseifner et al., 2009). The hybrid approach also led to a smaller over all memory footprint which made it more appropriate as supercomputers trend toward having less resources per core such as memory or bandwidth (Jin et al., 2011; Gorobets et al., 2011; Wei and Yilmaz, 2011; Rabenseifner et al., 2009).

2.2.5. Previous work on parallel PBMs

Due to the past success parallelization has had for other computationally intensive problems (Wilkinson and Allen, 1997), there has been interest in its application to PBMs. Prakash et al. (2013a, 2013b) utilized Matlab's Parallel Computation Toolbox (PCT) to parallelize the solution of a multidimensional PBM. Because it was performed using MATLAB it was easy to implement but lacked the performance of parallelized FORTRAN or C and furthermore, was only tested for a very small number of cores.

Gunawan et al. (2008) carried out parallelization studies of PBMs using high-resolution finite volumes solution methods and used load balancing techniques to effectively distribute workloads since they decomposed the internal coordinates of their PBM. Gunawan et al. (2008) studied the scale up of the parallel model on up to 100 cores achieving good efficiencies, suggesting an excellent parallelization technique. However, the method was not tested for highly dimensional models or with large internal or external coordinate domains. They also only carried out scaling studies for one system size which did not show which dimensions had the largest impact on the performance of their parallel model. Gunawan et al. (2008) described in their work the potential benefits that shared memory processing could bring to parallel computation of PBMs as it would significantly reduce message passing, suggesting that a hybrid MPI + OMP approach could improve performance further – which is the focus of this study.

In other work, Ganesan and Tobiska (2012) studied the parallelization of a spatially decomposed finite differences solution to their PBMs, which didn't need load balancing. Their usage of high performance matrix solvers and mathematical manipulation allowed them to partition their workloads very effectively. However, their model was not tested for PBMs with more than three dimensions or for large internal/external coordinate domains. Also no speed up studies were performed.

In this current work, a hybrid MPI + OMP methods were used to parallelize the evaluation of a five dimensional (5D) PBM using a finite differences approach. Parallelization of 5D PBMs is necessary since as they grow in size and complexity, they take significantly more time to evaluate. These are also needed to realistically depict the numerous dimensions/traits the model needs to capture to accurately represent the real granulation process. There is also a tradeoff between speed and accuracy of these models. Effective parallelization would allow for these 5D PBMs to be evaluated in a reasonable period of time, allowing engineers to utilize far more accurate models for better design of products and procedures. The methods of this work could be coupled with the work by Gunawan et al. (2008) and Ganesan and Tobiska (2012) to more efficiently use bandwidth and memory, further increasing performance.

3. Computational and parallelization methods

3.1. 5D population balance model development for wet granulation

The population balance model used in this current work was derived from the work of Barrasso et al. (2013b). The model developed by Barrasso et al. (2013b) was used to model continuous twin

Please cite this article in press as: Bettencourt, F.E., et al., Parallelization methods for efficient simulation of high dimensional population balance models of granulation. Computers and Chemical Engineering (2017), http://dx.doi.org/10.1016/j.compchemeng.2017.02.043

screw granulation of two types of solids, one type of liquid, and the gas parameter was lumped into the other three internal coordinates. The operation simulated in this current work was high shear batch granulation with one solid type, one liquid, and one gas type. Different operations and species types result in some changes needing to be made to adapt the five dimensional PBM developed by Barrasso et al. (2013b). After adaptation the following PBM was formulated.

$$\frac{\partial}{\partial t}F(s,l,g,\mathrm{x},\mathrm{y},t)+\frac{\partial}{\partial l}[F(s,l,g,x,y,t)\frac{dl}{dt}(s,l,g,x,y)] +\frac{\partial}{\partial g}[F(s,l,g,x,y,t)\frac{dg}{dt}(s,l,g,x,y)] +\frac{\partial}{\partial x}[F(s,l,g,x,y,t)\frac{dx}{dt}(s,l,g,x,y)] +\frac{\partial}{\partial y}[F(s,l,g,x,y,t)\frac{dy}{dt}(s,l,g,x,y)]=\Re_{agg}+\Re_{break} \quad (2)$$

where F is the number of particles with solid, liquid and gas volume of s, l and g respectively at time t. The $\frac{\partial F(s,l,g,x,y,t)}{\partial t}$ term represents the rate of change in the number of particles due to the various rate processes taking place. $\frac{\partial}{\partial l}[F(s,l,g,x,yt)\frac{dl}{dt}(s,l,g,x,y,t)]$ describes the rate of change of number of particles in the liquid direction due to external liquid addition to the system. Similarly, $\frac{\partial}{\partial g}[F(s,l,g,x,y,t)\frac{dg}{dt}(s,l,g,x,y,t)]$ represents the net rate of change in particle number due to consolidation. $\frac{\partial}{\partial x}[F(s,l,g,x,y,t)\frac{dx}{dt}(s,l,g,x,y)]$ and $\frac{\partial}{\partial y}[F(s,l,g,x,y,t)\frac{dy}{dt}(s,l,g,x,y)]$ describe the movement of the particles along the x and y directions respectively. $\Re_{agg}$ and $\Re_{break}$ are the rate of change of particle number due to aggregation and breakage respectively. Please refer to the appendix for more detailed explanation of rate processes and the evaluation of the aggregation and breakage rates.

3.1.1. Discretization and numerical techniques

For this work linear incremented sized bins were used for the x and y coordinates as well as the internal s, l, and g coordinates. It is important to note that commonly s, l, and g coordinates are given nonlinear spacing, which is also how authors of Barrasso et al. (2013b) spaced their internal coordinates.

To numerically solve the population balance, model a finite differences method was used. The spatial and internal coordinates were discretized which created a system of ordinary differential equations. To evaluate the system of ODEs, first order Euler integration was in this work, and is commonly used by others to solve multi-dimensional PBMs (Barrasso et al., 2015; Barrasso and Ramachandran, 2015; Chaudhury et al., 2014, 2015a,b). The Euler integration was used over more complex integration techniques because the authors of (Barrasso et al., 2013b) found that compared to the RK2 integration the Euler method was twice as fast and yielded nearly identical results. Due to the explicit nature of the Euler integration technique the program was made to satisfy Courant–Friedrichs–Lewis (CFL) to ensure numerical stability (Ramachandran and Barton, 2010). The time-step selected based on the CFL criterion was such that, the number of particles leaving a particular size class at any time is less than the number of particles present in that size class at that time (Ramachandran and Barton, 2010). The parameters used for the population balance model are detailed in Table 1.

Table 1
Parameters used for PBM studied in this work.

Parameter name	Value	Units
Time step (Δt)	0.1	s
Total granulation time (T)	20, 120*	s
Velocity in axial direction (v_{axial})	1	m/s
Velocity in radial direction (v_{radial})	1	m/s
Aggregation constant (β_0)	1×10^{-12}	–
Aggregation constant (α)	1	–
Aggregation constant (δ)	1	–
Initial particle radius (R)	15	μm
Breakage Kernel constant (B)	5×10^{4}	–
Diameter of impeller (D)	0.1	m
Impeller rotational speed (RPM)	300	rpm
Minimum granule porosity ($\in_{min}$)	0.1	–
Consolidation rate (C)	1×10^{-3}	–
Total starting particles in batch ($F_{initial}$)	1×10^{6}	–
Liquid to solid ratio (L/S)	0.7	–
Number of solid bins (S)	10, 15**	–
Number of Liquid bins (L)	10, 15**	–
Number of gas bins (G)	10, 15**	–
Number of spatial X bins (X)	240	–
Number of spatial Y bins (Y)	5	–

* For this study two sets of times were used, 20 s and 120 s.
** For this study two bin sizes were tested 10 of each s, l, and g and 15 of each s, l, and g.

3.2. Parallelization methodology

As stated earlier, today's clusters have many nodes with separate memory but each node usually has many compute cores and these cores can operate in shared memory. To effectively utilize this aspect of these clusters MPI was used for inter-node message passing. Several scenarios were tested using one MPI process per node, one MPI process per CPU (two MPI processes per node), and two MPI processes per CPU. Then each MPI process was parallelized further (to the extent that each core had one thread) using OMP to utilize the benefits of shared memory for the cores within each MPI process. This was done with two goals in mind. The first was to reduce the number of message passes, since many of the compute cores operated in shared memory mode only one message needed to be passed per MPI process being used in order to collect all of the computational work done by all of the cores within each MPI process. The second goal was to reduce the overall memory needed per core and per node, as stated by Jin et al. (2011) hybrid MPI + OMP reduces memory usage. The hybrid approach uses less memory because many of the variables can be shared between all the cores on a node, which are operating in shared memory mode. Using less memory is crucial since there are many variables which have dimensions of *s-l-g-x-y* and *s-l-g-s-l-g*, as the size and detail of a PBM grow these variables become large, shared memory allows the cores operating in shared memory together, to share one copy of these variables. Using only MPI results in one copy of each variable for each core, resulting in duplicate variables and huge amounts of memory usage which can limit the size of PBM which can be studied on a given system.

The program model was coded in FORTRAN utilizing MPI and OpenMP. The main computer cluster used for this study was the Rutgers School of Engineering High Performance Computing (SOEHPC) cluster. It has 80 nodes, each node has two Xeon E5-2670 with hyper-threading turned off (16 cores per node) and 128 GB of RAM. The nodes are interconnected via FDR infiniBand (56 Gb/s) and Ethernet networks (10 Gb/s), it utilizes a SLURM scheduler, and runs an Ubuntu (14.04) server operating system. Jobs were run using one core (serial) all the way up to 8 nodes (128 cores). On the SOEHPC the GCC-4.7.3 compilers were used, the OMP implementation used was the default GCC-4.7.3 implementation OpenMP 3.1, and the MPI implementation used was OpenMPI version 1.10.

The other cluster used for this work was the Texas Advanced Computing Center's Stampede cluster. The Stampede cluster architecture can be viewed on the TACC website. It has 32 GB of RAM on the nodes used for this study, a Xeon Phi accelerator, two Xeon E5-2680 V1 CPUs (16 cores), and the nodes are interconnected via

Please cite this article in press as: Bettencourt, F.E., et al., Parallelization methods for efficient simulation of high dimensional population balance models of granulation. Computers and Chemical Engineering (2017), http://dx.doi.org/10.1016/j.compchemeng.2017.02.043

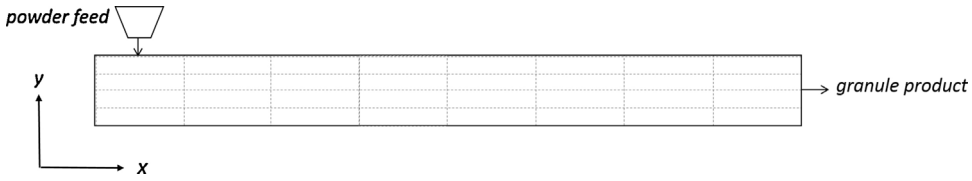

Fig. 1. Visual representation of spatial decomposition of physical granulator system being modeled by PBM.

FDR infiniBand. On Stampede the compiler used was Intel 15.0.2, the OpenMP implementation used was the default Intel 15.0.2 OMP version 3.1, and the MPI implementation used was MVAPICH2/2.1. The Xeon Phi accelerators were not used for this study as the *–no-offloading* flag was used to compile the PBM on Stampede.

To ensure that consistent PBM evaluation times were obtained for this study, all PBM evaluations were run three times and the simulation times were averaged.

3.3. Computational division of population balance model

To parallelize the PBM, spatial domain decomposition was used. Fig. 1 shows how the granulator was spatially decomposed. The dotted lines show one specific decomposition which is 4 bins in the *Y* direction and 8 bins in the *X* direction.

For this work the spatial domain was decomposed into 240 spatial bins along the *x* axis. The PBM was not decomposed on the *y* axis, however decomposition in *y* could have been used in addition to decomposition in *x*. As a result of this decomposition the smallest piece of the PBM and therefore smallest possible work load a core could receive, was a 1 unit in *x* by 5 units in y chunk.

A number of spatial bins were assigned to each computer core; each core would then evaluate rate process calculations for each of the spatial bins it was assigned. Once complete each core would send the updated particle size distribution (*F*) values to the master processor at the end of each time step.

The master processor would collect all of the updated particle size distributions from each slave core at every time step. The master core would then compute all of the particle transfers from one bin to another in serial, and broadcast the updated *F* values to the slave processes to be used for computation in the next time step. The parallelization and decomposition frameworks are shown below with the following pseudo code (Fig. 2).

```
Start Program
Initiate OpenMPI
Do
      {serial} - Time-Independent Calculations
End do
!$Initiate OpenMP threading
Do time loop
      Do spatial loop calcs for x_lower to x_upper
                {Parallel} - PSD changes due to rate processes
      End spatial loop

            MPI_Send (slaves send updated F values to Master)
            MPI_Receive (master receives updated F values from slaves)
      If Master
      Do radial and axial bin-to-bin transition calcs
                      {Serial} - bin-to-bin particle transfers
      End radial and axial bin-to-bin transition calcs

      Bcast (master sends Updated F to slaves)
      End If
End Time loop
!$Finalize OpenMP
End OpenMPI
End program
```

Fig. 2. Pseudo code of the parallel population balance model developed for this study. Light gray box shows the parallel portion of the code done by the slave processors while the dark gray box shows the potion performed in serial by the master process.

3.4. Parallel computing performance metrics

The speed up of a parallel program shows how much faster a program runs as more compute cores are used to solve it compared to the wall clock runtime of the serial program. It is one of the most common metrics for showing how effective parallelization is at increasing computational speed. Speed up is defined as the time it takes for a program to run in serial divided by the time it takes to run in parallel and is shown as Eq. (3). The time it takes a program to run from start to finish is referred to as the wall clock time. From here on simulated process time will refer to the time the physical process being simulated would have been carried out in the real world. In an ideal case the parallel program should speed up "*n*" times, where "*n*" is the number of compute cores in parallel used to run the parallel program. Scale up is studying the speed up as the number of cores used to solve the problem in parallel is increased.

$$Speed\ Up = \frac{Serial\ Wall\ Clock\ Time}{Parallel\ Wall\ Clock\ Time} \tag{3}$$

Another similar metric is the parallel efficiency. Parallel efficiency is the speed up divided by the number of processors used. This makes the parallel efficiency the normalized speed up and gives the value of what fraction of the ideal or perfect speed up a program is achieving with each increase in the cores used in parallel.

$$Parallel\ Efficiency = \frac{Serial\ Run\ Time}{Parallel\ Wall\ Time * n_{cores}} \tag{4}$$

4. Results and discussions

4.1. Model validation methods

The model was first developed in Matlab® and was checked for mathematical accuracy. For aggregation, 2 smaller particles should agglomerate to form 1 larger particle and therefore should have a ratio of 0.5 particles formed to depleted for Aggregation (RFDA). When a large particle breaks, it should form 2 smaller particles, therefore the ratio of particles formed/depleted from Breakage (RFDB) should be 2. For all of the models run the ratios were correct, these results can be seen in Fig. 3a. Since the ratios were correct the model was consistent with the assumptions made about the mechanisms of aggregation and breakage (Barrasso et al., 2013b). These results also show that the model demonstrates mass conservation. The PSD of the Matlab® model was evaluated at 10, 60, and 120 s to show that particles were aggregating over time, this can be seen in Fig. 3b. Once the model was completely verified, the PSD for the serial, MPI, and the hybrid MPI + OMP codes were compared to each other and the serial Matlab code to ensure they were working properly, the results are shown in Fig. 3c.

4.2. Parallel results: speed up and scaling

4.2.1. MPI vs hybrid – SOEHPC scaling

The following figures in this section show speed up values for programs as they were run with more cores and with different

Please cite this article in press as: Bettencourt, F.E., et al., Parallelization methods for efficient simulation of high dimensional population balance models of granulation. Computers and Chemical Engineering (2017), http://dx.doi.org/10.1016/j.compchemeng.2017.02.043

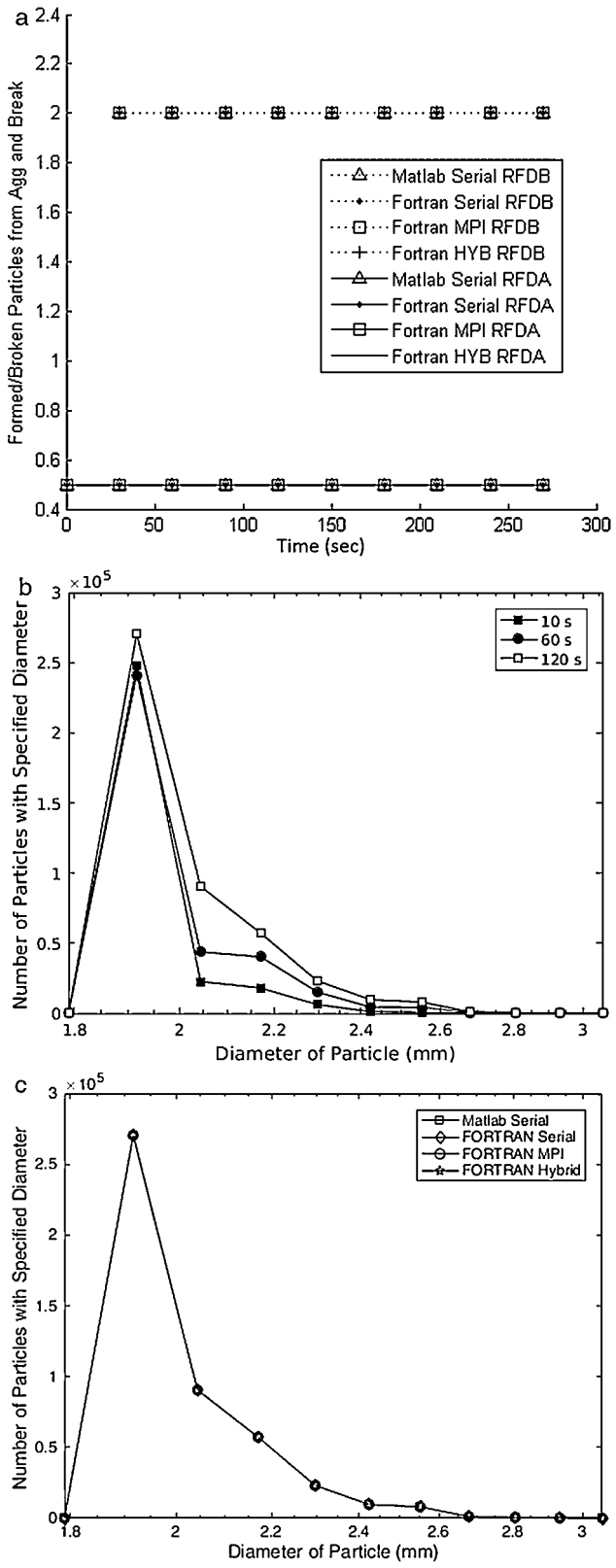

Fig. 3. (a) Ratio of the rate of particles formed to the rate of particles depleted from aggregation and breakage over time as granulation takes place. (b) The PSD at 10, 60, and 120 s of granulation. (c) The final particle size distribution after the process completion. It shows the number of particles in each size bin. A size bin represents a small range of particle sizes.

levels of compiler optimization. The compiler optimization options used were *–O2* and *–Ofast*. The hybrid and MPI programs are scaled using the same serial simulation run time for each PBM case. The serial code used was the direct translation of the Matlab program into FORTRAN, it implemented no MPI or OMP libraries.

It can be seen in Fig. 4a and Table 2 that the program scales relatively well reaching approximately 60 times speed up when using 128 cores in parallel. Fig. 4b shows relatively good parallel efficiency remaining above 40%. There is an initial sharp decline in efficiency for jobs of 2 and 4 cores, shown in the inset, which is due to the way the model was parallelized. In parallel there is a master process which deals with coordinating and communication and some number of slave processes which do all of the rate process calculations. When the PBM is run in parallel on just two cores there is a master and a slave which results in worse performance than if the PBM was solved serially. Due to the way the program was built, on just two cores, the slave computes all of the rate process calculations itself and sends the results to the master process. This is not an ideal method for small numbers of cores however as the number of cores increases this is a more preferable way to program as the master does not get bogged down by its own calculations as well as coordinating all of the slave processes. The decline in efficiency is very slow after 16 cores, eventually falling to approximately 50% efficiency at 128 cores. Table 2 shows various performance data for the *–O2* optimized simulations for the serial, the 128 core purely MPI, and the 128 core hybrid implementations. From Fig. 4 and Table 2 it can be seen that the *–O2* MPI and *–O2* hybrid programs run with similar performance.

For these studies the number of bins used to resolve the internal coordinates (solid-liquid-gas) of the PBM were varied to show how performance might be affected by increasing the detail of the solid, liquid, and gas bins. The two sets of bins used for the internal coordinates were 10-10-10 and 15-15-15, the number refers to the number of bins used and the position in the list refers to the internal coordinate type (solid-liquid-gas). Since number of solid, liquid, and gas bins were the same a single number was used to display the bin size for all three in Table 2 and Fig. 4. In all cases using *–O2* optimization there appears to be no major bottlenecking due to problem size that results in scaling differences since the 10-10-10 and 15-15-15 sized systems appear to scale nearly identically. There also does not appear to be a major difference in the speed up of the hybrid program and the purely MPI programs at the *–O2* level of optimization. This is interesting as the hybrid program passes less messages it should run faster if the system was limited by communication. Later in this work, based on studies conducted using the Stampede cluster, it is shown that the hybrid code using a single MPI instance per node is delayed by memory locality issues due to the two separate sockets on each node. These delays in memory retrieval slow the hybrid program to the same speed as the MPI program in this case. However, the hybrid model did utilize less memory (shown in later section) and would be advantageous for computer systems that have lower amounts of memory per core.

Fig. 5 shows the parallelization speed up and parallel efficiency for the *–Ofast* optimized programs. Table 3 shows performance metrics for the *–Ofast* optimized PBM simulations run on the SOEHPC cluster in serial and with 128 cores. Compared to the *–O2* optimized programs, the *–Ofast* optimized programs show far more variation in the run time and speed up. This could be for several reasons. One reason is that the *–Ofast* option is for aggressive optimization, in other words the speed gains come at the expense of program stability. Additionally, to improve speed rounding of calculations can take place. A list of all of the specific flags used for *–Ofast* compiled programs can be found in the GCC manual. Because of the rounding and the potential instabilities, the program out-

Please cite this article in press as: Bettencourt, F.E., et al., Parallelization methods for efficient simulation of high dimensional population balance models of granulation. Computers and Chemical Engineering (2017), http://dx.doi.org/10.1016/j.compchemeng.2017.02.043

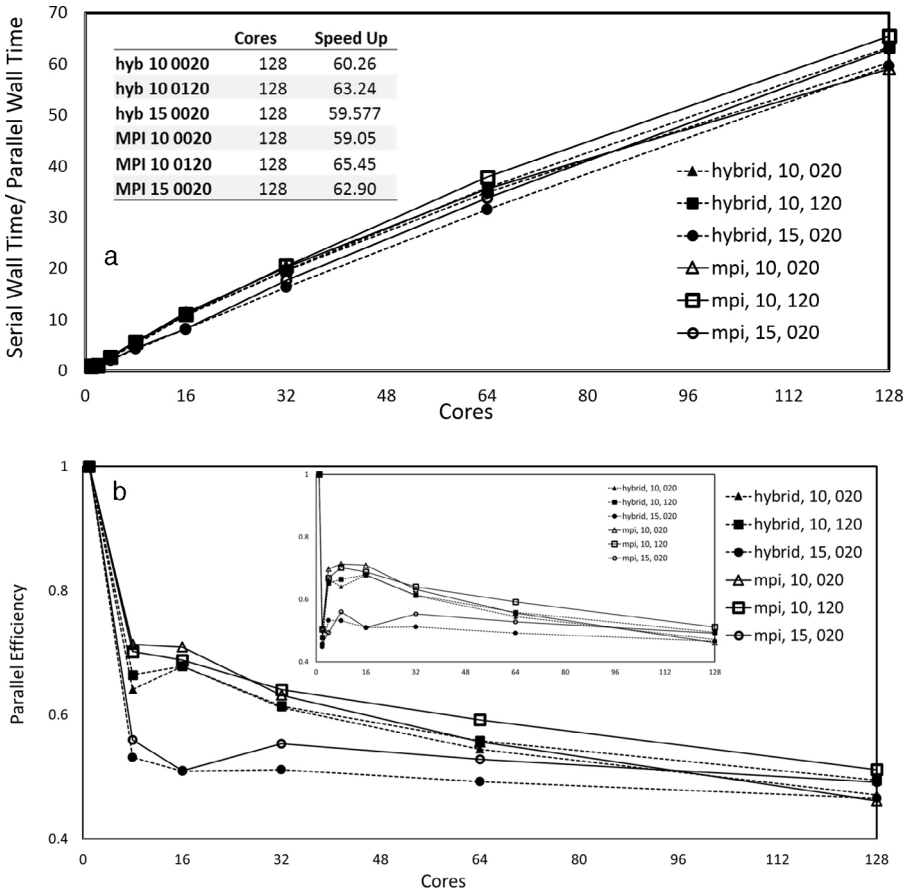

Fig. 4. (a) The plot shows the scaling results for *-O2* (default) optimized programs on SOEHPC cluster. The scaling was done using increments of 1, 2, 4, 8, 16, 32, 64, and 128 cores for all cases. The table inset details the speed up for each PBM when using 128 cores. (b) The parallel efficiencies of each job for all core allocations on the SOEHPC for *–O2* optimized programs. The legend lists jobs as hybrid or MPI, the number immediately after that is the number of size bins for *s*, *l*, and *g*, the number listed last is the physical process granulation time. The figure inset shows the full efficiency results including core allocations of 2 and 4.

Table 2
Summary of performance metrics for the *–O2* optimized PBM simulations run on the SOEHPC cluster.

Run type	Optimization	Bins	Granulation time	Cores	Wall time (s)	Hybrid wall time/MPI wall time	Speed up	Wall time/granulation time
Serial	*–O2*	10	20	1	3101			155.04
MPI	*–O2*	10	20	128	53		59.05	2.63
Hybrid	*–O2*	10	20	128	51	0.98	60.26	2.57
Serial	*–O2*	10	120	1	19080			159.00
MPI	*–O2*	10	120	128	292		65.45	2.43
Hybrid	*–O2*	10	120	128	302	1.03	63.24	2.51
Serial	*–O2*	15	20	1	46800			2339.98
MPI	*–O2*	15	20	128	744		62.90	37.20
Hybrid	*–O2*	15	20	128	786	1.06	59.58	39.28

Table 3
Summary of performance metrics for the *–Ofast* optimized PBM simulations run on the SOEHPC cluster.

Run type	Optimization	Bins	Granulation time	Cores	Wall time (s)	Hybrid wall time/MPI wall time	Speed up	Wall time/granulation time
Serial	*–Of*	10	20	1	537			26.83
MPI	*–Of*	10	20	128	16		34.11	0.79
Hybrid	*–Of*	10	20	128	17	1.06	32.16	0.83
Serial	*–Of*	10	120	1	2987			24.89
MPI	*–Of*	10	120	128	76		39.21	0.63
Hybrid	*–Of*	10	120	128	88	1.15	34.00	0.73
Serial	*–Of*	15	20	1	10072			503.58
MPI	*–Of*	15	20	128	226		44.47	11.32
Hybrid	*–Of*	15	20	128	213	0.94	47.77	10.65

put was checked against the *–O2* compiled model and there were no major differences between the accuracy of the *–Ofast* and *–O2* optimized programs.

Comparing the trends observed in Fig. 5a it is apparent that for both the hybrid and purely MPI cases the largest systems (15-15-15) scale better than the others when implemented on 128 cores.

Please cite this article in press as: Bettencourt, F.E., et al., Parallelization methods for efficient simulation of high dimensional population balance models of granulation. Computers and Chemical Engineering (2017), http://dx.doi.org/10.1016/j.compchemeng.2017.02.043

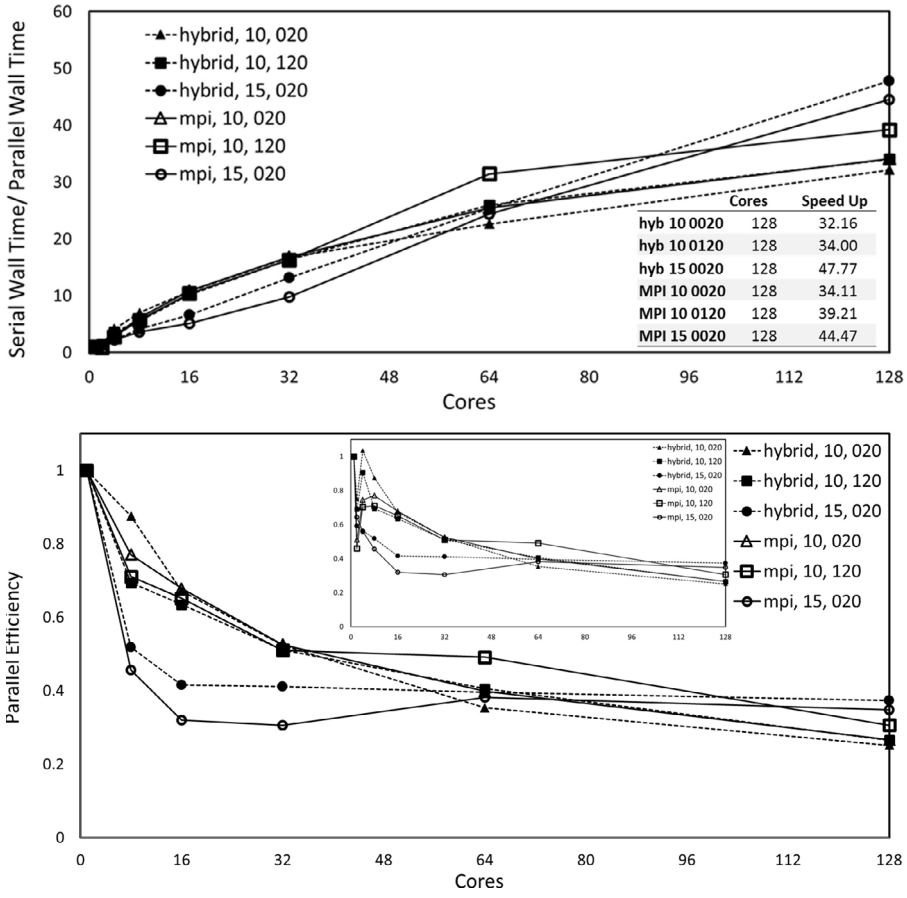

Fig. 5. (a) The scaling results for -*Ofast* optimized programs on SOEHPC cluster. The scaling was done using increments of 1, 2, 4, 8, 16, 32, 64, and 128 cores for all cases. The table inset details the speed up for each PBM when using 128 cores. (b) The parallel efficiency of each job for all core allocations on the SOEHPC for –*Ofast* optimized programs. The legend lists jobs as hybrid or MPI, the number immediately after that is the number of size bins for *s*, *l*, and *g*, the number listed last is the time in seconds that the simulated granulator was running for. The figure inset shows the full efficiency results including core allocations of 2 and 4.

This shows that the calculations in these larger systems take more time to complete than the message passing portion of the simulation, meaning at 128 cores the program is less communication limited than the 10-10-10 PBM simulations. This result is expected since the 15-15-15 has proportionally more rate process calculations to carry out than the 10-10-10 systems. Since the rate process calculations are solved in parallel and since the larger system has a larger ratio of rate process calculations to other calculations, the larger systems have a higher percentage of code which runs in parallel, which results in better parallel performance.

For the largest system (15-15-15) the hybrid program appears to be slightly faster, than the purely MPI program, which is likely due to the hybrid program's ability to handle larger data sets more effectively. Since the MPI program must handle significantly larger amounts of data it cannot fit all of the data needed into its cache memory resulting in more calls to the RAM and more cache misses than the hybrid program which slows performance. It could also be due to slightly faster message passing segments, since the hybrid code passes less inter-node messages, which would result in slightly faster step times. Since it does not appear that the 15-15-15 systems are limited by message passing the reason the hybrid program is faster than the purely MPI program is most likely because of the more efficient memory usage.

The two 10-10-10 systems studied show some interesting trends. For the shorter 20 second 10-10-10 systems it would appear that the hybrid program is slightly slower than it's purely MPI counterpart. This is partly due to additional overhead that comes with implementing OMP as well as MPI in the hybrid case. As mentioned in the section on –*O2* performance, the main reason the hybrid code under performs in this instance is because of memory locality issues which can be corrected using socket affinity options. However, the hybrid case uses less memory than the purely MPI case.

For the longer 120 second 10-10-10 system the purely MPI program outperformed the hybrid counterpart, in terms of speed up, by a larger margin than the other scenarios, particularly at 64 cores. This performance discrepancy between the purely MPI and hybrid programs was not anomalous since it was present in three repeated runs. Notice that there was only a slight scaling difference between the hybrid and MPI programs for the 20 second 10-10-10 runs, it appears that the additional time steps of the 120 second run amplified the difference. This would also support the finding that the memory locality has a significant impact in the hybrid performance. Since the hybrid code faces delays in computation due to memory locality issues the hybrid code would fall further behind the MPI program at each time step. Therefore the performance difference widened as the programs ran for the longer 120-s interval.

It is important to note that the best speed up values were obtained using the –*O2* compiler option, while the shorter compute times were achieved using the –*Ofast* option. The reason for this is that the –*Ofast* option significantly sped up each computation which had a larger impact on the serial job since there was no message passing. Since the speed up is defined as serial-wall-time/parallel-wall-time and because the serial code had a larger performance gain from the –*Ofast* option, the overall parallel speed up was less for the –*Ofast* simulations than for the –*O2* simulations.

Please cite this article in press as: Bettencourt, F.E., et al., Parallelization methods for efficient simulation of high dimensional population balance models of granulation. Computers and Chemical Engineering (2017), http://dx.doi.org/10.1016/j.compchemeng.2017.02.043

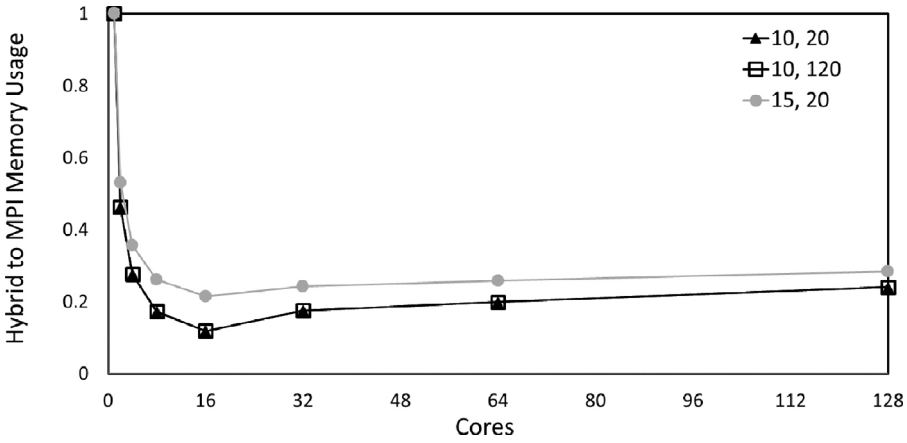

Fig. 6. Ratio of the memory used by hybrid jobs to the memory used by the MPI jobs for both levels of optimization. The trend lines for the 20 second and 120 second 10-10-10 systems overlap.

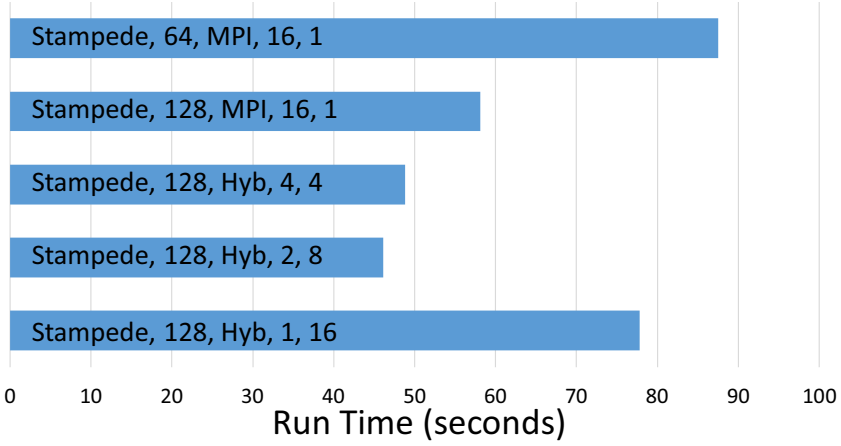

Fig. 7. Shows the wall clock run times of various jobs on Stampede as different combinations of MPI processes and OMP threads are allocated to each CPU socket on each node. This was done to show how socket affinity effects performance. The label on each bar lists the cluster name, the total number of cores used for that job, whether its hybrid or MPI, the number of MPI processes per node, and the number of OMP threads per MPI process. Each job was run using *tacc_affinity* and was compiled using the Intel 15.0.2 compilers with the *–O3* option. All of the jobs run for this figure were $ns = 10$ $nl = 10$ $ng = 10$ $nx = 240$ $ny = 05$ and time = 120 s.

However the fastest compute times were achieved using the *–Ofast* option in all cases.

4.2.2. Hybrid vs MPI – memory usage

From Fig. 6 it can be seen that the hybrid job uses far less memory than its MPI counterpart. In Fig. 6 the curves for both of the 10-10-10 systems fall directly on top of each other as they should since there are no arrays saved for each time step. Even at 128 cores the hybrid jobs used approximately 80% less memory than the MPI jobs used. This is useful as the hybrid program can be run on a larger variety of clusters that may have less memory per core.

4.2.3. Socket affinity studies on stampede

Fig. 7 shows wall clock runtimes for jobs run on the Stampede computer cluster. The simulations that were run using 128 compute cores in parallel were able to solve the PBMs faster than the granulation process would take place in real life. Importantly Fig. 7 shows the impact of socket affinity on program run time as the programs which used one or even two MPI processes per socket were significantly faster than those that utilized only one MPI process per node.

When using 128 cores on Stampede the run times are fastest when a total of two MPI processes are used per node and each process is allocated to its own CPU socket. The runtime using 128 cores was reduced from 77.8 s, when there was only one MPI process per node, to 46.1 s when using two MPI processes per node (about a 40% reduction in runtime). Even using four MPI process per node results in a significant improvement in performance as compared to one MPI process per node. The reason using one MPI process per socket, when using socket binding, gives better performance over using one MPI process per node is that one process per node will not be able to access the memory of all 16 cores on the node at the same speeds. The differences in memory access speeds are due to the two CPU sockets per node on Stampede. If a MPI process is located on one socket, then it must retrieve information stored on the other socket which is far slower than when retrieving information from its own socket. This results in a data retrieval bottleneck which slows performance and nearly eliminates the benefits of the speed gained by shared memory and fewer message passes. By allocating one MPI process per CPU a copy of all of the data that a process will need is stored on each socket. Therefore, all memory needed can quickly be accessed through shared memory in the same socket, resulting in much faster access, and faster run times. The same reasoning also works for two MPI processes per socket because all of the memory needed is located within one socket. However more than one MPI process per socket is slower because it must send more messages at the send/receive portions of the program.

Please cite this article in press as: Bettencourt, F.E., et al., Parallelization methods for efficient simulation of high dimensional population balance models of granulation. Computers and Chemical Engineering (2017), http://dx.doi.org/10.1016/j.compchemeng.2017.02.043

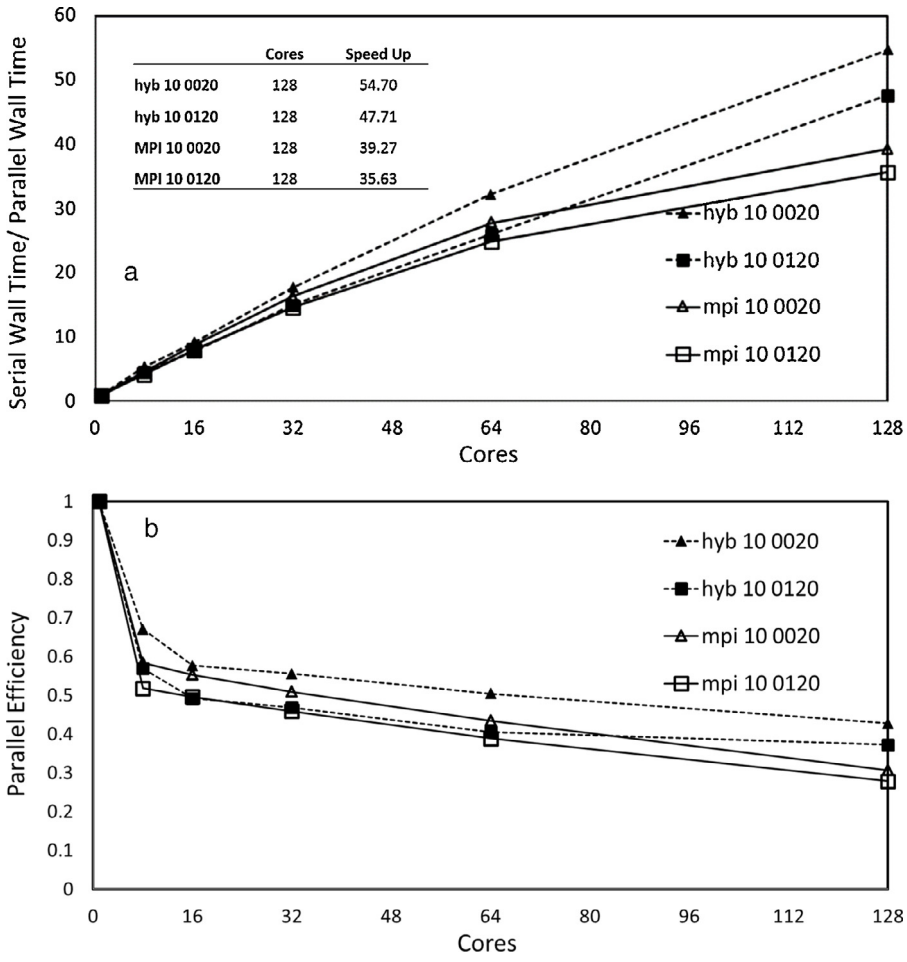

Fig. 8. (a) The scaling results obtained on Stampede for purely MPI jobs and for hybrid jobs when socket affinities were accounted for. Each job was compiled with the *–O3* option. The hybrid jobs were run using one MPI process per CPU and using the *tacc_affinity* option. The inset table shows the speed up obtained for each PBM when using 128 cores. (b) The resulting parallel efficiencies for the same jobs as in (a).

Table 4
Summary of results from PBMs run on the Stampede cluster.

Run type	Optimization	Bins	Granulation time	Cores	Wall time (s)	Hybrid wall time/MPI wall time	Speed up	Wall time/granulation time
Serial	*–O3*	10	20	1	401.32			20.07
MPI	*–O3*	10	20	128	10.22		39.27	0.51
Hybrid	*–O3*	10	20	128	7.34	0.72	54.70	0.37
Serial	*–O3*	10	120	1	2155.91			17.97
MPI	*–O3*	10	120	128	60.51		35.63	0.50
Hybrid	*–O3*	10	120	128	45.19	0.75	47.71	0.38

4.2.4. MPI vs hybrid MPI + OMP – speed up on stampede

Fig. 8a shows the scaling of hybrid and MPI jobs on the Stampede cluster. The hybrid jobs were run using the *tacc_affinity* option to account for socket affinities. For the hybrid jobs one MPI process was allocated to each CPU (2 MPI process per node). As can be seen the hybrid jobs perform far better than the MPI counter parts do when socket affinities are accounted for. The hybrid jobs ran 28% faster for the 20 second PBMs and 25% faster for the 120 second PBMs than the analogous MPI jobs, when running on 128 cores. The improved speeds when using one MPI process per CPU is due to the improved memory locality for each MPI process. From these results it appears that the hybrid method is faster, when one MPI process is used per socket, than a purely MPI approach. Fig. 8b shows the parallel efficiency as the number of cores in parallel is increased. The PBM program decreases in efficiency quickly from 1-8 cores but the decreasing efficiency slows once large core counts are reached. Part of the reason there is such a sharp initial drop in efficiency is due to how the model was set up. As mentioned earlier the master process does not calculate any of the rate processes which has an initial upfront cost but is better when using large core counts as the master process will not be bogged down by coordinating communication and calculation values for its own set of bins.

Table 4 shows the summary of results of the Stampede studies. The bins column lists the size of the solid, liquid, and gas, parameter bins (each was 10), and the granulation time column lists the simulated granulation time that the physical process would have been run for. This table also shows the serial compute time for each simulation run. For the hybrid jobs using 128 cores in parallel, and accounting for socket affinities reduced compute times from 401.32 s to 7.34 s and from 2155.91 s to 45.19 s for the 20 second granulation and 120 second granulation simulations respectively. Those decreased run times represent a 98.2% (100*(1–7.34/401.32)) and a 97.9% (100*(1–45.19/2155.91)) reduction in runtime for the 20 second and 120 second cases respectively. The reason the hybrid

Please cite this article in press as: Bettencourt, F.E., et al., Parallelization methods for efficient simulation of high dimensional population balance models of granulation. Computers and Chemical Engineering (2017), http://dx.doi.org/10.1016/j.compchemeng.2017.02.043

jobs outperform the MPI jobs on Stampede is because the hybrid jobs send fewer messages at each time step and the hybrid jobs use memory more efficiently resulting in more effective cache utilization. The last column shows the ratio of the wall clock compute time of the PBM to the time it would take for the real physical system to run in reality. It can be seen that by using these methods the simulations run in less than half the time it would take for a real process to run. This result suggests that using the methods of this work, even somewhat detailed models could be used for model predictive controllers since the models can be run faster than the real physical processes. For a detailed model like the ones in this work multiple jobs would have to be run simultaneously so that multiple implementations can be run to completion faster than the real process runs. Another option would be to reduce this model to a 4D PBM which would be about 5 times faster (reducing ny from ny = 5 to ny = 1 would be about a 5 fold decrease in computation) which would make the PBM around 10 times faster than the real granulation processes it is modeling.

5. Conclusions

In summary the spatial decomposition and parallelization of a five dimensional population balance model for studying wet granulation was successful in speeding up evaluation times. Using these methods simulation time was reduced by up to 98.2% using 128 cores. Further reductions in compute time could have been achieved if 241 cores were used (1 core for each bin in x). Given the differences in PBMs, hardware, and software used the 98.2% reduction in runtime is comparable to the 98.6% reduction achieved by Gunawan et al. (2008). The fastest times achieved on the SOEHPC were using purely MPI with the *–Ofast* option however socket affinities have not been studied on this system yet. The best speed up results achieved on the SOEHPC were for the *–O2* compiled PBMs. The fastest runtimes on Stampede were achieved using hybrid MPI + OpenMP, when socket affinities were accounted for using *tacc_affinity* option and one MPI process was used per socket. This finding is supported by the works of Adhianto and Chapman (2007), Mininni et al. (2011), and Rabenseifner et al. (2009). By finding that the hybrid MPI + OMP method was faster than a purely MPI method for solving PBMs we have confirmed that Gunawan et al. (2008) were correct when they asserted that shared memory methods for solving a parallel PBM could potentially improve performance. Furthermore the methods of this work could be combined with the load balancing techniques used by Gunawan et al. (2008) to potentially produce an even more efficient method for solving PBMs. Since load balancing was not used the methods of this work are somewhat easier to implement than the methods of Gunawan et al. (2008). The methods of this work also allow the programmer flexibility to tune the cluster scheduler settings and computational division methods to achieve the best possible performance for a wide range of cluster architectures. The methods in this work would still allow for tuning to cluster architectures if combined with the load balancing methods of Gunawan et al. (2008).

Based on the Stampede results the hybrid approach will outperform a purely MPI approach if the PBM is computationally intensive enough to merit the addition of the slightly larger overhead associated with the addition of OMP to the parallel program and socket affinities are accounted for. In most cases the PBM is sufficiently intensive enough that the use of hybrid methods will improve the performance as long as socket affinities are accounted for. From results comparing thread, core, and MPI process allocations on Stampede it can be seen that the cluster architecture has a huge impact on performance and models should be built keeping the architecture in mind to achieve the best performance. This finding is in agreement with the works of Adhianto and Chapman (2007), Mininni et al. (2011), and Rabenseifner et al. (2009). In this work it was also found that the hybrid method used significantly less memory than a purely MPI approach which is in agreement with the findings of many others who have studied hybrid methods (Jin et al., 2011; Gorobets et al., 2011; Wei and Yilmaz, 2011; Rabenseifner et al., 2009). In future work we will focus on more highly parallelized models, utilization of GPUs, and development of program features to further improve performance.

Funding

The authors would like to acknowledge funding from the Rutgers Aresty program and the National Science Foundation CAREER program, through grant number 1350152.

Acknowledgements

The authors would like to acknowledge funding from the Rutgers Aresty program. The authors acknowledge the assistances of Prof. Shantenu Jha for access to the Texas Advanced Computing Center's Stampede supercomputing cluster. The authors would like to thank Prentice Bisbal, former systems administrator for Rutgers Discovery Informatics Institute, for his assistance in implementing our programs on the clusters used for this study and for proof reading drafts of this manuscript. The authors would like to thank the system administrator of the SOEHPC cluster, Alexei Kotelnikov, for his help working with the cluster.

Appendix A. Rate processes and Kernels

A.1 Aggregation

Aggregation is when smaller particles come together to form a larger particle. For this work it is assumed aggregation occurs between only two particles at a time and is a function of particle number and composition. The net aggregation rate is shown below in Eq. (A1)

$$\Re_{agg}(s,l,g,t) = \Re_{form-agg}(s,l,g,t) - \Re_{dep-agg}(s,l,g,t) \tag{A1}$$

where $\Re_{agg}$ is the net rate of change in the particle number in a particular size class due to aggregation. $\Re_{form-agg}$ is the rate of formation due to aggregation and is calculated as shown in Eq. (A2) below.

$$\Re_{form-agg}(s,l,g,t) = \frac{1}{2}\int_0^s\int_0^l\int_0^g \beta(s-s',l-l',g-g',s',l',g') * F(s-s',l-l',g-g',t) * F(s',l',g',t)\,dg'\,dl'\,ds' \tag{A2}$$

$\Re_{dep-agg}$ is the rate of depletion of particles due to aggregation and is expressed in the following way in the flowing way:

$$\Re_{dep-agg}(s,l,g,t) = F(s,l,g,t) * \int_0^{smax}\int_0^{lmax}\int_0^{gmax} \beta(s,l,g,s',l',g') * F(s',l',g',t)\,dg'\,dl'\,ds' \tag{A3}$$

where β is the Madec kernel (Madec et al., 2003), expressed in the form shown in Eq. (A4).

$$\beta(s,l,g,s',l',g') = \beta_0(V+V') * \left((l+l')^{\alpha}\left(100 - \frac{l+l'}{2}\right)^{\delta}\right)^{\alpha} \tag{A4}$$

The Madec kernel, β, determines the probability of two particles aggregating based on their sizes and liquid contents. One particle is

represented by variables *s*, *l*, *g*, and *V*, while the other is represented by *s*′, *l*′, *g*′, and *V*′. The variable *V* is the volume of the particle, the term α, δ, and β_0 are adjustable parameters for tuning the kernel to experimental data (Madec et al., 2003). The variables *s*, *l*, and *g*, represent the content of solid, liquid, and gas, respectively, which compose the particle.

A.2 Breakage

Breakage is when a particle undergoes impact or attrition and fractures into smaller particles. For this work, it was assumed that one particle breaks into two particles at a time. Below is the equation to describe the breakage rate term as a function of formation and depletion.

$$\Re_{break}(s,l,g,t) = \Re_{form-break}(s,l,g,t) - \Re_{dep-break}(s,l,g,t) \tag{A5}$$

$$\Re_{form-break}(s,l,g,t) = \int_0^{smax}\int_0^{lmax}\int_0^{gmax} K_{break}(s',l',g') * b(s',l',g',s,l,g,t) * F(s',l',g',t)\,dg'\,dl'\,ds' \tag{A6}$$

$$\Re_{dep-break}(s,l,g,t) = K_{break}(s,l,g,t) * F(s,l,g,t) \tag{A7}$$

A uniform distribution was used for the breakage probability distribution, *b*. A uniform distribution means that the probability of forming every kind of particle in a breakage event is the same. K_{break} is a semi-empirical breakage kernel shown below in Eq. (A8) (Soos et al., 2006). The breakage kernel is a function of the particle radius (*R*), the shear rate *G*, and the tuning parameter *B*. Important to note, the breakage kernel used in this work is different from the Pandya-Spielman kernel (Pandya and Spielman, 1983) used by Dana et al. (2013b).

$$K_{break}(s,l,g,t) = \left(\frac{4}{15\pi}\right)^{1/2} G \exp\left(\frac{-B}{G^2 R}\right) \tag{A8}$$

$$G = \pi * v_{impeller} * D_{impeller} \tag{A9}$$

where $v_{impeller}$ and $D_{impeller}$ are respectively the rotational speed and diameter of the impeller in the granulator.

A.3 Liquid addition

In high shear wet granulation, liquid binder is added to increase the rate of aggregation. The rate of liquid additon is ususally kept fixed. The liquid addition rate process used for the model in this work is shwon below.

$$\frac{dl}{dt} = \frac{V_{spray}}{\int_{y1}^{y2}\int_{x1}^{x2}\int_0^{gmax}\int_0^{lmax}\int_0^{smax} F(s,l,g,x,y,t)\,ds\,dl\,dg\,dx\,dy} \tag{A10}$$

The rate, *dl*/*dt*, is the voluemteric liquid addtion per particle. For this work, it was assumed all particles absorb liquid binder at the same rate. The liquid spray rate is descirbed uisng a a liquid to solid ratio shown below.

A.4 Consolidation

Consolidation is when particles are compacted and the volume of gas in that particle is decreased. To describe the rate of consolidation in the model used for this work Eq. (A11) is used.

$$\frac{dg}{dt} = -c\frac{(s+1+g)(1-\varepsilon_{\min})}{s}\left[l - \frac{\varepsilon_{\min} * s}{1-\varepsilon_{\min}} + g\right] \tag{A11}$$

This equation is derived from Verkoeijen et al. (2002) and accounts for an exponential decrease of porosity where, $\in_{min}$ is the minimum granule porosity and *c* is the consolidation rate constant.

More details on the model development and specific kernels can be found in Barrasso et al. (2013b).

References

Adhianto, L., Chapman, B., 2007. Performance modeling of communication and computation in hybrid MPI and OpenMP applications. Simul. Model. Pract. Theory 15 (4), 481–491.
Barrasso, D., El Hagrasy, A., Litster, J., Ramachandran, R., 2015. Multi-dimensional population balance model development and validation for a twin screw granulation process. Powder Technol. 270 B, 612–621.
Barrasso, D., Oka, S., Muliadi, A., Litster, J., Wassgren, C., Ramachandran, R., 2013a. Population balance model validation and prediction of CQAs for continuous milling processes: toward QbD in pharmaceutical drug product manufacturing. J. Pharmaceut. Innovation 8 (3), 147–162.
Barrasso, D., Ramachandran, R., 2015. Multi-scale modeling of granulation processes: Bi- directional coupling of PBM with DEM via collision frequencies. Chem. Eng. Res. Design 93, 304–317.
Barrasso, D., Walia, S., Ramachandran, R., 2013b. Multi-component population balance modeling of continuous granulation processes: a parametric study and comparison with experimental trends. Powder Technol. 241, 85–97.
Chaudhury, A., Armenante, M.E., Ramachandran, R., 2015a. Compartment based population balance modeling of a high shear wet granulation process using data analytics. Chem. Eng. Res. Design 95, 211–228.
Chaudhury, A., Barrasso, D., Pandey, P., Wu, H., Ramachandran, R., 2014. Population balance model development, validation and prediction of CQAs of a high-shear wet granulation process: towards QbD in drug product pharmaceutical manufacturing. J. Pharmaceut. Innovation 9 (1), 53–64.
Chaudhury, A., Tamrakar, A., Schongut, M., Smrcka, D., Stepanek, F., Ramachandran, R., 2015b. Multi-dimensional population balance model development and validation of reactive granulation processes. Indus. Eng. Chem. Res. 54 (3), 842–857, 2015.
Christofides, P., Li, M., Madler, L., 2007. Control of particulate processes: recent results and future challenges. Powder Technol. 175 (1), 1–7.
Ganesan, S., Tobiska, L., 2012. An operator-splitting finite element method for the efficient parallel solution of multidimensional population balance systems. Chem. Eng. Sci. 69 (1), 59–68.
Gorobets, A., Trias, F., Borrell, R., Lehmkuhl, O., Oliva, A., 2011. Hybrid MPI + OpenMP parallelization of an FFT-based 3D Poisson solver with one periodic direction. Comput. Fluids 49 (1), 101–109.
Gunawan, R., Fusman, I., Braatz, R., 2008. Parallel high-resolution finite volume simulation of particulate processes. AIChE J. 54 (6), 1449–1458.
Jin, H., Jespersen, D., Mehrotra, P., Biswas, R., Huang, L., Chapman, B., 2011. High performance computing using MPI and OpenMP on multi-core parallel systems. Parallel Comput. 37 (9), 562–575.
Madec, L., Falk, L., Plasari, E., 2003. Modelling of the agglomeration in suspension process with multidimensional kernels. Powder Technol. 130 (13), 147–153.
Mininni, P., Rosenberg, D., Reddy, R., Pouquet, A., 2011. A hybrid MPI-OpenMP scheme for scalable parallel pseudospectral computations for fluid turbulence. Parallel Comput. 37 (6–7), 316–326.
Muzzio, F., Shinbrot, T., Glasser, B., 2002. Powder technology in the pharmaceutical industry: the need to catch up fast. Powder Technol. 124 (1–2), 1–7.
Prakash, A., Chaudhury, A., Ramachandran, R., 2013a. Parallel simulation of population balance model-based particulate processes using multi-core CPUs and GPUs. Model. Simul. 2013, 1–16.
Prakash, A., Chaudhury, A., Barrasso, D., Ramachandran, R., 2013b. Simulation of population balance model-based particulate processes via parallel and distributed computing. Chem. Eng. Res. Design 91 (7), 1259–1271.
Pandya, J.D., Spielman, L.A., 1983. Floc breakage in agitated suspensions: effect of agitation rate. Chem. Eng. Sci. 38, 1983–1992.
Rabenseifner, R., Hager, G., Jost, G., 2009. Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. Proceedings of The Parallel, Distributed and Network-Based Processing, 427–436.
Ramachandran, R., Barton, P., 2010. Effective parameter estimation within a multidimensional population balance model framework. Chem. Eng. Sci. 65, 4884–4893.
Ramakrishna, D., Singh, M.R., 2014. Population balance modeling: current status and future prospects. Ann. Rev. Chem. Biomol. Eng. 5, 123–146.
Sen, M., Chaudhury, A., Singh, R., Ramachandran, R., 2014. Two-dimensional population balance development and validation of a pharmaceutical crystallization process. Am. J. Modern Chem. Eng. 1, 3–29.
Seville, J., Tüzün, U., Clift, R., 1997. Processing of Particulate Solids, 1st ed. Springer, Netherlands.
Soos, M., Sefcik, J., Morbidelli, M., 2006. Investigation of aggregation, breakage, and restructuring kinetics of colloidal dispersions in turbulent flows by populations balance modeling and static light scattering. Chem. Eng. Sci. 61, 2349–2363.
Verkoeijen, D., Gerard, A., Gabriel, P., Meesters, M.H., Scarlett, B., 2002. Population balances for particulate processes—a volume approach. Chem. Eng. Sci. 57 (12), 2287–2303.

Please cite this article in press as: Bettencourt, F.E., et al., Parallelization methods for efficient simulation of high dimensional population balance models of granulation. Computers and Chemical Engineering (2017), http://dx.doi.org/10.1016/j.compchemeng.2017.02.043

Wei, F., Yilmaz, A.E., 2011. A hybrid message passing/shared memory parallelization of the adaptive integral method for multi-core clusters. Parallel Comput. 37, 279–301.

Wilkinson, B., Allen, M., 1997. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, 1st ed. Prentice Hall, Lebanon, IN, USA.

Please cite this article in press as: Bettencourt, F.E., et al., Parallelization methods for efficient simulation of high dimensional population balance models of granulation. Computers and Chemical Engineering (2017), http://dx.doi.org/10.1016/j.compchemeng.2017.02.043