Accelerating multi-dimensional population balance model simulations with a highly scalable framework on GPUs.

Chaitanya Sampat[a], Yukteshwar Baranwal[a], Rohit Ramachandran[a,*]

[a] *Chemical and Biochemical Engineering, Rutgers University, Piscataway, NJ, USA - 08854*

Abstract

Population Balance models (PBMs) are widely used to simulate and optimize the distributed particulate process. Prediction of time evolution of the distribution of the particulate properties using these PBMs, is very important in understanding key dynamics. However, with an increase in the number of components and spatial configurations the process, the complexity of a PBM increases. This leads to multi-dimensional matrix calculations potentially requiring significant computational power. Solving such a system of equations with a traditional central processing unit (CPU) would be computationally intractable. This study focuses on the development of a scalable algorithm to parallelize the nested loops inside the PBM via a GPU framework. The developed PBM was unique since it adapts to the size of the problem and uses the GPU cores accordingly. This algorithm was parallelized specifically for NVIDIA GPUs as it was written in CUDA C/C++. The major bottleneck to such algorithms is communication time between the CPU and the GPU, which in our studies contributed to less than 1% of the total run time leading to a maximum speedup of about 12 over the serial code being achieved on the GPU. The speed up reported for the GPU PBM was about 2x the PBM code on multi-core configuration on a desktop computer. The speed improvements are also reported for various CPU & GPU architectures and configurations.

[*] Corresponding author
Email address: `rohitrr@soemail.rutgers.com` (Rohit Ramachandran)

Keywords: Population Balance Model, GPU, Parallel Computing, Granulation, MPI, CUDA

1. Introduction

Over the past decade, Population Balance Models (PBMs) have been used widely to predict dynamics of distributed processes (Ramkrishna and Singh, 2014). The population balance equation is essentially a number conservation of particles (entities) which regulate the behavior of the entire system. PBMs consist of several external (i.e spatial) and internal coordinates and with an increase in grids of these coordinates it can lead to a more accurate model. With The increase in the number of grid of these coordinates, it leads to an increase in calculations for each time step, leading to a higher simulation times. The calculations increase by a factor of n^4 where n being the number of entity grids. PBMs describe the evolution of entities into different states and into newer entities. Deriving accurate equations to determine these rate of change of internal coordinates requires precise empirical correlations from experimental data. Another way to increase the accuracy of these models is to introduce a first-principle based kernel into these models (Barrasso and Ramachandran, 2015). An accurate model which incorporates higher number of grids as well as includes a mechanistic kernel in its calculations is expected to be sluggish to simulate and could take upto an hour (Barrasso et al., 2015) to complete. Such models and their solving techniques are not viable to be used in real time system control. Thus, there is a need to improve the time it takes to simulate the model.

The advancement of computers and its components in recent years have led to a great increase in computational resources leading to faster simulations. The recent central processing unit (CPU) now contain various cores thus making it possible to run multiple processes in parallel. In order to take advantage of a highly parallel framework, large number of cores are required which may not be possible in a personal desktop and a supercomputer cluster would be needed.

Another computer component that can to be used to run a highly parallel code is the computer's graphic processing unit (GPU) (Prakash et al., 2013). These GPUs contain thousands of compute cores that can be used run tasks in parallel. Thus, a desktop equipped with a GPU could compute the same results as a CPU code on supercomputers in lesser amount of time as seen in Section 4. With the launch of Compute Unified Device Architecture (CUDA), NVIDIA made it easier to use GPUs for general parallel programming in an approach usually termed as general purpose computing on GPUs (GPGPUs).

Various chemical industries such as detergent, food, pharmaceutical, fertilizers, catalyst encounter particulate processing daily. These processes constitute to about 50% of the world's chemical production (Seville et al., 2012). In the pharmaceutical industry, particulate processes are widely used to increase the size of the granules, improve flow-ability, increase yield strength etc. One of major processes is granulation, in which fine pharmaceutical powder blends are converted to larger granules using a liquid or a dry binder (Chaturbedi et al., 2017). These larger granules help in better flow ability and strength to these mixtures aiding further processing.

The main objective of this work is to develop a highly scalable framework for to simulate PBMs on GPUs. This framework would be independent of the type of kernels as well as number of spatial compartments as well as the number of solid bins used. The framework would be universal and would be able to run of different NVIDIA GPUs without the need for any changes. The number of threds that the PBM will execute depends upon the size of the problem as well as the number of cores available on the GPU. This code was developed in NVIDIA CUDA C/C++. A similar code was also developed on C++ to be run on the CPU which has limited scalability due to number of CPU cores available on desktop computer. This work also enables the use of desktop computers to obtain numerical solutions to computationally intensive tasks rather than relying on supercomputers for quicker results.

2. Background and previous works

2.1. Population balance modeling and Granulation

Population balances equations have been successful in predicting physical phenomena occurring in granulation such as aggregation, breakage and consolidation. These models predict how groups of distinct entities inside the pharmaceutical powder behave on a bulk scale due to process parameters over time of granulation. A general representation of the model is:

$$\frac{\partial}{\partial t}F(\mathbf{v},\mathbf{x},t)+\frac{\partial}{\partial \mathbf{v}}[F(\mathbf{v},\mathbf{x},t)\frac{d\mathbf{v}}{dt}(\mathbf{v},\mathbf{x},t)]+\frac{\partial}{\partial \mathbf{x}}[F(\mathbf{v},\mathbf{x},t)\frac{d\mathbf{x}}{dt}(\mathbf{v},\mathbf{x},t)]$$
$$=\Re_{formation}(\mathbf{v},\mathbf{x},t)+\Re_{depletion}(\mathbf{v},\mathbf{x},t)+\dot{F}_{in}(\mathbf{v},\mathbf{x},t)-\dot{F}_{out}(\mathbf{v},\mathbf{x},t) \quad (1)$$

where $\mathbf{v}$ is a vector of internal coordinates. $\mathbf{v}$ is commonly used to describe the solid, liquid, and gas content of each type of particle. The vector $\mathbf{x}$ represents external coordinates, usually spatial variance. F represents the number of particles present inside the system, $\dot{F}_{in}$ and $\dot{F}_{out}$ is the rate of particles coming in and going out the system respectively. $\Re_{formation}$ and $\Re_{depletion}$ are the rate of formation and depletion due various phenomena. $\Re_{formation}$ described in 5 is the most computationally intensive component of the PBM. This calculation consists of a double integral for each entity being tracked. In a system tracking 2 particulates, it would require 4 nested loops to compute these integrals. Figure 5 also indicates that this calculation takes 50% of the total run time. Thus, parallelizing this calculation if crucial to the effectiveness of the parallel algorithm.

Granulation is the process of engineering granules from pharmaceutical powder blends with the addition of liquid or solid binders. This process is usually carried out to obtain granules with a certain PSD, bulk densities and other physical properties (Barrasso and Ramachandran, 2015). There are about 3 rate processes that occur due the addition of a liquid binder to the powder mixture are wetting and nucleation, consolidation and aggregation, and breakage and attrition (Sen et al., 2014). One way to the perform granulation of pharmaceutical powders is to use a high shear granulator. In a high shear granulator, the particles are considered to be wet when they come in contact with the liquid binder, which also aids in granule formation due to liquid bridges. These granules can also break into smaller fragments due to shear stresses, compressive and tensile forces that are exerted on to the system due the impeller, particle-particle interactions and particle-wall interactions.

2.2. Parallel Computing

Computing with its current infrastructure has become an important tool in science. Parallel computing is one of the more extensively used type of computing used by scientists to perform simulations. It is the process of splitting of larger calculations into many smaller processes executed concurrently (Almasi and Gottlieb, 1989). This type of execution helps achieve large speed gains overs simulations run on a single core in a serial manner. The computational task can be decomposed by various means to help simulate the system in a reasonable amount of time. The simulation problem can be decomposed either at the task level or at the data level. Task parallelism involves each process to behave distinctively from another as they would each be performing different operations. These multiple operations could be performed on a single data set or on multiple data sets, known as multiple instruction single data (MISD) and multiple instruction multiple data (MIMD) respectively. On the other hand, data parallelism involves the distribution of data across various processes which usually perform same set of operations on the data (Solihin, 2015). This type of parallelism is known as single instruction multiple data (SIMD). MIMD and SIMD can also be combined in certain systems, thus decreasing the simulation times further.

2.3. GPU based parallel computing

Traditionally, large parallel jobs needed to be run on supercomputers which had thousands of cores, but these are require special components making them expensive. Graphic processing unit (GPU) were initially used for vector calculations to support graphics inside a computer system. But, lately GPU manufacturers have started to promote them general computing as well. This form of computing has been gaining popularity among scientists to accelerate simulations (Kandrot and Sanders, 2011). These GPUs comprise of a massively parallel architecture with hundreds to thousands of computational cores which can have thousands of active threads running simultaneously (Keckler et al., 2011). This means that GPUs have large cWomputing potential the GPU computing can be exploited using parallel programming languages such as OpenCL and CUDA.

CUDA is an application programming interface (API) developed by NVIDIA (NVIDIA Corporation, 2012) that enables users to program parallel code for execution on the GPU. This framework is an extension implemented on top of C/C++ or Fortran. Parallel code for the GPU is written as kernels, which theoretically are similar to functions or methods in traditional programming languages. As several parts of the code need to executed only once during a simulation, only few sections of the code can be written in terms of kernel while the remaining has to be executed in serial on the CPU of the system. The `nvcc` compiler from the CUDA toolkit prioritizes the compilation of these kernels before passing the serial section of the code to the native C/C++ compiler inside the system. There are three main parallel abstractions that exist in CUDA are grids, blocks and threads (Santos et al., 2013). Each CUDA kernel is executed in a serial manner during the execution of the program unless specified, where the kernels can be run in parallel using CUDA streams. Each kernel executes as a grid which in turn consists of various blocks which are consistuted by various threads. This thread-block-grid hierarchy helps obtain fine grained data level and thread level parallelism. An illustration of this hierarchy is observed in Figure 1.

Another important aspect related to GPU parallelization is the data communication between the threads. The GPU consists of various memory modules with different access limitations as shown in Figure 1. The threads inside each block can communicate with each other using the shared memory. This memory is local to the block where these threads exist i.e. they are not accessible by threads from other blocks. In addition to the shared memory each thread has its own local memory where local/temporary variables for each kernel can be saved to them. The threads from different communicate with each other using the global memory which is visible to all blocks inside the GPU at the cost of higher communication times. Accessing data from the local memory is the fastest for a thread and its slows down when the thread needs to retrieve data from the shared memory inside a block and accessing data is the slowest from the global memory of the GPU.

2.4. Previous parallelized PBM works

PBMs have been computationally intensive notably ones with larger number of internal coordinates and higher dimensions. Thus, several researchers have made attempts to increase the speed of these simulations. Gunawan et al. (2008) developed a parallelization technique using a high-resolution finite volume solution of the PBM. The studies carried out by Gunawan et al. (2008) for their parallel PBM algorithm achieved good parallel efficiency upto 100 cores. This study was limited by the number of grids used for the PBM,which meant the problem size was not computationally very heavy. This study also suggested that an algorithm with a shared memory model could help improve simulation speeds further. A hybrid memory model which uses both shared and local memory was implemented by Bettencourt et al. (2017) to obtain speed improvements of about 98% from the serial code. This implementation took into account both Message Parsing Interface (MPI) as well as Open Multi-processing (OMP). A similar PBM parallelization approach was also undertaken in Sampat et al. (2018) and an approximate speedup of 13 w;as obtained. The reduction in speed was attributed to the use of dynamic arrays used in their PBM framework to accommodate the hybrid nature of the model being used.

Algorithms to parallelize the PBM codes on GPU have been studied briefly by Prakash et al. (2013) using the inbuilt MATLAB's parallel computing toolbox

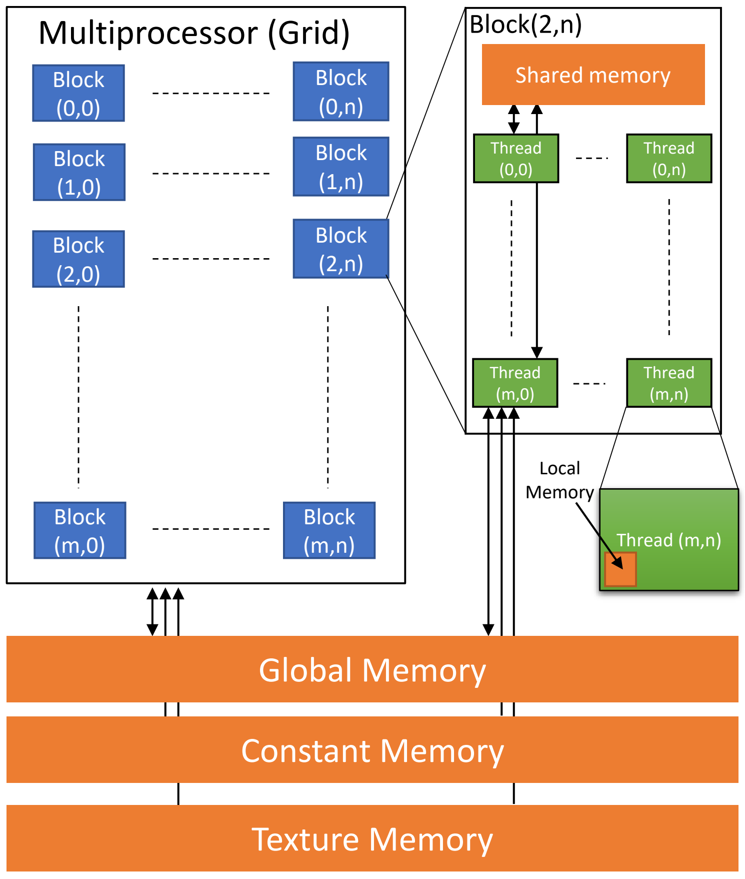

Figure 1: The parallel structure matrix inside the GPU and the various memories associated with each structure.

(PCT). This study was able to achieve good speedup values but could have been higher if the code had been implemented in native programming languages such as C or FORTRAN. Since MATLAB is a high level language it internally converts the written code to native programming languages before it is sent to the processor leading to excess computation which can be avoided (MathWorks$^{\text{TM}}$ Documentation, 2017b). Other works that have used GPU acceleration to improve computation times for their population balance simulations include those from various other chemical engineering processes such as crystallization (Szilgyi and Nagy, 2016), combustion (Shi et al., 2012) , multiphase flow (Santos et al., 2013), coagulation dynamics (Xu et al., 2015). Several of these works fail to address the problem size as a factor in determining number of GPU cores used for the simulation. These algorithms are restricted to a maximum number of cores that are used and with an increase in the problem size, the algorithm would not adapt to increase the number of cores used. This could potentially lead to computation power that may go unused in high-end GPUs that have the capacity to perform several teraflops of double-precision calculations/.

3. Method and implementation

3.1. PBM implementation

The overall population balance equation used in our algorithm looked like (Ramkrishna and Singh, 2014):

$$\begin{aligned}\frac{d}{dt}F(s_i,l,g,x,t) &= \Re_{agg}(s_i,l,g,x,t) + \Re_{break}(s_i,l,g,x,t)\\ &\quad + \dot{F}_{in}(s_i,l,g,x,t) - \dot{F}_{out}(s_i,l,g,x,t) \qquad (2)\end{aligned}$$

where, $F(s_i,x)$ represents the number of solid particles of type i being studied in each spatial compartment x of the granulator. The rate of aggregation $\Re_{agg}(s_i,x)$ and the rate of breakage $\Re_{break}(s_i,x)$ determines the rate at which particles density changes within different size classes. The rate of particles entering, $\dot{F}_{in}(s_i,x)$ and exiting, $\dot{F}_{out}(s_i,x)$, the spatial compartment due to particle transfer also affects their number in each size class. The rate of change of internal liquid volume in each particle can be calculated as:

$$\begin{aligned}\frac{d}{dt}F(s_i,x)l(s_i,x) &= \Re_{liq,agg}(s_i,x) + \Re_{liq,break}(s_i,x) + \dot{F}_{in}(s_i,x)l_{in}(s_i,x)\\ &\quad - \dot{F}_{out}(s_i,x)l_{out}(s_i,x) + F(s_i,x)\dot{l}_{add}(s_i,x) \qquad (3)\end{aligned}$$

where, $l(s_i,x)$ is the internal liquid volume in each particle with s_i as the solid volume for solid type 1 in the spatial compartment x. $\Re_{liq,agg}(s_i,x)$ and $\Re_{liq,break}(s_i,x)$ are the rates at which liquid is transferred between size classes due to aggregation and breakage respectively. $l_{in}(s_i,x)$ and $l_{out}(s_i,x)$ are the