Graphical Processing Unit (GPU) Acceleration for Numerical Solution of Population Balance Models Using High Resolution Finite Volume Algorithm

Botond Szilágyi[1], Zoltán K. Nagy[2,3,*]

[1]*Department of Chemical Engineering, "Babes-Bolyai" University, Arany Janos Street 1, Cluj Napoca 400028, Romania,*

[2]*Department of Chemical Engineering, Loughborough University, Leichestershire, Loughborough Le11 3TU, United Kingdom,*

[3]*School of Chemical Engineering, Purdue University, West Lafayette 47907-2100, USA*

**zknagy@purdue.edu, z.k.nagy@lboro.ac.uk*

Abstract

Population balance modeling is a widely used approach to describe crystallization processes. It can be extended to multivariate cases where more internal coordinates i.e. particle properties such as multiple characteristic sizes, composition, purity, etc. can be used. The current study presents highly efficient fully discretized parallel implementation of the high resolution finite volume technique implemented on graphical processing units (GPUs) for the solution of single- and multi-dimensional population balance models (PBMs). The proposed GPU-PBM is implemented using CUDA C++ code for GPU calculations and provides a generic Matlab interface for easy application for scientific computing. The case studies demonstrate that the code running on the GPU is between 2…40 times faster than the compiled C++ code and 50…250 times faster than the standard MatLab implementation. This significant improvement in computational time enables the application of model-based control approaches in real time even in case of multidimensional population balance models.

Keywords: population balance modelling, finite volume algorithm, GPU, crystallization modelling

Highlights:

High resolution finite volume method solution of population balances using GPU acceleration

Significant speed improvement achieved with the used low - cost GPUs

Development of a computationally highly efficient generic crystallization simulation tool

1. Introduction

The population balance (PB) modelling framework was introduced by Hulburt and Katz (1964) to describe the population dynamics and thence it have been used in numerous fields of science including meteorology, biology, physics, chemistry as well as in different aspects of engineering. In crystallization science and technology it is the trivial modelling approach enabling the description of particulate properties such as particle size, shape, age, composition *etc*. A huge variety of experimental and theoretical works appeared discussing its different aspects.

From mathematical point of view, the PB equation (PBE) is a partial differential equation which may involve integral terms if secondary processes like breakage and agglomeration are included. A variety of solution methods were proposed, each having their advantages and disadvantages. The method of moments was introduced by Randolph and Larson (1971) and it is based on reduction of the original PBE. The closure problem of the generated moment equation system significantly reduces its practical applicability. However some closure techniques exist like the cumulant neglect method (Lakatos, 2008) or the interpolative closure (Frenklach, 2002), the generic solution is the quadrature method of moments (McGraw, 1997).

The quadrature based moment methods are computationally effective, accurate and can be applied even on the most complicated PBE-s but, it computes only averaged particulate properties.

As the modern measuring devices have become able to record on-line distributional data (Nagy et al, 2013), a high demand appeared to develop new solution methods to compute the particle size distribution (PSD) instead of some statistics based averages. Numerous methods exists to restore the PSD from the moments, including the approximated density functions with moment dependent parameters (Randolph and Larson, 1988) or linear and nonlinear inversions (Aamir, 2010) and were successfully applied in various studies (Szilágyi et al, 2015), although mathematically none of those is exact. The combined quadrature method of moments – method of characteristics was a successful technique to solve full 1D PBE-s with nucleation and growth only (Aamir et al, 2009, Aamir et al, 2010). The Monte Carlo simulations were successfully applied to solve PBE-s as well (Bárkányi et al., 2013; Irizarry, 2008; Smith and Matsoukas, 1988) but the computational costs are far too increased to be applicable in the majority of engineering problems. The method of classes is based on the discretization of internal variables but for the more complicated PBEs a large number of classes is required to maintain the accuracy (Valentas and Amundson, 1966). The method of weighted residual/orthogonal collocation with finite element discretization was also successfully applied to solve PBE-s and seems to be an attractive alternative (Ulbert and Lakatos, 2007), however the finite volume based methods presents increased accuracy especially near to sharp variations.

The high resolution finite volume method (HR-FVM) was developed to solve the hyperbolic partial differential equation (LeVeque, 2002) and was adapted for the solution of PBEs by Gunawan et al. (2004). The HR-FVM is able to solve numerically the PBEs even with agglomeration and breakage, moreover, computes the PSD without significant numerical

diffusion and dispersion. The FVM solution is based on the discretization of the PSD, presented in Figure 1a and Figure 1b. For more for details about these methods, see the work of Qamar et al. (2006).

Unfortunately the computational costs of the HR-FVM may become large compared to the moment based methods especially when using finer meshes. Basically, in order to allow on-line process optimization and control, it is crucial to have an adequate model, which is solved accurately with orders of magnitude faster than the real process time. Controlling the particulate properties like particle size distribution (PSD) and particle shape is essential as it can affect significantly the product quality (specific surface, porosity, dissolution rate *etc.*) and downstream operations (filtration, granulation, milling *etc.*). Thus there is strong need to accelerate FVM based PBE solution for process optimization and real-time model based control.

A several attempts were made to improve the computational efficiency of the HR-FVM. Qamar et al. (2007) presented an adaptive mesh strategy making possible to reduce the mesh size maintaining the accuracy. Gunawan et al. (2008) proposed parallelized solution using a master/slave structured CPU cluster. Majumder et al. (2010) developed the Fast HR-FVM method which uses a coordinate transformation to speed up the simulation by maintaining its accuracy. Prakash et al. (2013) exploited the Matlab Parallel Computing Toolbox and Distributed Computing Server capabilities to parallelize the HR-FVM codes on CPUs. In the above presented methods significant speed up was achieved but the increased price of the used supercomputers, from industrial point of view, limit the applicability of these approaches. Also in the case of real time control in an industrial setup, would be difficult to set up a control system that implements real time model solution on remote supercomputers.

Due to their massively parallel hardware architecture, GPUs have been used for accelerating scientific calculations (Shane Cook, 2012). In the field of crystallization several works were published discussing mainly the GPU acceleration of Monte-Carlo methods. Wei and Kruis (2013) presented a Monte Carlo simulation for particle coagulation problem using an acceptance-rejection method. Wei (2014) published a parallel Monte Carlo method using a bookkeeping strategy and Xu et al. (2015) applied Markov jump model to simulate the coagulation dynamics. Out of the Monte Carlo methods, Santos et al. (2013) presented the GPU accelerated dual quadrature method of generalized moments to solve PBEs.

Despite of its advantages, GPUs were not used yet in HR-FVM. The aim of this work is to apply GPU acceleration for HR-FVM PBE solution using a low-cost device as well as to analyze the performance of codes in order to find reasonable trade-off between the accuracy and computational costs, to provide a framework for the solution of single or multidimensional PBEs suitable for model-based optimization and real time control.

2. Population balance models and the HR-FVM algorithm

In this section a brief overview of the HR-FVM is provided that was implemented on the parallel GPU system. Figure 1a. presents the finite volume discretization of a continuous 1D size density function and Figure 1b. the analogue 2D case. Note that the main contribution of this work is the very efficient solution of multidimensional PBEs, which are computationally much more demanding than 1D PBEs, so we will focus on presentation of the 2D HR-FVM. A detailed description of the 1D HR-FVM was provide by Gunawan et al. (2004).

Let us denote with h the size interval and with k the time interval. Then $n_{l,w}^{m}$ is an approximation of the average population density:

$$n_{l,w}^{m} \approx \frac{1}{h^2} \int_{(l-1)h}^{l\,h} \int_{(w-1)h}^{w\,h} n(l, w, m\,k)\, dwdl \tag{1}$$

Where *m*, *l* and *w* are integers such that $m \geq 0$ and $N \geq \mathrm{l}, \mathrm{w} \geq 1$. *N* denotes the mesh size (i.e. the number of discretization points) along an internal coordinate. In the equation, $n_{l,w}^{m}$ gives the number of crystals being in the *(l-1)·h,l·h* and *(w-1)·h,w·h* discrete size domain – or in (*l*,*w*) grid cell in m^{th} discrete time moment.

The general 2D population balance equation with nucleation and growth take the form:

$$\frac{\partial n(L_1, L_2, t)}{\partial t} + \frac{\partial G_1(L_1) n(L_1, L_2, t)}{\partial L_1} + \frac{\partial G_2(L_2) n(L_1, L_2, t)}{\partial L_2} = B\delta(L_1 - L_{1n})(L_2 - L_{2n}) \tag{2}$$

with the initial condition:

$$n(L_1, L_2, 0) = n_0(L_1, L_2) \tag{3}$$

B is the nucleation rate while $G_1(L_1)$ and $G_2(L_2)$ stands for the growth rates along the axes and $n(L_1, L_2, t)$denotes the bivariate size density function in *t* time moment. LeVeque (2002) presented a high resolution method for such hyperbolic system, where the growth rates are evaluated at the endpoints of each grid cell. This is a formal second-order accurate method. According to the algorithm, the $n_{l,w}^{m+1}$ is computed as:

$$n_{l,w}^{m+1} = n_{l,w}^{m} + \mathbf{L} + \mathbf{W} + (l-1)^0 (w-1)^0 \frac{k}{h^2} B \tag{4}$$

L and **W** are the operators governing the number variation caused by length and width growths but the last term is the nucleation which exist if and only if *l = w = 0*. So it influences the number of smallest crystals. The **L** and **W** operators are of the forms:

$$\mathbf{L} = -\frac{k}{h}\left(G_{1,l} n_{l,w}^{m} - G_{1,l-1} n_{l-1,w}^{m}\right) - \left[\frac{kG_{1,l}}{2h}\left(1 - \frac{kG_{1,l}}{h}\right)\left(n_{l+1,w}^{m} - n_{l,w}^{m}\right)\phi_{1,l} - \frac{kG_{1,l-1}}{2h}\left(1 - \frac{kG_{1,l-1}}{h}\right)\left(n_{l,w}^{m} - n_{l-1,w}^{m}\right)\phi_{1,l-1}\right] \tag{4a}$$

$$\mathbf{W} = -\frac{k}{h}\left(G_{2,w} n_{l,w}^{m} - G_{2,w-1} n_{l,w-1}^{m}\right) - \left[\frac{kG_{2,w}}{2h}\left(1 - \frac{kG_{2,w}}{h}\right)\left(n_{l,w+1}^{m} - n_{l,w}^{m}\right)\phi_{2,w}\right.$$

$$\left. - \frac{kG_{2,w-1}}{2h}\left(1 - \frac{kG_{2,w-1}}{h}\right)\left(n_{l,w}^{m} - n_{l,w-1}^{m}\right)\phi_{2,w-1}\right]$$

Where *l* and *w* corresponds to the mesh element thus the calculations are repeated for ***each*** mesh size in ***every*** time moment.

The physical meaning of Eq.(4) is the follows. In Figure 1c the *(l,w)* cell is highlighted. The number of crystals being within this cell in the m^{th} time moment is $n_{l,w}^{m}$. For the next time moment, assuming crystal growth, a certain number of crystals are "coming" to this cell from *(l-1,w)* as a result of length growth of crystals whose width is *w* but are shorter with one discrete size bin. The number in *(l,w)* cell increases from *(l,w-1)* neighborhood cell as the result of width growth of crystals whose width is *l* but are narrower with one discrete size bin. Naturally, due to the same crystal growth process, some crystals "grow out" from this cell by length growth to *(l+1,w)* and width growth to the *(l,w+1)*.

In Eq.(4a) $\phi_{1,l} = f(\theta_{1,l})$ and $\phi_{2,w} = f(\theta_{2,w})$ denotes the flux limiter functions which depends on the degree of smoothness of the distribution which is expressed as a ratio of two consecutive gradients:

$$\theta_{1,l} = \frac{n_{l,w}^{m} - n_{l-1,w}^{m}}{n_{l+1,w}^{m} - n_{l,w}^{m}}$$

$$\theta_{2,w} = \frac{n_{l,w}^{m} - n_{l,w-1}^{m}}{n_{l,w+1}^{m} - n_{l,w}^{m}} \qquad (5)$$

In the smooth regions the following conditions ensure the second order accuracy:

1. ϕ is Lipshitz continuous at θ = 1 and $\phi(\theta)$ is bounded with $\phi(\theta) = 1$
2. $0 \le \frac{\phi(\theta_l)}{\theta_l} \le 2$ (6)
3. $0 \le \phi(\theta_l) \le 2$

A huge variety of flux limiter functions have been proposed and each of them leads to different high resolution method. The Van Leer flux limiter has been successfully applied in the simulation of population balance equations and provides full second order accuracy. This flux limiter function has the general form

$$\phi(\theta) = \frac{|\theta| + \theta}{1 + |\theta|} \tag{7}$$

Note that Eqs.(4)-(7) presents the two dimensional formulation of the HR-FVM. Multiple dimensional cases can be simulated by the means of dimension splitting too which is a straightforward and simple extension of 1D HR-FVM to multiple dimenisons (LeVeque 2002).

To close the model – so compute the, usually, concentration dependent nucleation and growth rates the macroscopic mass balance equation is required.

$$\frac{dc}{dt} = -k_V \rho_c \left(\int_0^\infty \int_0^\infty G_1(L_1) L_2^2 n(L_1, L_2, t) dL_1 dL_2 + 2 \int_0^\infty \int_0^\infty G_2(L_2) L_1 L_2 n(L_1, L_2, t) dL_1 dL_2 \right) \tag{8}$$

with the $c(0) = c_0$ initial condition. k_V denotes the volume shape factor and ρ_c stands for the crystal density. The mass balance also should be discretized in time and solved simultaneously with Eq.(4) system:

$$c^{m+1} = c^m - k_v \rho_C h^2 \left(\sum_{l=1}^{N} \sum_{w=1}^{N} G_{1,l} l^2 n_{l,w}^m + 2 \sum_{l=1}^{N} \sum_{w=1}^{N} G_{2,w} w^2 n_{l,w}^m \right) \tag{9}$$

Note that this is a fully discretized HR-FVM algorithm, which means that both the spatial coordinates and the time are discretized. The applied time step either is fixed or is adaptively recalculated in every iteration. Nevertheless, an algebraic equation system (AES) Eq.(4) and Eq.(9) is solved which is used to reconstruct the original PSD. Semi-discrete formulations of the HR-FVM have also been proposed. These methods adopt the discretization of particle size and for each resulted size bin a differential equation is formulated. Consequently the semi-discrete solution implies the solution of an ordinary differential equation (ODE) system which

is convenient to solve, for instance, with the MatLab's ODE solvers. In contrast, in custom codes the fully discretized algorithm may be more advantageous due to the it's simpler implementation.

The upper mentioned algorithm is used in simulations in three compute implementations:

a) in the form of Matlab function,
b) C++ code as compiled .mex function called from the Matlab running on the CPU; and
c) parallel CUDA C++ code in form of compiled .mex function called from Matlab, running on GPU *and* CPU.

It should be noted that when developing the programs, the capability of the tool to be embedded in Matlab was a key factor as it is the generally used environment by the process engineering community. The most important properties of the computers are listed in Table 1. Low cost devices are used with optimizing compilers which boosts up the performance of compiled codes. In the following parts, these implementation strategies will be discussed briefly.

2.1. Implementation as Matlab function

Matlab provides flexible engineering tools with a series of additional toolboxes and is widely used in engineering practice. However, the computational performance compared to the high level programming languages is reduced even if the codes are optimized (variable pre-allocation, vectorization, optimal entering sequence of the matrix elements – column-wise/row-wise) which makes it a less efficient environment for running codes involving increased computational demand. In this study, the optimized Matlab implementation is the basis to which the other implementations are compared.

2.2.Implementation in C++ code

C++ is known as a very fast high level programming language thus it may be a good environment to implement the HR-FVM. In order to simultaneously profit from the flexibility of Matlab and improved speed of C++, the HR-FVM algorithm is written in C++ code and compiled to *.mex* function. The *.mex* file, in essence, is a special *.dll* which can be called directly from Matlab. The *Gateway function* of the .mex makes the connection between the Matlab and the crude C++ code (see Figure 2). In this case the memory management is handled by the programmer providing extra flexibility when optimizing the code. The operations are still executed serially, which, especially for finer meshes, can significantly slow down the solution. Parallelizing the code can improve the speed but it may overload the CPU thus is not applied in this work.

2.3. Implementation in CUDA C++ code

More recently there has been an increased interest to apply parallel computing and computations using GPUs, which typically has a hardware architecture consisting of multiple parallel computing units. The HR-FVM algorithm presents high potential for parallelization according to Eqs.(4) which shows that the in each time step similar calculations are required for every grid cell. Based on the algorithm description the calculation of $(t+k)$ time moment depend only on the data at time t thus these equations can be solved in parallel. In this point the natural questions rises if the GPU could accelerate the simulation.

The Matlab's Parallel Computing Toolbox (PCT) offers three implementation ways to run code on GPU: (i) run built-in Matlab function, (ii) run element-wise Matlab code and (iii) run *.ptx* code as parallel CUDA Kernel object. The *.ptx* code offers the highest flexibility and computing performance, which enables for the programmer maximal control of data flow in the CUDA cores. The memory management in all cases is handled by Matlab, which reduces

the code-optimization possibilities. In addition, these implementations require the PCT, resulting additional cost requirements.

Another way to apply GPU calculations in Matlab is via CUDA containing *.mex* function. This has a structure of a conventional *.mex* function, in the sense that it has a gateway function, which may call not only the serial functions (running on the CPU) but also the parallel routines (running on GPU). The CUDA code of the parallel *.mex* function is exactly what is required by the PCT CUDA Kernel object. As long as the parallel *.mex* function may contain CUDA C++ and C++ parts also, its compilation requires both the parallel and serial compilers. These compilers work according to the simplified scheme in Figure 3; first the parallel (GPU - "Device") code is compiled using the nVidia CUDA compiler, which is passed to the C++ compiler (in this case MS Visual Studio 2010 C++ compiler) creating, together with the serial parts of the code the final *.mex* file.

In order to maximally explore the capacities of the CPU and GPU, a hybrid calculation strategy is applied in which only the parallel parts of the code are executed on the GPU. In the HR-FVM the flux-limiter function, size dependent growth rate, HR-FVM algorithm and the integral calculations are parallelizable. The serial calculations, in which the considerably slower GPU cores present poor performance, are executed on the CPU. These include the mass balance, temperature, supersaturation, growth and nucleation rate calculation as well as the adaptive time stepping. As long as the GPU device has separate memory unit, the necessary data have to be repeatedly copied from the GPU to the main memory before the parallel calculations and back to the GPU memory after them. Naturally, this memory copy process also has a time requirement. The flow-sheet of the GPU assisted *.mex* function is presented in Figure 4.

The main limitations of the proposed method are the follows:

- Due to the repeated memory copy operation from and to the on-board GPU memory and to the fact that the benefits of GPU calculations is known to decrease with the volume of parallel operations, for crude meshes especially in one dimension (depending on the CPU configuration but generality if N < 1000) the serial implementation might be more beneficial. This will be analyzed later in this article.
- The fact that we use CUDA C++ language, which is an extension of C++ ensures a straightforward .mex file creation but this technique excludes all of non-CUDA capable GPU's. Moreover, as double precision operations are carried out which are supported starting from the "computing capability" of 1.3 CUDA enabled cards it further limits the list of accepted GPU's.

3. Results and discussions

The aforementioned three implementations are applied to solve three benchmark cases. Note that in this study all of the presented timings are the averages of three consecutive runs.

3.1. Mono dimensional pure growth PBE

For the first benchmark case let us consider a mono dimensional PBE written in volume form with growth only:

$$\frac{\partial n(v,t)}{\partial t} + \frac{\partial G(v) n(v,t)}{\partial v} = 0 \tag{10}$$

The initial distribution is expressed as:

$$n(v,0) = \frac{N_0}{v_0} exp\left(-\frac{v}{v_0}\right) \tag{11}$$

The growth rate is linearly size dependent given by the following function:

$$G(v) = G_0 v \tag{12}$$

In these conditions an analytical solution of the PBE can be found:

$$n(v,t) = \frac{N_0}{v_0} exp\left[-\frac{v}{v_0} exp(-G_0 t) - G_0 t\right] \quad (13)$$

Here the kinetic parameters given by Gunawan et al. (2004) are used which are listed in Table 2. Note that the implemented AES is the one dimensional analogue of Eq.(4) which, for space constraints, is not detailed here.

In the first investigation the PSD's calculated by the HR-FVM implementations are compared to the analytical solution after, presented in Figure 5. The numerical solutions practically are identical so only one is represented, which apparently overlaps with the analytical solution. Based on the error curve the deviation from the analytical solution is smaller than 0.03 % at each size. It seems that the .mex function is an order of magnitude faster than the Matlab function but still 6.3 times slower than the CUDA .mex. This can be explained with the fact that here all operations are parallelizable, which favors the use of the GPU.

The mesh size is known to significantly affect the HR-FVM solution: applying a finer mesh is expected to increase the accuracy but in the same time the computational costs are also rising.

In order to quantify the error committed by the numerical solutions, we use a sum square error based criteria defined as:

$$SSE = \frac{1}{(N-1)} \sum_{i=1}^{N-1} \frac{1}{n_a(i \cdot h)} \sqrt{(n_a(i \cdot h) - n_c(i \cdot h))^2} \quad (14)$$

where n_a denotes the analytically calculated number density, n_c is the numerically approximated number density at a given size.

In Figure 6 it can be seen that the committed error is the same for all implementations and it monotonically decreases with the mesh size. It is interesting that the .mex function/Matlab function speed up shows only weak dependence on the mesh size and generally is between 8

and 9. However, the CUDA .mex/.mex speed up is significantly increasing with the mesh size (for N = 2,000 the speed up is 2 but for the N = 17,000 it raises to 7). This suggests that the advantage of GPU is more significant when higher discretization is needed but for crude meshes the pure serial *.mex* implementation might become the most beneficial.

Due to the special hardware architecture of the GPUs, in contrast with the CPUs, not a single value is passed for computations at a time but a *vector of variables*. This vector of variables is handled by the *streaming multiprocessors* and finally each element of the vector is passed to a GPU core (for the nVidia cards the so-called CUDA core). In this way, the original data vector is divided to shorter vectors. The length of these shorter vectors is called the *thread dimension* and its maximal value is given by the GPU type. The number of these smaller vectors is called as *block dimension*. It is obvious that the *smaller thread dimension* results in *bigger block dimension* and reverse. Despite of the fact that finally elements of the input vector are processed, it seems that this division also affects the calculation speed.

The Figure 7 presents the effects of thread dimension on the GPU run time, using the same numerical configuration as in the case of Figure 5. It is observed that under the thread dimension of 64 the run time is considerably longer – so the GPU is heavily under-utilized. Between 64 and 416 a minimum point in run time exists at thread dimension of 256. Above the thread dimension value of 416 the run times seems to vary chaotically reaching a minimal value at 512 which is very similar to the run time obtained with 256 thread dimension. This means that, in order to maximize the performance, either 256 or 512 thread dimension should be applied. Note than in previous runs we used a 256 thread dimension. This thread dimension is used in the rest of the simulations presented in this article.

3.2. Mono-dimensional PBE with secondary nucleation and size dependent growth

In this part of the study, a batch cooling crystallization is considered with secondary nucleation, size dependent growth, linear cooling profile and mass balance. If the particles are characterized with a linear particle size, the corresponding PBE takes the form:

$$\frac{\partial n(L,t)}{\partial t} + \frac{\partial G(\sigma,L)n(L,t)}{\partial L} = B(\sigma)\delta(L - L_n) \tag{15}$$

With the initial condition:

$$n(L,0) = n_0(L) \tag{16}$$

The nucleation rate is expressed as:

$$B(\sigma) = k_b V_C \sigma^b \tag{17}$$

Here, the V_C denotes the total volume of existing crystals, σ stands for the relative supersaturation, $\sigma = S - 1 = c/c_s - 1$. For the expression of growth rate, the widely used relation is applied expressing size dependent growth that linearly depends on size:

$$G(\sigma,L) = k_g \sigma^g (1 + \gamma L) \tag{18}$$

where L is the particle size. In these experiments a linear cooling profile is assumed:

$$T = T_0 - c_r t \leftrightarrow \frac{dT}{dt} = -c_r, \qquad T(0) = T_0 \tag{19}$$

In Eq.(19) c_r denotes the cooling rate. The mass balance for the solute concentration takes the form:

$$\frac{dc}{dt} = -3k_V \rho_c \int_0^\infty L^2 \, G(\sigma,L) n(L,t) dL, \quad c(0) = c_0 \tag{20}$$

In Eq.(20) k_V is the volume shape factor. The solubility is considered temperature dependent and is described by the power law relation:

$$c_s(T) = a_0 + a_1 T + a_2 T^2 \tag{21}$$

The time step used is however constantly recalculated using an adaptive time-stepping approach using the Courant-Friedrichs-Lewy (CFL) criterion. In order to stabilize the numerical solution of an explicit method (like the HR-FVM), the CFL criterion should be less or equal than 1:

$$CFL = \max\left(G\frac{k}{h}\right) \tag{22}$$

In this study the CFL number is fixed and the time step is recalculated in each iteration according to the Eq.(22). This gives an adaptive time stepping feature for the simulation. The process and kinetic parameters for this test case used in the simulations are listed in Table 3. The kinetic parameters were chosen based on literature data (Ma, Tafti, and Braatz 2002), originally valid for the crystallization of potassium nitrate (KNO_3). Nevertheless, the objective of this work is not the analysis of a particular chemical system but the PBE solution method, thus the kinetic parameters in some batches were modified and a different size dependent growth rate equation is defined, which fits better the original purpose of the given work.

Note that the PBE Eq.(15) and mass balance Eq.(20) are discretized as it was presented in the second section of this paper and the resulted AES is solved. For space constrains the 1D HR-FVM is not presented.

The initial (seed) distribution is expressed as:

$$n_0(L) = \begin{cases} -3.48\,10^{-4}\,L^2 + 0.136L - 13.2 \; if \; 180.5 \le L \le 210.5 \\ 0 \; elsewhere \end{cases} \tag{23}$$

In the first investigation, the PSD's are simulated using the three different implementations. For the PBE shown in Eq.(15) there is no analytical solution thus the numerical distributions are compared to each other. In Figure 8, the particle size distributions are plotted in some

representative moments. It seems that after 200 s the nucleation becomes significant but the crystal growth is also significant (observe that a semi-logarithmical representation is applied). As long as the numerical results are practically identical, only one of them is represented to avoid the figure overloading. The corresponding timings are presented in Table 4. After 50 seconds simulation, the CUDA .mex code is 2.46 times faster than the .mex code and almost 19 times than the Matlab function. It is interesting to observe that the advantage of GPU accelerated solution is increasing with the simulation time. After 450 second simulated time the CUDA .mex function is almost 18.2 times faster than the .mex function and almost 39 times faster than the Matlab function. A reason for this may be that, due to the nucleation, in the number density vector the number of non-zero elements increases with the time. As double precision floating point operations are involved, handling these non-zero elements is significantly slower, which emphasize the advantage of GPU – where these operations are running in parallel.

Since there is no exact analytical solution for the Eq. (15), the accuracy of numerical solutions is verified by comparing the moments of the distributions calculated using the HR-FVM with the methods obtained from the method of moments approach. The method of moments is a widely used approach to solve the PBEs enabling the calculation of mean particulate quantities based on the moments of the distribution (Randolph and Larson, 1971). In this study we apply the moment transformation on the PBE and solve the generated moment equation system using the ode15s solver of the Matlab which uses back-differentiation formulas, with increased relative and absolute error tolerances (10^{-12} for both). Despite these moments are calculated numerically, due to the increased accuracy criteria used, we consider these moments as an "accurate" solution. Here we use the third moment of distribution (denoted as μ_3), a quantity which is proportional to the volume of particles, for comparison purposes.

Figure 9 a) illustrates the percentage error of the μ_3 of numerically calculated distributions as well as the speed up of .mex and CUDA .mex codes as a function of mesh size. It seems that, similarly to the pure growth case, the .mex : MatLab speed up shows weak dependency on the mesh size but the CUDA .mex : mex speed up increases significantly. The lowest speed up is ~1.7 for N = 2,000, and rises until ~15.5 for N = 40,000. It is observed that the .mex: MatLab speed up is almost constant, around 2.5. In the case of cruder meshes (N < 5,000) the committed error is higher, above 0.3 % and generally decreases with the mesh size, however, not monotonically. The mesh size N = 17,000 seems to be a good trade-off between the accuracy and computational costs. In this point, the CUDA .mex function is almost 10 times faster than the serial .mex.

The CFL criterion is also an important parameter of HR-FVM solution. As the CFL increases, the time step rises – so the run time decreases. Figure 9 b) presents the effects of CFL number on accuracy of HR-FVM solutions as well as the speed ups. It is observed that the .mex: MatLab speed up increases (from 4 to 9) and the CUDA .mex : .mex ratio decreases (from 6 to 2) with the CFL. At CFL = 0.55 there is a significant decrease in serial .mex run time, as it seen on speed up ratios. It seems again that the advantage of GPU acceleration is more accentuated for computationally expensive problems. The relative error in calculation of μ_3 is exactly the same for all implementations, which increases until the CFL = 0.7 above of which it presents a chaotic variation. According to the results, the 0.85 CFL has similar error as the 0.35 but the computational cost is less than half, which seems to be an excellent choice to use in simulations. However, when using higher CFLs, the possibility of numerical oscillations in the system is increasing thus the 0.85 CFL should be applied only after further investigations. Note that in these simulations size independent growth ($\gamma = 0$) was applied in order to avoid the over-stabilization of solution by applying the maximal growth rate which is defined by the maximal size-bin.

As long as the maximum allowable time step depends on the maximal growth rate, the computational time should depend on the crystallization kinetics. Figure 10 presents the effects of nucleation and growth rate constant on the speed up. Note that the error surfaces are not represented because, according to previous run, they are identical. The .mex : Matlab speed up seems to decrease with the growth rate constant (from 7 to 2.3) and no significant dependence can be observed with the nucleation rate. The CUDA .mex : .mex speed up is higher and surprisingly it increases with the growth rate constant (from 2.5 to 12) but the nucleation rate constant seems to not affect the speed up considerably. Note that the noise in the speed up surfaces is a result of run time variations of the serial code as the CPU is also used by the operating system. This noise is partially reduced by averaging three run times.

3.3 Two dimensional PBE with secondary nucleation and size dependent growths

More recently, there is an increased need to describe not only the size but the shape variations of the particles during the crystallization. This is achieved by the so-called morphological population balances which have at least two dimensions. Now let us consider a two dimensional PBE, generally used to describe the crystallization of rod-like crystals (Borsos and Lakatos, 2013, Szilagyi et al, 2015) and plate like crystals (Szilagyi and Lakatos, 2015). The Eq.(2) is the general two dimensional PBE taking into the consideration the growth along the length and width coordinates as well as the nucleation. The mass balance, required to concentration and supersaturation calculation is given by Eq.(8). The secondary nucleation rate Eq.(17) and the growth rates Eq.(18) are used distinguishing the kinetic parameters for the growth rate of two facets.

A linear cooling profile is applied Eq.(19). The solubility is described by a power-law equation Eq.(21) and the adaptive time stepping Eq.(22) is applied. The parameters used in simulations are listed in Table 5. The population balance model, the mass balance and the 2D HR-FVM equations are described by Eqs.(1)-(9). The applied kinetic parameters are listed in Table 5

which are inspired based on literature data (Ma, Tafti, and Braatz 2002) and were slightly modified to obtain a system behaviour, which fits better the goals of the current analysis. Uncorrelated bivariate log-normal based seed distribution was considered with m_1 = 50 and m_2 = 6 μm means and, respectively, v_1 = 6 and v_2 = 4 μm dispersion along the length and width axes:

$$n_0(L_1, L_2) = 10^8 \frac{1}{\sqrt{2\pi} L_1 L_2 \prod_{i=1}^{2} \sigma_i} exp\left[-\sum_{i=1}^{2} \frac{[lnL_i - \mu_i]^2}{2\sigma_i} \right] \tag{24}$$

Where

$$\mu_i = ln\left(\frac{m_i}{\sqrt{1 + \frac{v_i}{m_i^2}}} \right), i = 1,2$$

$$\sigma_i = \sqrt{ln\left(1 + \frac{v_i}{m_i^2}\right)}, i = 1,2 \tag{24a}$$

Let us start the investigations with computing the PSD-s in some representative time moments. As the graphical representation of a bivariate PSD is a surface in the plot shown in Figure 11 only the PSD's calculated by the CUDA .mex are represented at different times during the simulation. In the surface-series can be observed that the nucleation has visible significant effects only after 1800 s, under that the growth is the dominant phenomena. A reason for this may be that here a secondary nucleation is assumed, which has a rate that is proportional to the total volume of existing particles. Moreover, the supersaturation exponent is higher than for the growth rates thus the system presents an explosive nucleation at higher – but decreased nucleation rate at lower supersaturations. Based on the run times, at the first look it seems that the .mex function is with 1 order of magnitude faster than the MatLab but with 1 order of magnitude slower than the CUDA .mex function. However, here the PSD computed by the

CUDA .mex is represented, according to the numerical results, the surfaces would practically overlap as in the Figure 8.

Similarly as in the 1D PBE case with nucleation and growth, the method of moments is applied to compute the mixed moments of the bivariate size distribution. The μ_{12} joint moment is then used for comparison purposes, a quantity proportional to the specific volume of crystals. This, as applied high accuracy tolerances, is considered as the “accurate solution”.

In Figure 12 a) the accuracy and the speed up is represented as a function of CFL criterion. The .mex : MatLab speed up slightly increases (from 21 to 30) but the CUDA .mex : .mex decreases with the CFL number (from 10 to 7). The CFL = 0.5 seems to be a threshold for the .mex function where is getting significantly faster generating a step-like variation in both speed up curves. The accuracy of all implementations is practically the same and is increasing with the CFL until 0.45 above which presents chaotically variations. Note that in this investigation size independent growths were applied to avoid the solution over-stabilization caused by the reasons discussed earlier in this article. Thus, these significant variations in accuracy may be explained with the possibly appearance of small numerical oscillations when running simulations with higher CFL-s.

The Figure 12 b) presents the effects of mesh size (N) on accuracy and speed up. It seems that for the cruder mesh (N = 300 or h = 2 μm discretization) the error is almost 1 % and it decreases fast with the mesh size (at N = 600 or h = 1 μm discretization is around of 0.5 %) and for the finest division (N = 3000 or h=0.2 μm) is only 0.25 %. Note that the N = 1,500 element number (0.375 μm) presents a local minima in the error curve thus it may be a good choice to use in simulation of this system from the point of view of accuracy. The CUDA .mex : .mex speed up is monotonically increasing with the mesh size from 5 to 18 and the .mex : MatLab is decreasing. Note that the MatLab simulations were not carried out for the finer meshes as the

computational demands are extremely high. According to the Figure 12 b), the run time, which for a bivariate PBE is a quadratic function of mesh size, with N = 1,500 division required ~18,000 seconds (~5 hours). According to both investigations, the advantage of CUDA .mex over the .mex function is higher for the heavier calculations.

The following investigation focuses on effects of growth rate constants on the CUDA .mex and .mex code performances. Note that here the MatLab function is not used due to its increased run time, which makes it practically unusable for simulation purposes compared with the other two implementations. In Figure 13 a) seen that the advantage of GPU accelerated solution is sensitive to the applied crystallization kinetics: for the lower growth rates the advantage is the smallest, presenting a speed up of 6 and for the higher growth rates it increases almost to 9. The actual CUDA .mex run time, illustrated on Figure 13 b) presents obvious trends: increasing the growth rates the run time increases almost linearly: in these simulations it varies between 29 and 41 seconds. Taking into consideration that 2100 second process time is simulated, the GPU accelerated solution seems to be fast enough, namely at least 41 times and up to 73 (which may be slightly enhanced by optimizing the thread dimension and can be boosted further by reducing the mesh size), to be applicable even in real time control systems. For the .mex function, the run times varies between 183 and 358 seconds thus, it is also provides real time simulation but, for instance in a model based control system, due to the reduced possibly iteration number (the simulation is up to 11 times faster than the process) optimizer would hardly found the optimal control signal.

To investigate how the performance of the GPU-PBM implementation depends on hardware architecture, a comparison study running the same 2D simulation (t = 3600 s) on different computers and GPUs is carried out. The kinetic and process parameters applied in the comparison study are listed in Table 6. The first PC involved in these tests was used in previous investigations presented so far in this article (~2012 technology); the second is a notebook

(~2014 technology); the thirth is a Dell Precision workstation (~2011 technology) equipped with a compute nVidia Tesla GPU; the fourth is a mid-priced custom configuration workstation destined to number crunching built from last generation components (by late 2015) while the last is a Dell Precision workstation (~2015 technology) equipped with a high-end nVidia Tesla K20X compute processor. The results are presented in Table 7. In the line of "Host" specifications, the serial .mex function results are listed but in the row of "Device" properties the CUDA C++ timings are presented (that implements the parallel GPU-PBM). It can be observed that in each computer the CUDA C++ simulation over-performed the serial C++ code. Surprisingly, the nVidia GeForce GTX 970 GPU over-performed both nVidia Tesla GPU's, which were developed for massive scientific calculations. The explanation might be that in this GPU the memory and GPU clock is also considerably higher than in the Tesla cards. It is also important that the CPU clock in this computer is almost the double of the Tesla workstation's CPU clock, which also has a significant effect on the speed of calculations. On the other hand, the Tesla GPUs produced the highest speed-ups, which justify their application, as well as the error-correcting memory. As it was expected the last generation 4GHz i7 processor equipped with DDR4 memory gave far the best serial performance but surprisingly, the notebook i3 processor produced outstanding serial computation performance. This suggests that the memory frequency is also a significant aspect in running these massive simulations.

4. The CrySiV tool, a MatLab based simulation software

The speed ups achieved with the .mex and CUDA .mex functions over the MatLab implementations presented so far are attractive. The increase of 1-2 orders of magnitude in computational time for the solutions of 2D-PBM can make the GPU-PBM solution framework proposed in this paper a unique platform to bring real-time model-based control using full 2D-PBM codes into the realm of possibility. To share with other users this highly efficient

numerical solution platform, we created a software, the Crystallization Simulation and Visualization Tool, called *CrySiV*.

The main *criteria* in developing the tool were to provide a software package for dual use, namely:

- Should have similar structure to the usual MatLab functions: the inputs (constants of the kinetic equations) are given as row vectors, similarly the temperature profile data and initial PSD. The specific solver options, as the mesh size, CFL number etc. should be specified in a separate structure for options. This generic function is typically designated for process optimization and control purposes and provides full flexibility for a MatLab user to incorporate 1D or 2D PBM solutions in their custom Matlab codes.
- The program should also have a Graphical User Interface (GUI), which should enable an interactive and clear visualization of the crystallization process. The GUI is addressed for basic users and/or educational purposes.

In both implementations using the GPU acceleration via the CUDA C++ implementation is an option, including automatic detection of the suitable GPU card in the computer.

Some of the extended features of software are:

- Include the primary nucleation rate:

$$B(S) = k_p \sigma^b exp\left(-\frac{k_e}{\ln^2(\mathrm{S})}\right) exp\left(-\frac{E_p}{RT}\right) \tag{24}$$

- Use a more general, temperature dependent secondary nucleation rate equation:

$$B(S) = k_b V_c^j \sigma^b exp\left(-\frac{E_b}{RT}\right) \tag{25}$$

- Use a more general, temperature and size dependent growth rate equation:

$$G(\sigma, L) = k_g \sigma^g (\alpha + \gamma L^\beta) exp\left(-\frac{E_g}{RT}\right) \tag{26}$$

- Include the dissolution rate into the model:

$$D(\sigma, L) = k_d(1-\sigma)^d(\alpha_d + \gamma_d L^{\beta_d})exp\left(-\frac{E_d}{RT}\right) \quad (27)$$

- Extend the cooling equation (and, permit the use of custom temperature profile, given as a vector of time and corresponding temperature values):

$$T = T_0 + c_1 t + c_2 t^2 + c_3 \exp(c_4 t) \quad (28)$$

- Include both the power law solubility equation, as well as the Apelblat solubility model:

$$c_s(T) = \exp\left[a_0 + \frac{a_1}{T} + a_3 \ln(T)\right] \quad (29)$$

Applying the extended kinetics and process Eqs. (24)-(29), a wide variety of cooling crystallization problems can be easily simulated, from the size dependent growth to the Oswald ripening using temperature cycling. Using the user-defined temperature profile and enabling the dissolution, it makes possible the easy simulation of the cyclic temperature profile – and its effects to the crystal shape, if applied to the two dimensional PBE. Using the GPU acceleration the solution time can significantly be shortened especially for the more complex calculations, as it was presented in the previous sections.

A generic MatLab function was created named as *crysiv*, which can be called form the MatLab environment, and configured with the constants of the Eqs. (24)-(29) (the initial conditions, process conditions and solver specifications as mesh size, minimal and maximal crystal size, CFL number etc.). This function has a detailed input data verification, which ensures that the compiled .mex files are called with correct inputs avoiding the fatal memory errors and helping the users with the correct parameterization. The function returns the PSD, concentrations, moments calculated based on the PSD and the quadrature method of moments based errors of the FVM moments, in the specified sample times.

A user-friendly GUI was also created in MatLab, which aims to provide an interactive and easy-to-use platform to simulate, analyse and visualize in 1D and 2D the crystallization process. Figure 14 shows the main window of the CrySiV GUI. During the simulation, the actual system states and the simulation results obtained from the beginning are presented and dynamically updated on two plots. In both plots, a variety of quantities can be presented *via* the selection from a pop up menu like the variation of concentration and temperatures, mean crystal size(s), PSD in 2D and 3D representation and phase diagram. The GUI enables the saving and loading the model parameterization and simulation data, too. The simulation results can be loaded separately and can be compared to other simulations, as the CrySiV makes possible the animation of simulations from the saved data. This tool is freely available for non-profit use, by contacting the corresponding author and from the project website.

5. Conclusions

In the current work three different implementations of the high resolution finite volume method (HR FVM), namely as a MatLab function, compiled C++ .mex file and compiled CUDA C++ .mex file, were studied and presented for solving mono and bivariate population balance equations (PBE). Generally, the performance in different applications is strongly related to the nature of problem: the advantage of compiled .mex file is the faster serial calculation but GPU has a massive parallel architecture which makes it advantageous to perform parallel calculations of higher computational demand. In order to simultaneously benefit from the advantages CPU and GPU, in this study a hybrid algorithm was developed solving the parallel operations on GPU and passing the serial calculations to the CPU exploring maximally the computational power of the computer without overloading the CPU.

The effects of different aspects of the algorithm were studied on computational speed like the mesh size or Courant-Friedrichs-Lewy (CFL) criterion as well as the parallel programming

specific properties like the thread dimension. It was found that all these setting affect the computational performance and accuracy as well but not necessarily linearly thus attractive numerical configurations can be found. According to the simulation results, in the case of mono dimensional PBE with nucleation and growth the runs with CFL = 0.85 gave as accurate results as the CFL = 0.35 compared to the method of moments solution, with less than half computational cost. The advantage of GPU in all cases increased monotonically compared to the pure CPU performance (both .mex and MatLab function). Moreover, it was found that the GPU calculations do not introduce observable additional errors.

For the pure growth one dimensional population balance involving only parallelizable calculations the GPU was up to 60 times faster than the MatLab function and up to 7 times faster than the .mex code. In the case of one dimensional PBE with secondary nucleation and size dependent growth speed up was more spectacular: here the MatLab : .mex : CUDA .mex run time ratio is up to 38 : 18 : 1. In two dimensional PBE solutions the computational cost is quadratic function of the mesh size. The MatLab implementation showed poor performance; it was generally 20-50 times slower compared to the .mex implementation. The GPU acceleration led to even 18 times speed up compared to the pure serial .mex function. In all cases (1D and 2D) it was observed that the improvement is more significant for the heavier calculations (finer mesh, lower CFL, higher growth rate, longer simulated time *etc.*).

One of the key advantages of the GPU-PBM framework proposed here is the fact that in the current study a low-cost GPU card is used, similar to what already exists in many PC's, and significant speed up is achieved, which makes the method more attractive in application of real time model-based control approaches.

Based on the presented programs, a MatLab based simulator was created, named as CrySiV (Crystallization Simulation and Visualization Tool), which uses extended crystallization

kinetics, including primary and secondary nucleation, growth and dissolution of crystals. The software has similar structure to the built-in MatLab programs to ease its application for the process engineers, whose this function was typically created. A graphical user interface was created to the CrySiV which aims to make the simulation of crystallization process spectacular and easy to use and the comparison of simulation results simple and fast. This is addressed for the basic users and/or educational purposes.

Acknowledgments

Funding is acknowledged from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement No. [280106-CrySys]. Financial support of the Sectorial Operational Programme for Human Resources Development 2007-2013, co-financed by the European Social Fund, under the project POSDRU/159/1.5/S/132400 – "Young successful researchers – professional development in an international and interdisciplinary environment" is also acknowledged. The authors would also like to acknowledgement prof. Zsolt Ulbert for running the simulations in the nVidia Tesla machine.

References

Aamir E., 2010. Population balance model-based optimal control of batch crystallisation processes for systematic crystal size distribution design. PhD thesis. Loughborough University, Department of Chemical Engineering, UK.

Aamir E., Nagy Z.K., Rielly C.D., 2010. Optimal seed recipe design for crystal size distribution control for batch cooling crystallization processes. Chemical Engineering Science 65, 11, 3602-3614.

Aamir E., Nagy Z.K., Rielly C.D., Kleinert T., Judat B., 2009. Combined quadrature method of moments and method of characteristics approach for efficient solution of population balance models for dynamic modeling and crystal size distribution control of crystallization processes. Industrial and Engineering Chemistry 48, 18, 8575 – 8584.

Bárkányi, Á.,Németh, S., Lakatos, B.G., 2013. Modelling and simulation of suspension polymerization of vinyl chloride via population balance model. Computers and Chemical Engineering 59, 211-218.

Borsos, Á., Lakatos, B.G., 2013. Investigation and simulation of crystallization of high aspect ratio crystals with fragmentation. Chemical Engineering Research and Design 92/6, 1133-1141

CookS., 2012, CUDA Programming, Morgan Kaufman, USA

Frenklach M., 2002. Method of moments with interpolative closure. Chemical Engineering Science 57, 12, 2229-2239

Gunawan R., Fusman I., Braatz R.D., 2004. High Resolution Algorithms for Multidimensional Population Balance Equations, AIChE Journal 50-14, 2738 – 2749

Gunawan R., Fusman I., Braatz R.D., 2008. Parallel High Resolution Simulation of Particulate Processes, AIChE Journal, DOI: 10.1002/aic.11484

Hulburt H.M., Katz S., 1964, Some Problems in Particle Technology. A statistical Mechanical Formulation, Chemical Engineering Science 19, 555-574

Irizarry, R., 2008. Fast Monte Carlo methodology for multivariate particulate systems—I: Point ensemble Monte Carlo. Chemical Engineering Science 63, 1, 95-110

Lakatos G. B., 2008. Population balance model for mixing in continuous flow systems. Chemical Engineering Science 63, 2, 404-423.

LeVeque R.J., 1992. Numerical methods for conservation laws. Birkhauser Verlag, Basel, Germany.

LeVeque R.J., 2002. Finite volume methods for hyperbolic problems. Cambridge University Press, Cambridge, UK.

Ma D.L., Tafti D.K., Braatz, R.D, 2002. “High-Resolution Simulation of Multidimensional Crystal Growth. Industrial & Engineering Chemistry Research 41,25, 6217–23

Majumder A., KariwalaV., AnsumaliS., RajendranA., 2010, Fast High-Resolution Method for Solving Multidimensional Population Balances in Crystallization, Industrial & Engineering Chemistry Research 49, 3862-3872

McGraw, R., 1997. Description of aerosol dynamics by the quadrature method of moments. Aerosol Science and Technology 27, 255-265.

Nagy Z.K, Fevotte G., Kramer H., Simon L.L., 2013. Recent advances in the monitoring, modelling and control of crystallization systems. Chemical Engineering Research and Design 91, 10, 1903-1922.

Prakash A.V., ChaudhuryA., BarrassoD., RamachandranR., 2013, Simulation of Population Balance Model Based Particulate Processes via Parallel and Distributed Computing, Chemical Engineering Research and Design 91-7, 2159-1271

Qamar S., Ashfaq A., Warnecke G., Angelov I., Elsner M.P., Seider-MorgesternA., 2007, Adaptive High-Resolution Schemes for Multidimensional Population Balances in Crystallization Processes, Computers & Chemical Engineering 31-10, 1296-1311

QamarS., Elsner M.P., Angelov I., WarneckeG., Seider-MorgesternA., 2006, A Comparative Study of High Resolution Schemes for Solving Population Balances in Crystallization, Computers & Chemical Engineering 30-6/7, 1119-1131

Randolph A., Larson M., 1971.Theory of particulate processes. Academic Press, New York

Randolph A., Larson M., 1988. Theory of particulate processes: analysis and techniques of continuous crystallization**.** Academic Press, Salt Lake City

Santos F.P., Senocak I., Favero J.L., Lage P.L.C., 2013. Solution of the population balance equation using parallel adaptive cubature on GPUs. Computers & Chemical Engineering 55, 8, 61-70.

Smith M., Matsoukas T., 1988. Constant-number Monte Carlo simulation of population balances. Chemical Engineering Science 53, 9, 1777-1786

Szilágyi B, Muntean N, Barabás R, Oana P, Lakatos G.B, 2015. Reaction precipitation of amorphous calcium phosphate: Population balance modeling and kinetics, Chemical Engineering Research and Design, 93, 278 – 286.

Szilágyi B., Agachi P.S., Lakatos B.G., 2015. Numerical investigation of crystallization of high aspect ratio crystals with breakage. Powder Technology, 283, 152-162.

Szilágyi B., Lakatos B.G., 2015.Batch cooling crystallization of plate-like crystals: a simulation study. Periodica Polytechnica Chemical Engineering, DOI: 10.3311/PPch.7581

Ulbert Zs, Lakatos G.B., 2007. Dynamic simulation of crystallization processes: Adaptive finite element collocation method. AlChE Journal 53, 12, 3089-3107.

Valentas J.V., Amundson N.R., 1966. Breakage and coalescence in dispersed phase systems. Industrial & Engineering Chemistry Fundamentals 5, 4, 533-542.

Wei J., 2014. A parallel Monte Carlo method for population balance modeling of particulate processes using bookkeeping strategy. Physics A: Statistical Mechanics and its Applications 402, 186-197.

Wei J., Kruis F.E., 2013. GPU-accelerated Monte Carlo simulation of particle coagulation based on the inverse method. Journal of Computational Physics 249, 67-79.

Xu Z., Zhao H., Zheng C., 2015. Accelerating population balance-Monte Carlo simulation for coagulation dynamics from the Markov jump model, stochastic algorithm and GPU parallel computing. Journal of Computational Physics 281, 844-863.

List of notations

a_k - coefficients of the characteristic solubility expression, -

b - exponent nucleation rate, -

B - nucleation rate # $\mu m^{-3}s^{-1}$

c - concentration of solute, kg m^{-3}

c_i - coefficient of temperature profile equation, i = 1,2,…,5

c_r - cooling rate, $^oC\ s^{-1}$

c_s - solubility concentration, kg m^{-3}

E - activation energy, kJ $kmol^{-1}\ K^{-1}$

G - crystal growth rate, m s^{-1}

g - exponent of crystal growth rate

k_b - rate coefficient of nucleation, # $\mu m^{-3} s^{-1}$

k_e - parameter of primary nucleation, -

k_g - rate coefficient of crystal growth, $\mu m\ s^{-1}$

k_V - volume shape factor , -

L - linear size of crystals, m

N - mesh size, #

n - population density function, # m^{-5}

R - gas constant, 8.31 kJ $kmol^{-1}$

S - supersaturation ratio, -

T - temperature, ^oC

V_C - volume fraction of crystal population, $m^3 m^{-3}$

Greek letters

ρ - density, kgm^{-3}

σ - relative supersaturation, -

$\mu_{k,m}$ - $(k,m)^{th}$ order mixed moment

α - coefficient of growth rate function

β - exponent of growth rate function

δ - Dirac delta function

γ - coefficient of growth rate function, μm^{-1}

Subscripts

0 - initial value

1,2 - characteristic crystal facet, 1…3

b - secondary nucleation

d - dissolution

n - nuclei

p - primary nucleation

Abbreviations

AES - Algebraic Equation System

CFL - Courant Friedrichs Lewy criterion

CPU - Central Processing Unit

CUDA - Compute Unified Device Architecture

FVM - Finite Volume Method

GPU - Graphical Processing Unit

GUI - Graphical User Interface

HR - High Resolution

PB - Population Balance

PBE - Population Balance Equation

PCT - Parallel Computing Toolbox (MatLab)

PSD - Particle Size Distribution

ODE - Ordinary Differential Equation

List of Tables

List of Figures

Figure 11. The calculated bivariate PSD's after a) 600, b) 1200, c) 1800 and d) 2100 seconds

Figure 12. Dependence of acceleration ratio and accuracy on the a) CFL criterion with size independent growth and b) division number in one direction (N) with size dependent growth in solution of a two dimensional PBE.

Figure 13. a) Dependence of acceleration ratio and b) CUDA .mex function run time on growth rate constants (k_{g1} and k_{g2}) compared to the .mex code

Figure 14. The interface of the *CrySiV* program, a MatLab based simulator for cooling batch crystallization processes

Table 1. Machine specifications

Property	***"Host" – CPU***	***"Device" – GPU***
Type	AMD Phenom II X4 965	Gigabyte nVidia GT 640
No. of proc. / freq.	4; 3400 MHz	384; 1046 MHz
Memory specifications	16GB DDR3;1333 MHz	1GB GDDR5; 5000 MHz
Price	~ 100 USD	~ 100 USD
Compiler	Visual Studio 2010 Prof.	nVidia CUDA Toolkit 6.5

Table 2. The process and kinetic parameters used in the solution of mono-dimensional population balance with constant growth only

$k = 10^{-5}$ [s]	$v_0 = 10^{-2}$ [μm^3]	$N_0 = 1$ [#/μm^3]
$t_{max} = 7$ s	$G_0 = 0.1$ [$\mu m/s$]	N = 10000 [#]

Table 3. The process and kinetic parameters used in the solution of mono-dimensional population balance equation with nucleation and growth

$k_b = 4.48\ 10^{-7}$ [#$\mu m^{-3} s^{-1}$]	$k_g = 116$ [$\mu m\ s^{-1}$]	g = 1.32 [-]
b = 1.78 [-]	$c_r = 0.0167$ [$^oC\ s^{-1}$]	$\gamma = 0.1$ [μm^{-1}]
$T_0 = 32$ [^oC]	$c_0 = c_s(T_0)$ [kg m^{-3}]	$a_0 = 0.1286$ [kg m^{-3}]
$a_1 = -5.88\ 10^{-3}$ [kg $m^{-3}\ ^oC^{-1}$]	$a_2 = 1.72\ 10^{-4}$ [kg $m^{-3}\ ^oC^{-2}$]	N = 32000 [#]
h = 0.05 [μm]	CFL = 0.5 [-]	$k_V = \pi/6$ [-]
$t_{max} = 450$ [s]	$\rho_c = 2110$ [kg m^{-3}]	

Table 4. The corresponding computational times obtained for the simulations presented in Figure 8.

Simulated time [s]	CUDA .mex run time [s]	CUDA .mex : .mex speed up	CUDA .mex : MatLab speed up
50	0.2222	2.46	18.88
100	0.9250	2.67	22.11
200	4.4490	3.35	23.87
300	11.7358	5.98	26.72
400	22.5248	13.24	36.19
450	32.6778	18.24	38.68

Table 5. The process and kinetic parameters used in the solution of two dimensional population balance equation with nucleation and growth

$k_b = 7.49 \cdot 10^{-8}$ [#µm^{-3}s^{-1}]	$k_{g1} = 5.75$ [µm s^{-1}]	$k_{g2} = 4.21$ [µm s^{-1}]
$g_1 = 1.34$ [-]	$g_2 = 1.38$ [-]	$\gamma_1 = 1.5 \cdot 10^{-3}$ [µm^{-1}]
$\gamma_2 = 1.8 \cdot 10^{-3}$ [µm^{-1}]	b = 2.04 [-]	$c_r = 0.0167$ [^{o}C s^{-1}]
$T_0 = 32$ [^{o}C]	$c_0 = c_s(T_0)$ [kg m^{-3}]	$a_0 = 0.2087$ [kg m^{-3}]
$a_1 = -9.76 \cdot 10^{-5}$[kg m^{-3} ^{o}C^{-1}]	$a_2 = 9.3 \cdot 10^{-5}$ [kg m^{-3} ^{o}C^{-2}]	N = 1200 [#]
h = 0.5 [µm]	CFL = 0.5 [-]	$k_V = 1$ [-]
$t_{max} = 3600$ [s]	$\rho_c = 2338$ [kg m^{-3}]	

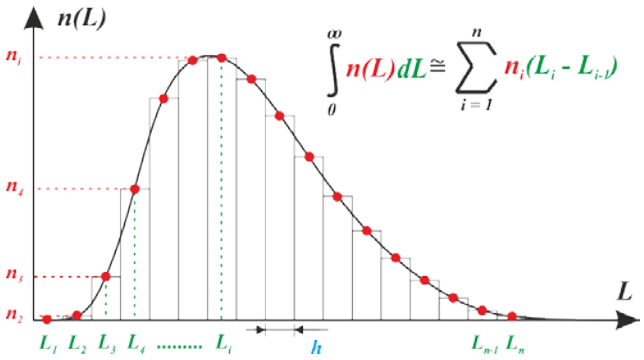
Table 6. Kinetic and process parameters used in comparison studies

$k_b = 0$ [#µm^{-3}s^{-1}]	$k_{g1} = 110$ [µm s^{-1}]	$k_{g2} = 60$ [µm s^{-1}]
$g_1 = 1.44$ [-]	$g_2 = 1.24$ [-]	$\gamma_1 = 6 \cdot 10^{-4}$ [µm^{-1}]
$\gamma_2 = 2 \cdot 10^{-4}$ [µm^{-1}]	b = 0 [-]	$c_r = 0.0033$ [^{o}C s^{-1}]
$T_0 = 32$ [^{o}C]	$c_0 = c_s(T_0)$ [kg m^{-3}]	$a_0 = 0.2087$ [kg m^{-3}]
$a_1 = -9.76 \cdot 10^{-5}$[kg m^{-3} ^{o}C^{-1}]	$a_2 = 9.3 \cdot 10^{-5}$ [kg m^{-3} ^{o}C^{-2}]	CFL = 0.5 [-]
$k_V = 1$ [-]	$t_{max} = 3600$ [s]	$\rho_c = 2338$ [kg m^{-3}]
$m_1 = 110$ [µm]	$m_2 = 100$ [µm]	$v_1 = 150$ [µm]
$v_2 = 70$ [µm]		

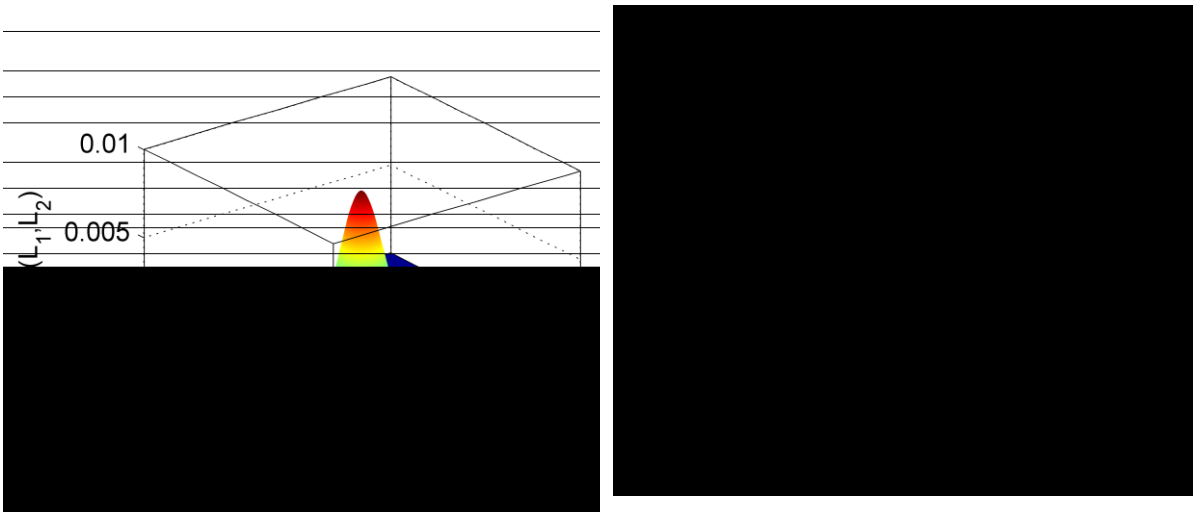
Table 7. Computational performance on different machines. The simulated (process) time is 3600 s

Machine			Timings (Mesh size) [s]		
no.	Machine specifications		1000	2000	3000
1 (PC)	Host:	AMD Phenom 2 X4 3.4 GHz CPU, 1333 MHz memory	359	2510	6780
	Device:	***nVidia GT 640 GPU, 384 cores, 5000 MHz memory***	***40.3***	***161***	***368***
	Speedup		8.9	15.6	18.24
2 (notebook)	Host:	Intel Core i3 4000M 2.4 GHz CPU, 1600 MHz memory	248	2265	4750
	Device:	***nVidia GT 720M GPU, 96 cores, 5000 MHz memory***	***56***	***X****	***X****
	Speedup		4.42	-	-
3 (Workstation)	Host:	Intel Xeon E5500, 2.3 GHz CPU, 1066 MHz memory	515	4232	11556
	Device:	***nVidia Tesla C2075 GPU, 448 cores, 3000 MHz memory***	***10.2***	***39***	***95.9***
	Speedup		50.4	108.5	120.5
4 (Workstation)	Host:	Intel i7 6700K, 4 GHz CPU, 2133 MHz memory	102	778	1977
	Device:	***nVidia GeForce GTX 970 GPU, 1664 cores, 7000 MHz memory***	***8.46***	***31***	***73.5***
	Speedup		12.0	25.1	26.6
5 (Workstation)	Host:	Intel Xeon E5-2620, 2.4 GHz CPU, 2133 MHz memory	252	1761	4379
	Device:	***nVidia Tesla K20X GPU, 2688 cores, 5200 MHz memory***	***9.00***	***35.1***	***85.5***
	Speedup		28	50.1	51.1

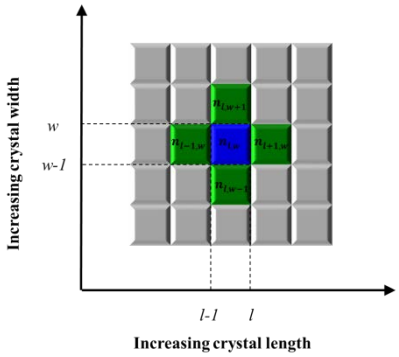
* out of GPU memory



a)



b)



c)

Figure 1. Representation of the finite volume discretization of a) one dimensional PSD b) two dimensional PSD and c) uniform 2D grid

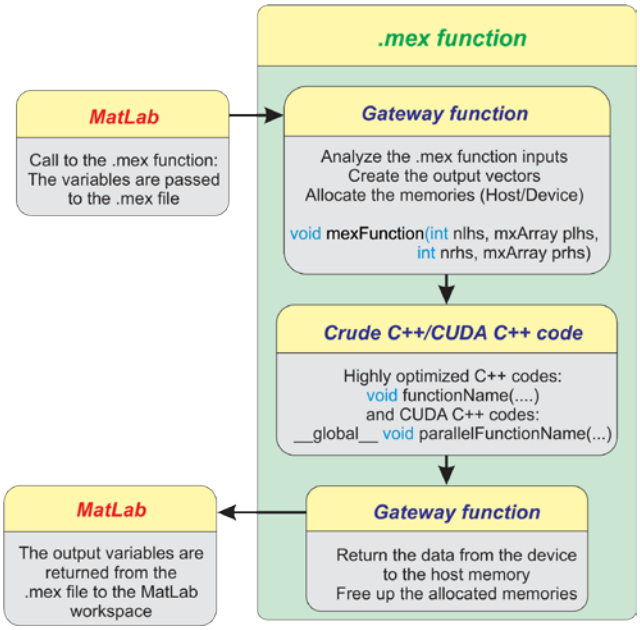


Figure 2. The anatomy of a *.mex* function: the Gateway function makes the connection between the MatLab and the crude C++ code

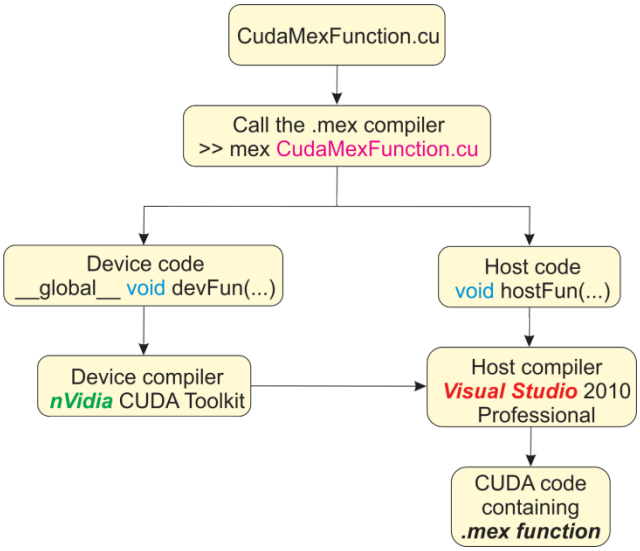


Figure 3. Schematic representation of compilation process of the CUDA code containing .mex file: the role of nVidia CUDA – and Visual Studio C++ compilers

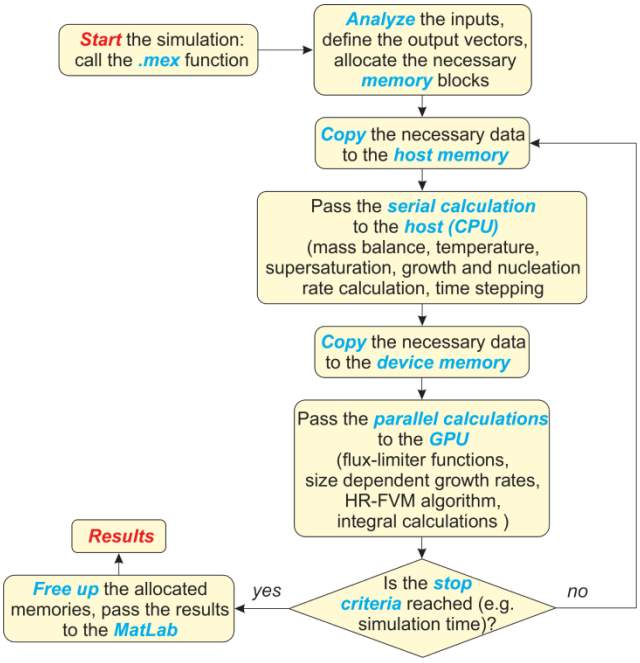


Figure 4. The flow-sheet of the GPU acceleration: the parallel calculations are performed on the GPU but the CPU runs the serial computations

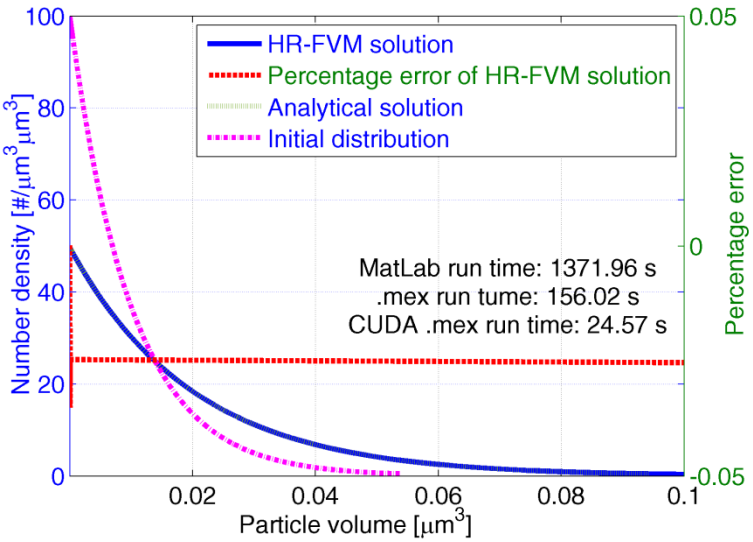


Figure 5. Comparison of the analytical solution with the three HR-FVM implementation results and the percentage errors committed by the numerical solutions of a mono dimensional pure-growth PBE

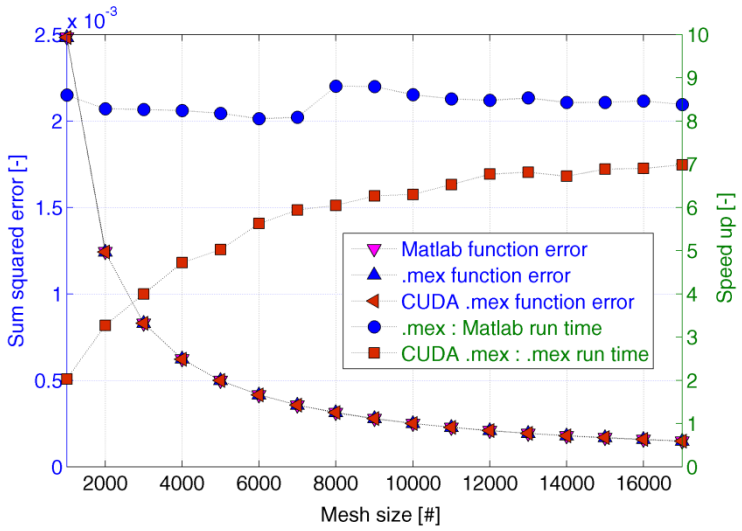


Figure 6. The dependence of acceleration ratio and accuracy on the mesh size (*N*) in solution of a mono dimensional pure-growth PBE

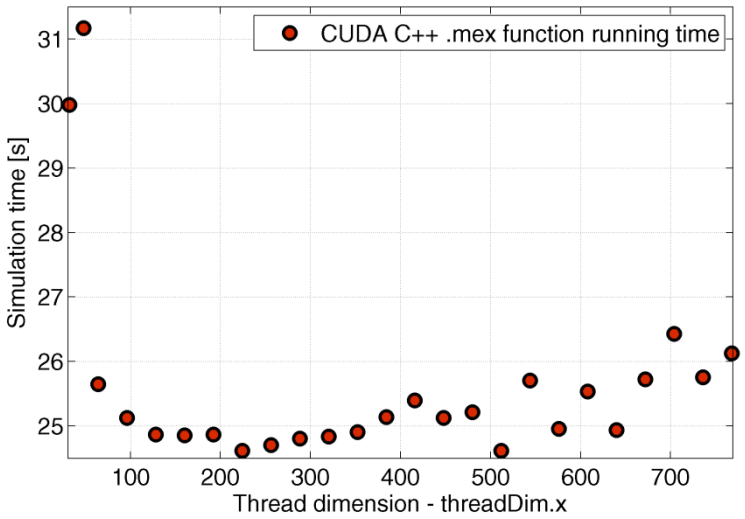
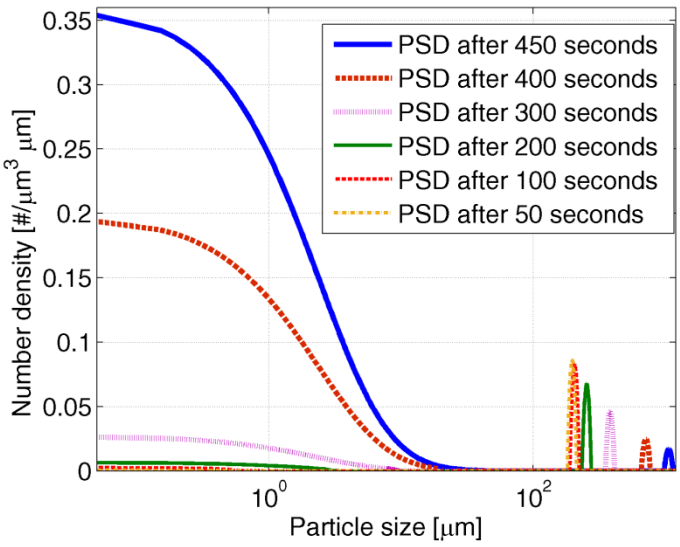
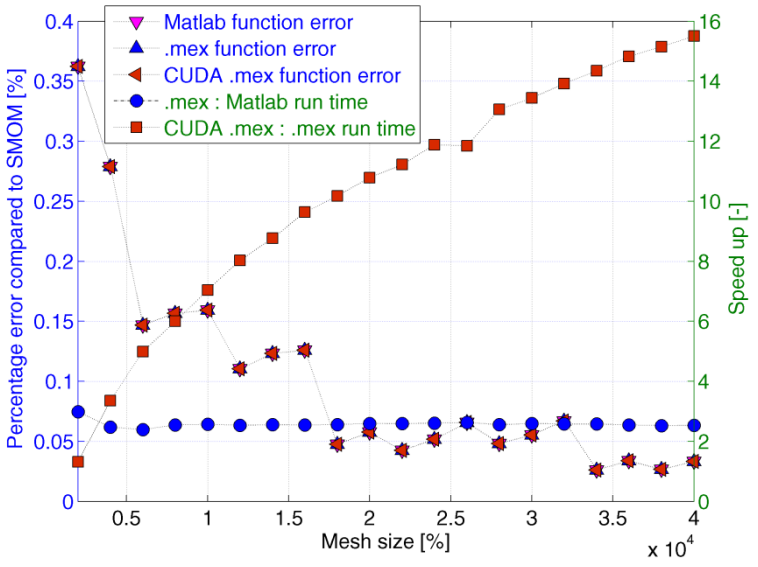


Figure 7. Effects of thread dimension on GPU accelerated solution time of a mono dimensional pure growth PBE

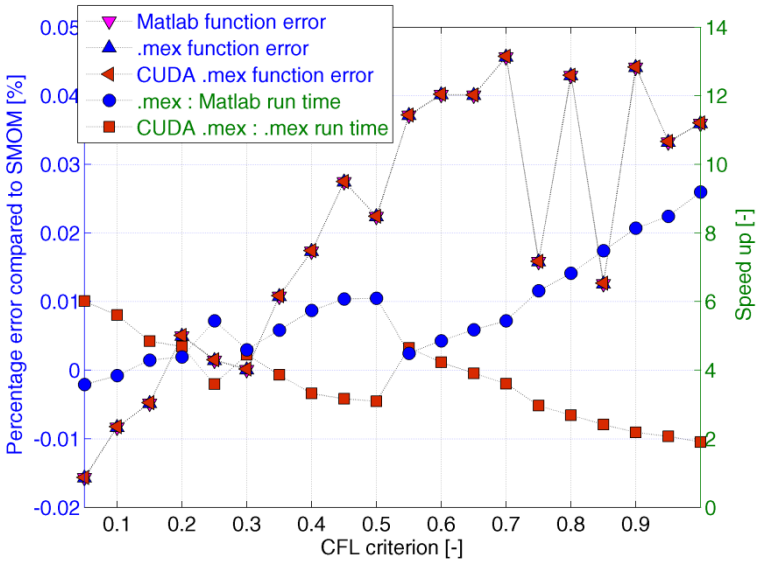


e)

Figure 8. The calculated PSD's after 50, 100, 200, 300, 400, 450 seconds for a mono-dimensional PBE with nucleation and size dependent growth



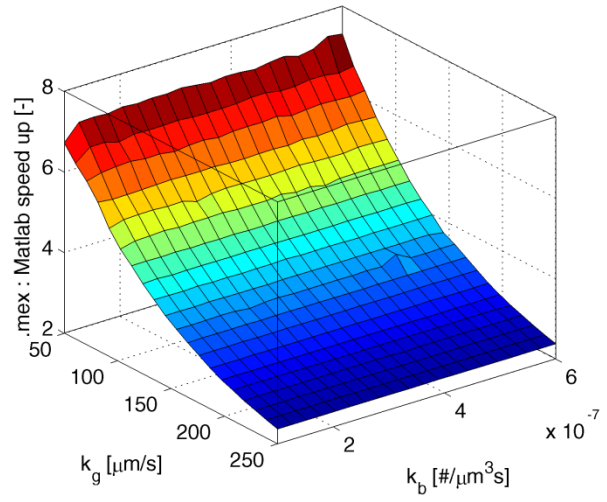
a)



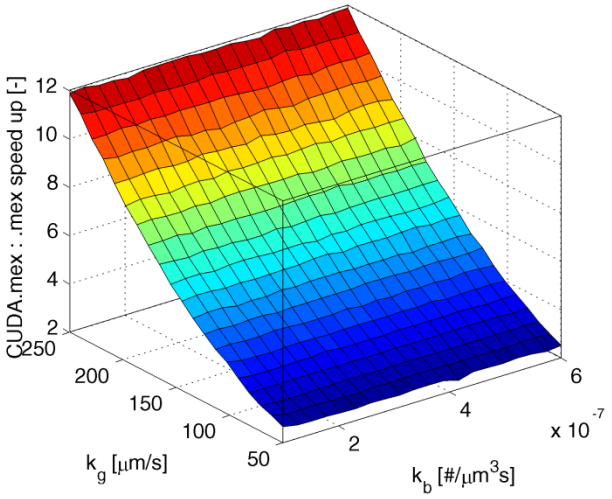
b)

Figure 9. a) The dependence of acceleration ratio and accuracy on the mesh size (*N*) in solution of a mono dimensional PBE with nucleation and size dependent growth (t_{max} = 400 s, *CFL* = 0.5)

b) The dependence of acceleration ratio and accuracy on the *CFL* criterion for size-independent growth and (t_{max} = 1,500 s, *N* = 18,000)



a)



b)

Figure 10. a) The dependence of .mex : MatLab acceleration ratio and b) CUDA .mex : .mex acceleration ratio on the nucleation rate constant k_b *and* growth rate constant k_g, N = 18000, t_{max} = 300 s.

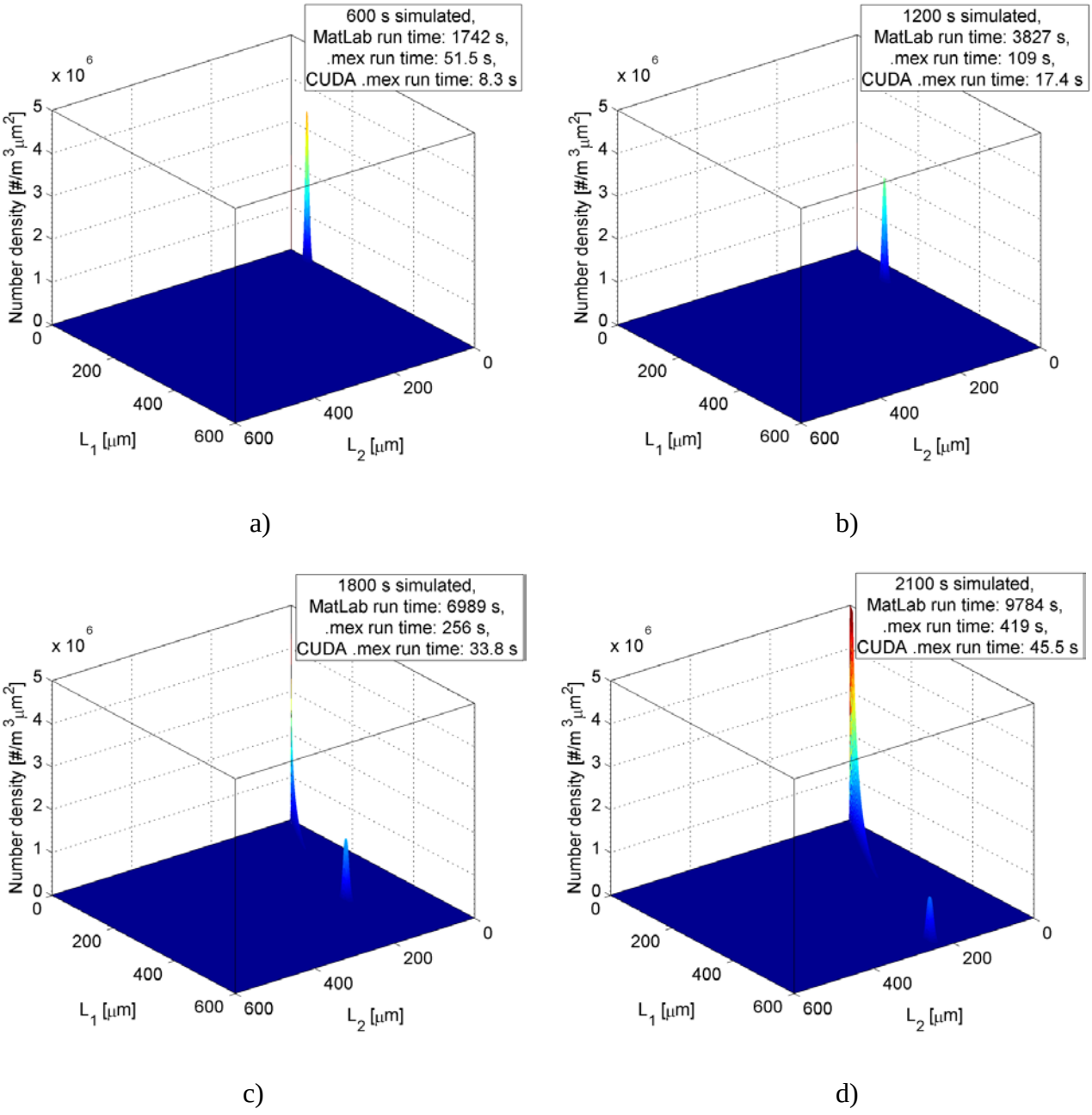
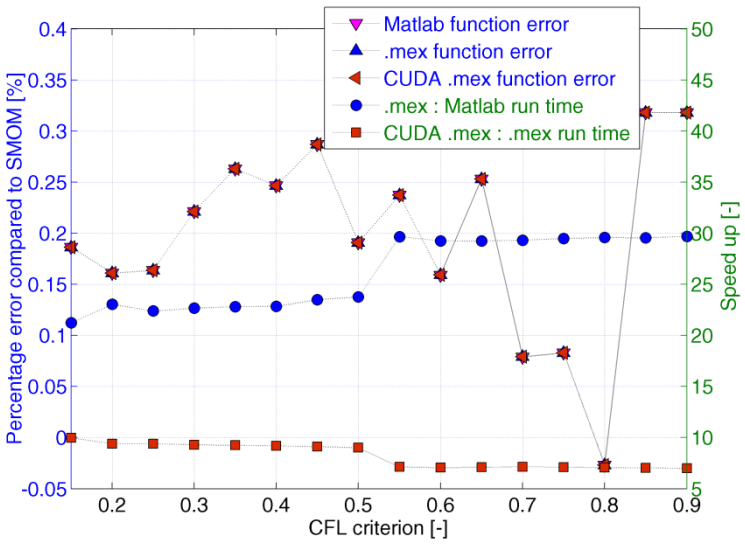
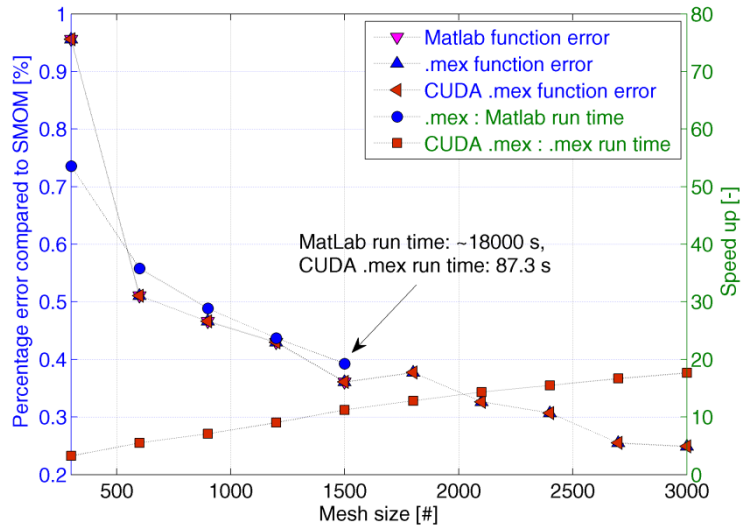


Figure 11. The calculated bivariate PSD's after a) 600, b) 1200, c) 1800 and d) 2100 seconds

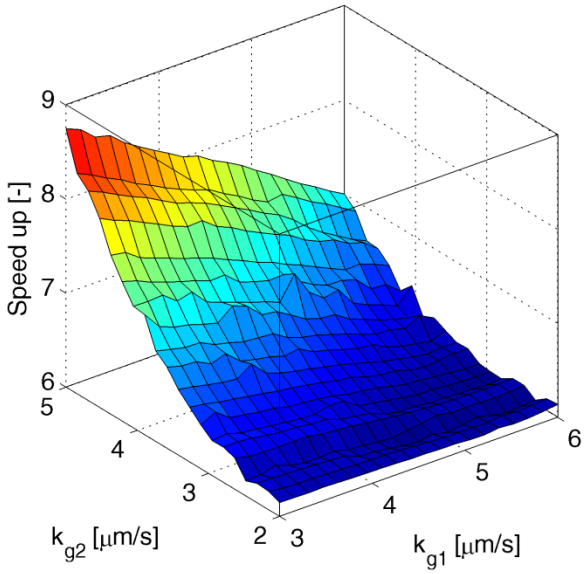


a)

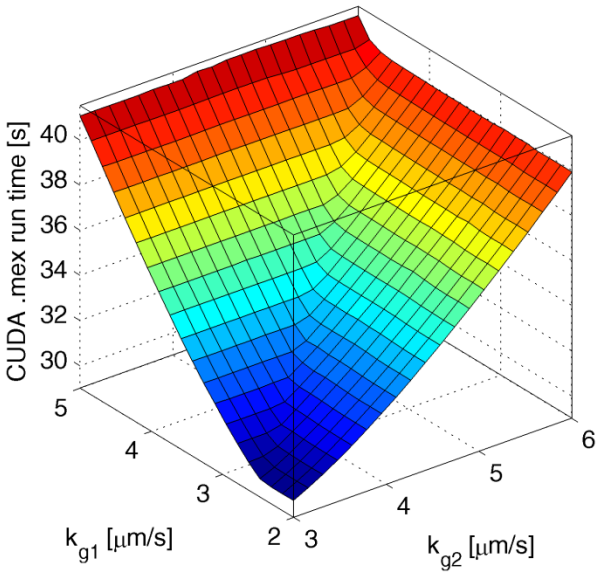


b)

Figure 12. Dependence of acceleration ratio and accuracy on the a) CFL criterion with size independent growth and b) division number in one direction (N) with size dependent growth in solution of a two dimensional PBE



a)



b)

Figure 13. a) Dependence of acceleration ratio and b) CUDA .mex function run time on growth rate constants (k_{g1} and k_{g2}) compared to the .mex code

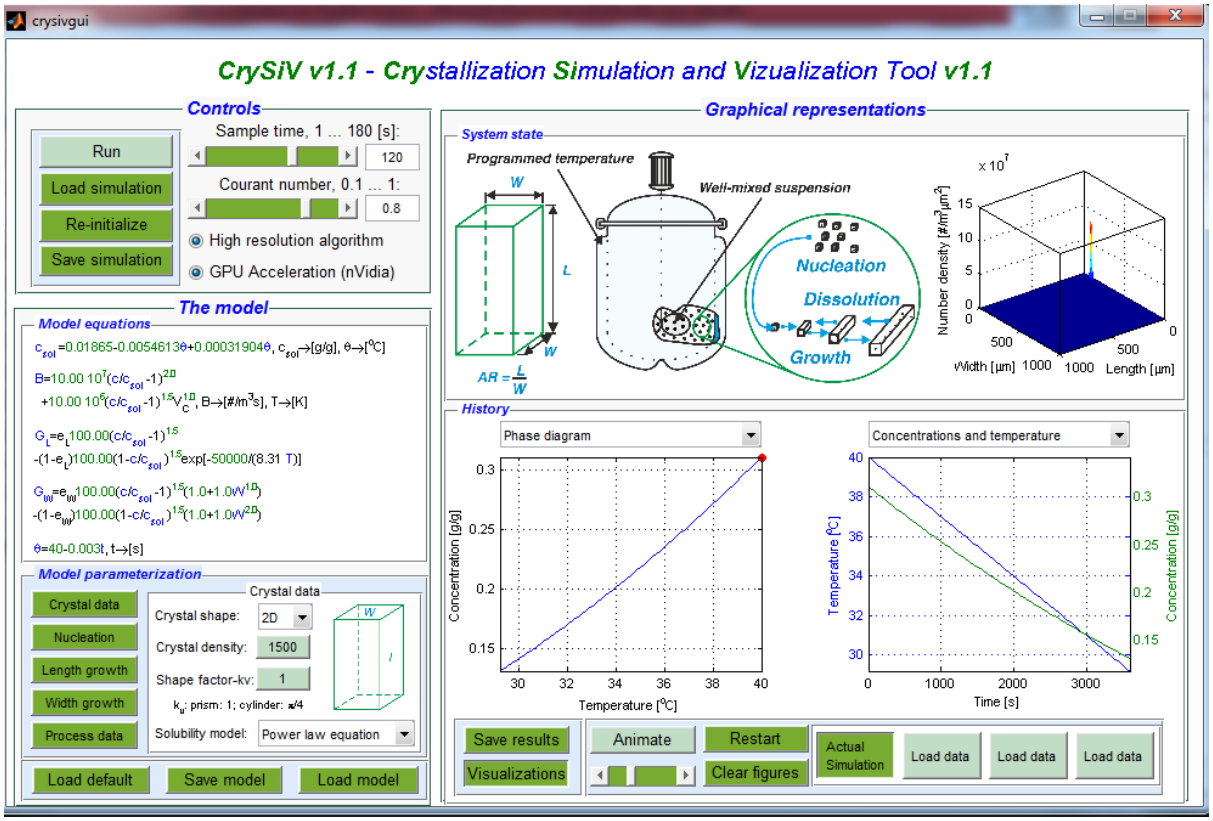


Figure 14. The interface of the *CrySiV* program, a MatLab based simulator for cooling batch crystallization processes