2024 Tenth Indian Control Conference (ICC)
9 - 11 December, 2024
IISER Bhopal, India

Accelerating the Multi-Objective Optimization of Crystallization Process using High Fidelity Population Balance Model Under NUMBA CUDA Environment

Anirudh Rangavajhala
Dept. of Chemical Engineering
Indian Institute of Technology, Hyderabad
Hyderabad, Telangana, India
anirudhr91@gmail.com

Aswitha Tadepalli
Dept. of Chemical Engineering
Indian Institute of Technology, Hyderabad
Hyderabad, Telangana, India
ch23resch11002@iith.ac.in

Dr. Kishalay Mitra*
Dept. of Chemical Engineering
Indian Institute of Technology, Hyderabad
Hyderabad, Telangana, India
kishalay@che.iith.ac.in

Crystallization is one of the majorly used separation techniques owing to its high efficiency and ease of operation. The optimization of the crystallization process involves the high-fidelity population balance models, thereby significantly increasing the computational complexity. This study, therefore, explores the acceleration of the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) using NUMBA CUDA for multi-objective optimization for crystallization of ortho-amino benzoic acid. By harnessing GPU parallel processing, significant reductions in computation time are achieved, particularly for large populations and complex optimization problems. The GPU-accelerated NSGA-II algorithm in the optimization of ortho amino benzoic acid crystallization demonstrates significant performance boost by reducing the time by 122 times without any reduction in the solution quality. This research, therefore, underscores the potential of GPU acceleration in evolutionary algorithms and sets the stage for further advancements in high-performance parallelized optimization techniques, promising significant improvements in real-time optimization scenarios.

***Keywords— Non-Dominated Sorting Genetic Algorithm, Multi-Objective Optimization, Crystallization, Graphical Processing Unit, Parallelization*, NUMBA CUDA.**

I. INTRODUCTION

Crystallization is a critical process widely utilized in various industries, including pharmaceuticals and fine chemicals, for the separation and purification of products. In the pharmaceutical industry, crystallization is often employed to purify active pharmaceutical ingredients (APIs) from solutions. This separation technique is favored because it can achieve high purity levels and control the crystal size distribution (CSD), both of which are crucial for the efficacy and safety of pharmaceutical products. Traditionally, batch crystallization has been the predominant method; however, continuous crystallization processes have gained traction due to their ability to improve the yield, enhance product quality and consistency, and reduce operational costs [1].

Among continuous crystallization systems, Mixed-Suspension Mixed-Product Removal (MSMPR) crystallizers are particularly notable for their operational simplicity and adaptability. MSMPR crystallizers allow better control of the crystallization environment, leading to more uniform CSDs and higher product purity. Despite these advantages, optimizing MSMPR crystallization processes remains a complex multi-objective problem that involves balancing trade-offs among various parameters such as residence time, supersaturation, and temperature. To address these challenges, researchers have developed high-fidelity population balance models (PBMs) that describe the dynamics of crystal growth and nucleation within crystallizers. These models provide a detailed understanding of the crystallization process, enabling the optimization of operational conditions to achieve desired product characteristics. However, the computational complexity and time required to solve PBMs make real-time optimization challenging [2].

Although these high-fidelity models are highly accurate and can be validated, they come with significant computational complexity and optimizing MSMPR crystallization systems using these models is very time-consuming. This is due to the iterative nature of optimization algorithms, which need repeated model evaluations. The challenge becomes even more difficult when the problem involves conflicting objectives. One way to get around this problem is using surrogate models to determine the fitness values instead of solving multiple complex differential equations and leveraging these surrogate models in the optimization process [3]. But one of the main disadvantages of using these models is that they come at a cost of accuracy, which may lead to misleading results. They also risk the problem of over or under sampling which can lead to inaccurate models and suboptimal optimization results.

Surrogate models may also struggle with high-dimensional optimization problems due to the curse of dimensionality. As the number of dimensions increases, the amount of data required to build an accurate surrogate model grows exponentially. Another way to optimize these high-fidelity models in a very quick and efficient manner without compromising the accuracy is to employ the concept of parallel computing in population based Evolutionary Algorithms. Finding optimal values using Genetic Algorithms (GAs) involves calculating the fitness values of a set of individuals which are calculated without any dependency on the other individuals. This fact can be used to perform these calculations parallelly by assigning each thread in the Graphical Processing Unit (GPU). The crossover, mutation operations on these individuals can also be performed parallelly in the same manner. Overall, the cores of GPU can be exploited for parallel computing in many steps of the optimization process using the high-fidelity models thereby significantly reducing the time complexity. One of the popularly used algorithm among the GAs for multi-objective optimization problems (MOOP) is the Non-Dominated Sorting Genetic Algorithm (NSGA-II) [4]. Thus, to solve the MOOP, the idea of parallel processing using GPUs can be employed to speed up the execution time of NSGA-II algorithm.

Aguilar-Rivera *et.al*., attempted to parallelize of NSGA-II algorithm on GPU using the ArrayFire Library in Julia [5]. The algorithm was entirely vectorized, and stochastic variants of grid-crowding and non-domination sorting are introduced. This allows for large speedups without sacrificing the quality of the answer. They also reported the challenges in parallelizing Genetic Algorithms such as managing data dependencies and make sure parallel threads are properly synchronized in order to prevent race situations and guarantee accurate results, uniformly distributing the workload among the parallel processing units in order to maximize usage and coordinating tasks and sharing data between parallel processing units for effective communication. In parallel algorithms, minimizing communication cost is essential for attaining good performance.

Although attempts to parallelize the NSGA-II algorithm have been made before, the full-scale parallelization of the algorithm in python code using NUMBA CUDA has not been explored. Numba is an open source Just-In-Time (JIT) compiler that translates interpreted code into machine language and it uses Compute Unified Device Architecture (CUDA), which is a parallel computing platform developed by NVIDIA to leverage GPU capabilities to translate a portion of Python and NumPy code into machine code [6]. CUDA lets programmers create programs that run on the GPU using extensions in the form of a few fundamental keywords added to C, C++, and Fortran. Together, NUMBA CUDA offers a straightforward approach to develop GPU-accelerated Python functions that are and can target both CPU and GPU hardware with significant speedups.

This study, thus, aims to parallelize the NSGA-II algorithm using NUMBA CUDA to accelerate the MOOP of the crystallization process using the high fidelity PBM. This work firstly involves developing the entire NSGA II algorithm in Python despite the fact that it is included in the "**pymoo**" library of python, as the custom code offers more control at each operation of the algorithm especially in the steps where crossovers and mutations are involved. Then, this algorithm is modified to run on GPU asynchronously using NUMBA CUDA. This GPU enabled NSGA-II code is finally leveraged in the optimization process of the crystallization process.

The rest of the paper is organized as follows. In Section II, the MOOP of the crystallization process and the methodology of the parallelizing NSGA-II using NUMBA CUDA is presented. In Section III, the results of the optimization problem are discussed and finally in Section IV, the conclusions of the study are summarized.

II. Methodology

A. Multi objective optimization formulation of the crystallization of Ortho-amino Benzoic acid in MSMPR

The crystallization process of ortho-amino benzoic acid is carried out in a three-stage cascade MSMPR crystallizer with recycling, based on a mathematical model inspired by existing literature. This model assumes fixed-size secondary nucleation, size-independent growth, and uniform breakage. The function n(L, t) represents the number of crystals produced within a size range of L to L + dL per unit volume of slurry at any given time t. The one-dimensional population balance equation (PBE) for the i^{th} MSMPR with recycle R is described by (1).

$$\frac{\partial n_i(L,t)}{\partial t} + G_i \frac{\partial n_i(L,t)}{\partial L} = B_i \delta(L - L_n) + \left(B_{bre,i}(L) - D_{bre,i}(L)\right) + \frac{(1-R_i) n_{i-1}(L,t) + R_i (n_{rec}(L,t)) - n_i(L,t)}{\tau_i} \quad (1)$$

where G is the SI growth rate, B is the secondary nucleation rate, L_n is the size of the nucleons, B_{bre} represents the crystals' birth rate, D_{bre} represents the crystals' death rate due to breakage, δ is the Dirac delta function, n_{rec} is the crystal size density of the recycle stream, and τ is the slurry's total residence time in MSMPR. The equations for GGG and BBB involve the kinetic parameters given in (2) and (3) respectively. All kinetic parameters for ortho-amino benzoic acid used in the model are taken from literature [7]. In (1), the left-hand side describes the evolution of CSD (first term) and crystal growth (second term), while the right-hand side accounts for nucleation with uniform breakage.

$$G_i = \begin{cases} k_g \sigma_i^g & \text{if } c \geq c_{sat} \\ -k_d \sigma_i^d & \text{if } c < c_{sat} \end{cases} \quad (2)$$

$$B_i = \begin{cases} k_b \sigma_i^b & \text{if } c \geq c_{sat} \\ 0 & \text{if } c < c_{sat} \end{cases} \quad (3)$$

Here, c represents the concentration of the solute, while c_{sat} denotes the solubility, which is expressed as a second-order polynomial function of temperature T [as per (4)]. The

symbol σ represents the relative supersaturation, which is calculated using (5).

$$c_{i,sat} = a_0 + a_1\, T_i + a_2\, T_i^2 \quad (4)$$

$$\sigma_i = \left|1 - \frac{c_i}{c_{i,\,sat}}\right| \quad (5)$$

The time-dependent temperature T is determined fro m an exponentially decreasing temperature profile until a steady state is reached, as described in (6).

$$T_i = T_{i-1} - \theta_i\left(1 - e^{-\frac{t}{100}}\right) \quad (6)$$

where θ denotes the temperature decrease.

In (1), B_{bre} represents the birth rate of crystals of size L resulting from the breakage of crystals of size λ, as calculated using (7).

$$B_{bre,i}(L) = \int_{\lambda}^{L_{max}} b\left(\frac{L}{\lambda}\right) b_0\, S_0(\lambda) n_i(\lambda, t) d\lambda \quad (7)$$

In this context, $b\left(\frac{L}{\lambda}\right)$ is the distribution function of daughter crystals, which describes the probability that a crystal of size λ will break into a crystal of size L, and it is calculated using (8) under the assumption of a uniform distribution. $S(\lambda)$ is a function that relates crystal size to breakage, following a power law as specified in Eq. (9), and L_{max} represents the maximum crystal size.

$$b\left(\frac{L}{\lambda}\right) = \frac{2}{\lambda} \quad (8)$$

$$S_0(\lambda) = \lambda^k \quad (9)$$

The breakage of L size crystals is presented by D_{bre} [(1)] and is calculated using (10).

$$D_{bre,i}(L) = b_0\, S_0(L) n_i(L, t) \quad (10)$$

In (7), (9) and (10), the subscript "0" indicates that the property is calculated at the feed stream. Finally, the mass balance equation is given by (11)

$$\frac{\partial c_i}{\partial t} = -\rho_c k_v \left[B_i L_n^3 + 3G_i \int_0^{L_{max}} L^2 n_i(L,t) dL\right] + \frac{(1 - R_i)c_{i-1} + (R_i)c_{rec} - c_i}{\tau_i} \quad (11)$$

where L is the internal coordinate, c_{rec} is the recycle stream concentration, k_v is the volume shape factor, and ρ_c is the density of the crystals. The subscript "i" in all the aforementioned equations represents the "ith" MSMPR.

The Partial Differential Equations (PDEs) involved in the PB equations are numerically solved using the FVM-HR technique. This method converts the PDEs into ordinary differential equations (ODEs) by discretizing the length of the crystals to a certain degree. These ODEs are then solved using the Euler method. The input set for this ODE model includes the initial temperature of crystallizer-1 T_0, temperature decrease parameters $\theta_1, \theta_2, \theta_3$, residence times τ_1, τ_2, τ_3, and recycle ratios R_1, R_2, R_3 for all three crystallizers, totaling 10 inputs. The outputs, which are obtained by solving the ODEs, include the total residence time τ_{total} , CSD spread $\bar{\sigma}$, and nominal crystal size L.

The optimization of Crystallization process of Ortho amino benzoic acid using MSMPR process using NSGA-II is considered [7]. In order to obtain a better product quality, three objective functions are considered, which includes minimizing the spread of the crystal size distribution (CSD), which impacts the uniformity of the final product, minimizing the total residence time of the crystals in the reactor, which can improve productivity and maximizing the nominal crystal size, which can enhance product quality. The MOOP is given in (12). Since this is a multi-objective optimization problem, NSGA-II algorithm can be utilized to obtain the Pareto optimal solutions.

Objectives: $$\underset{T_0,\, \theta_i,\, \tau_i,\, R_i}{Min} \ \overline{-L},\ \tau_{total},\ \sigma \ \forall\, i = 1, 2, 3 \quad (12)$$

Equation (12) is subject to Crystallization PB equations. Here i corresponds to each one of the three MSMPR Crystallizers, $\bar{L}$, is nominal crystal size, τ_{total} is the total residence time of all the three MSMPR Crystallizers and σ is the SD spread. The negative sign before $\bar{L}$ in equation (4) is because they have to be maximized.

The decision variables and their lower and upper bounds used in the optimization of the crystallization of ortho amino benzoic acid are listed in Table I.

TABLE I. DECISION VARIABLES AND THEIR BOUNDS.

Decision Variable	**Lower Bound**	**Upper bound**
First MSMPR Temperature (T_0)	31^{0}C	36^{0}C
Decrease in temperature in each MSMPR while cooling (θ)	1^{0}C	4^{0}C
Recycle ratio of each MSMPR (R)	0	0.3
Residence time of each MSMPR (τ)	200 s	360 s

B. Parallelization of NSGA-II using NUMBA CUDA

The entire NSGA-II algorithm is written in python from scratch to facilitate it to run on GPU using NUMBA CUDA. A population class is created that has instances of the class individuals as its attributes. The individuals have the attributes of genes, binary variables, real variables, rank of the individual and its dominated set. The entire algorithm is divided into functions which predominantly deal with initiating the population with genes and real values, decoding the genes into real values, calculating the fitness values, non-dominated sorting, crossovers and mutations.

Each of the above mentioned functions work at the individual level, and when the values of attributes of each of these individuals are being updated, they mostly occur independently of other individuals. This creates an opportunity to exploit the concept of parallel processing especially on the GPUs which have many cores [8]. Since, the GPU has many cores, the calculation of the all attributes of an individual inside the population can be assigned to a thread and multiple threads perform multiple calculations on these individuals asynchronously which leads to massive parallelization thereby significantly reducing the execution time.

The functions of the algorithm are enabled to run on GPU using the decorators on the function from NUMBA CUDA [9]. A decorator is a design pattern in Python that allows to change a function, method, or class's behaviour dynamically without altering the source code. Decorators accept an argument from another function or procedure and are implemented as functions in and of themselves. Usually, they return a new function or alter the current one to include new behaviour or capability. The decorators commonly used to decorate a function in NUMBA CUDA are **"@vectorize"**, **"@guvectorize"** and **"@cuda.jit"** decorators. The @vectorize decorator makes it easier to create element-wise operations that can be applied to arrays. The functions that are decorated with @guvectorize decorator support broadcasting and can act on full arrays (or subarrays), allowing for multidimensional, sophisticated operations. High-performance GPU code can be built directly by defining custom CUDA kernels and device functions which involve calculations that are more complex than vectorized ones in Python using the @cuda.jit decorator found in Numba's CUDA module. With the help of this decorator, CUDA code may be written inside of the Python environment and take advantage of the computing power of NVIDIA GPUs in a flexible manner.

Inside the decorated function, the iterables in loops and nested-for loops are replaced with **"cuda.grid(x)"** where x is the number of dimensions of the dataset to be operated upon. This command divides the task done on each data element under a for loop among threads so that they can be run in parallel so long as there is no dependency between the data elements and the rest of the code remains virtually same. Since, there are minimal changes to be done to the existing function, NUMBA CUDA can be preferred to accelerate python code on GPU. One more useful tool for optimizing scientific and numerical Python computations is Numba's **@njit** decorator. @njit offers notable speed gains by converting functions into machine code while preserving the readability and simplicity of Python programming. Compatibility with NumPy and support for parallel execution make @njit a vital tool for high-performance Python development. Using the @njit decorator with the parallel=True option and Numba's '**prange**', which may be used in place of the 'range' keyword in for loops to run for loops parallelly, Numba supports for parallel execution on the CPU. Using these decorators and in-built functions offered by NUMBA CUDA, the functions of the NSGA-II algorithm can be run on GPU allowing the computations inside these functions to be run parallelly.

III. Results and Discussions

The development of NSGA-II, its parallelization in Python were done in NVIDIA Corporation GV100GL [Tesla V100 PCIe 16GB] configuration. The *Numpy, Numba* packages in PYTHON were used for the same.

To solve the MOOP of the crystallization of ortho amino benzoic acid presented in Section II, the NSGA-II algorithm was initially executed for 200 population and 100 generations. The development of the Pareto front in this case is plotted in Fig. 1.

However, the calculation of fitness values viz., the size of the crystals, spread of CSD and total time for each individual by solving the differential equations presented in Section II.A using Euler method was tremendously time-consuming. From the results, it was seen that each generation took around approximately 1500 seconds which is too high when compared with other functions of the NSGA-II algorithm. So, the execution time of entire NSGA-II algorithm predominantly depends upon the execution time of the calculation of fitness values of individuals.

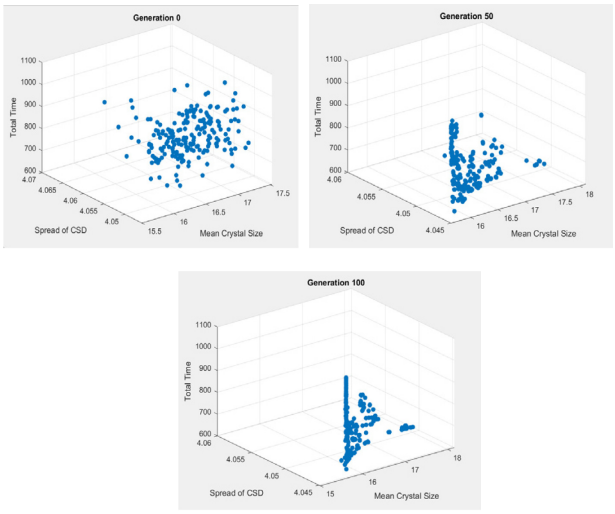


Figure 1: Development of the Pareto front of the crystallization MOOP for 100 generations.

Thus, to accelerate this function and ultimately speed up the optimization process, the parallelization of NSGA-II was performed using the NUMBA CUDA. The "func_cuda" function in the parallelized NSGA-II code calculates the fitness values of the individuals asynchronously thereby decreasing the execution time of the algorithm significantly. The parallelized NSGA-II was also executed for 200 population and 100 generations. The development of the Pareto front in this case is plotted in Fig. 2.

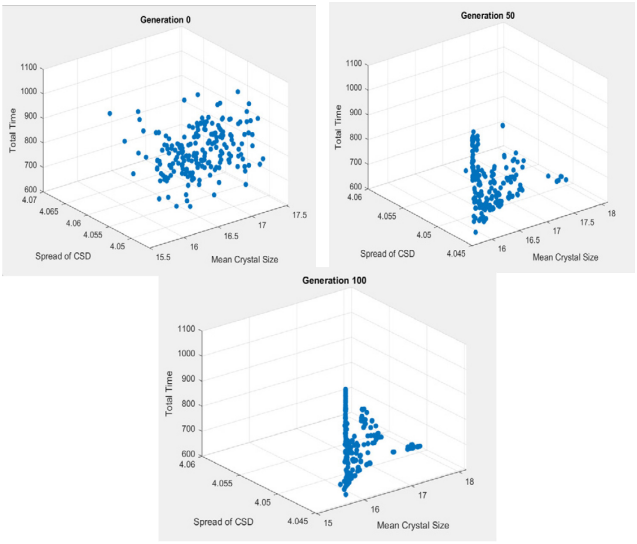


Figure 2: Development of the Pareto front of the crystallization MOOP for 100 generations using parallelized NSGA-II.

In this case, the time taken for each generation was around 10 seconds. Thus, it can be seen that the execution time for one generation has reduced by 150 times. The execution times for 100 generations of NSGA-II code that uses regular function and the parallelized NSGA-II code that uses the CUDA decorated function is given in the Table II.

TABLE II. PERFORMANCE COMPARISON OF NSGA-II CODE TO SOLVE THE MOOP OF CRYSTALLIZATION PROCESS WITH AND WITHOUT PARALLELIZATION

	Without Parallelization (s)	**With Parallelization (s)**
Execution Time	163859.53	1333.91

As it can be inferred from the Table II, the execution time of parallelized NSGA-II function with the "func_cuda" function runs 122.84 faster than the regular NSGA-II code that has "func" function which is a significant speed-up. Also, from the Fig. 1 and 2, it can be seen that at various generations, the Pareto fronts in both the cases are exactly the same. Thus, the time complexity of the crystallization MOOP using the high-fidelity PBM through NSGA-II algorithm was notably reduced by 99% without actually compromising on the solution. Such kind of applications can be run in an advisory mode, where from the set point (one of the points from the Pareto set) can be downloaded to a controller, which can in turn control the crystallization process around the set point using standard control algorithms such as model predictive control.

IV. Conclusions

This work demonstrates the potential of leveraging GPU parallel processing using NUMBA CUDA to accelerate the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) for multi-objective optimization of crystallization process of ortho amino benzoic acid. By addressing the computational challenges in traditional optimization methods, the study showcases significant reductions in computation time, especially for large populations and complex problems, without compromising the quality of solutions. In the optimization of Crystallization process of Ortho amino Benzoic acid, the benchmarking against CPU-based implementations revealed substantial speed improvements, with GPU-accelerated functions achieving up to 122.84 times faster performance. This application can be helpful for advisory level control of crystallization process which can be utilized for downloading set points to the controller controlling a crystallization process. This advancement in GPU-accelerated optimization sets the stage for broader industrial applications and further exploration of high-performance parallelized optimization techniques, emphasizing the transformative potential of such approaches in real-time optimization scenarios.

Acknowledgement: Authors acknowledge the support provided by the project # DST/NSM/R&D_HPC_Applications/2021/23 funded by the Department of Science and Technology (DST), Government of India, under the National Supercomputing Mission for this work.

References

[1] Orehek, J.; Teslić, D.; Likozar, B. Continuous Crystallization Processes in Pharmaceutical Manufacturing: A Review. Org. Process Res Dev. 2021, 25(1), 16–42. DOI: 10.1021/acs.oprd. 0c00398.

[2] Wang, T.; Lu, H.; Wang, J.; Xiao, Y.; Zhou, Y.; Bao, Y.; Hao, H. Recent Progress of Continuous Crystallization. J. Ind. Eng. Chem. 2017, 54, 14–29. DOI: 10.1016/j.jiec.2017.06.009.

[3] Inapakurthi, Ravi kiran, Sakshi Sushant Naik, and Kishalay Mitra. "Toward faster operational optimization of cascaded MSMPR crystallizers using multiobjective support vector regression." *Industrial & Engineering Chemistry Research* 61, no. 31 (2022): 11518-11533.

[4] Deb, K., A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation* 6, no. 2 (2002): 182-197.

[5] Aguilar-Rivera, Anton. "A GPU fully vectorized approach to accelerate performance of NSGA-2 based on stochastic non-domination sorting and grid-crowding." *Applied Soft Computing* 88 (2020): 106047.

[6] Lam, Siu Kwan, Antoine Pitrou, and Stanley Seibert. "Numba: A llvm-based python jit compiler."

In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1-6. 2015.

[7] Tadepalli, Aswitha, Keerthi NagaSree Pujari, and Kishalay Mitra. "A crystallization case study toward optimization of expensive to evaluate mathematical models using Bayesian approach." *Materials and Manufacturing Processes* 38, no. 16 (2023): 2127-2134.

[8] Owens, John D., Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips. "GPU computing." *Proceedings of the IEEE* 96, no. 5 (2008): 879-899.

[9] Oden, Lena, and Tarek Saidi. "Implementation and Evaluation of CUDA-Unified Memory in Numba." In *Euro-Par 2020: Parallel Processing Workshops: Euro-Par 2020 International Workshops, Warsaw, Poland, August 24–25, 2020, Revised Selected Papers 26*, pp. 197-208. Springer International Publishing, 2021.