Parallelisation Strategies for Large Scale Cellular Automata Frameworks in Pharmaceutical Modelling

Marija Bezbradica, Martin Crane and Heather J. Ruskin
Centre for Scientific Computing and Complex Systems Modelling, School of Computing
Dublin City University, Dublin 9, Ireland
{*mbezbradica, martin.crane, hruskin*}*@computing.dcu.ie*

ABSTRACT

Cellular Automata (CA) properties facilitate the detail required for the bottom-up approach to modelling and simulation of a broad range of physico-chemical reactions. In pharmaceutical applications, CA models use a combination of discrete-event rules based on probabilistic distributions and fundamental physical laws to predict the behaviour of active substances (drug molecules) and structural changes in Drug Dissolution Systems (DDS) over time. Several models of this type have been described so far in the scientific literature. Yet, practical applications are lacking in the context of large-scale, high-precision, high-fidelity simulations. The key obstacle to parallelisation of such models is not only the amount of data involved, but also the fact that many of these models incorporate agent-like behaviour within the CA framework in order to describe pharmaceutical components. This makes communication across process boundaries expensive. In this paper, we apply different parallelisation strategies to a large scale CA framework, used to model coated drug spheres. We use two parallel-computing application programming interfaces (APIs), namely OpenMP and MPI, to partition the simulation space. We analyse the applicability of each API to the problem individually, as well as in the hybrid solution. We examine speedup potential and overhead for local and global communication for simulation speed and solution scalability. For these types of problems, our results show that performance is much improved for appropriate combinations of parallelisation solutions.

KEYWORDS: Cellular Automata, Modelling, Hybrid models, HPC, Spatio-temporal model

1. INTRODUCTION

As modern drug formulations become more advanced, pharmaceutical companies face the need for adequate tools to help them to satisfy complex requirements, such as reduction of unnecessary adsorption rates, by allowing them to model release kinetics of controlled and targeted drug release [6]. The design needs to consider a plethora of chemical interactions, for which information is partial with details often inferred only through observing superposition of underlying phenomena. This is particularly the case when knowledge of the DDS is incomplete. Thus, probabilistic computational modelling is crucial to the design of such experiments, both in terms of reducing high experimental costs and in predicting system-dependent changes.

Following initial use of CA in probabilistic models in the pharmaceutical context [12], models developed to date are numerous [1], [4], [7], [11], [13] and provide several possible perspectives on pharmaco-kinetic system representation in CA space. Most are limited to 2D geometries, with only occasional consideration of the complexities of 3D space. The CA approach itself is applicable to a wide range of pharmaceutical systems, without requiring detailed initial knowledge of dissolution mechanisms. For this model type to make high fidelity predictions, (based on underlying structural behaviour), while dealing with a broad range of simulation parameter values, such as drug diffusion coefficients, (which significantly affect times needed for simulation), their complexity and size must increase. In the present high performance computing era, the resources to process such models efficiently are readily available, but algorithmic solutions in this field tend to lag behind. The key problem to solve is the expense in terms of time and data load of cross-process communication present in parallelisation of such models, a consequence of the fact that many of them incorporate agent-like behaviour within the

CA framework [2].

In this paper, we apply different parallelisation strategies to an existing CA framework, used to model novel, controlled drug formulations such as coated drug spheres [3]. In controlled drug design, one of the main factors influencing drug dissolution is sensitivity to the physical thickness of spherical coatings. This is usually orders of magnitude smaller than the core sphere radius, so that large 3D models are needed to investigate different scales.

The following sections deal with: an overview of the model dynamics with a short introduction to dissolution phenomena (Section 2); Section 3 discusses implementation of different parallelisation algorithms on the model space, while Section 4 provides results of the performance tests. Section 5 summarises conclusions and outlines possible future directions.

2. MODEL DYNAMICS

The essence of probabilistic models in pharmaceutical modelling is the representation of different dissolution phenomena, such as erosion (polymer chain disentanglement), swelling (polymer chain expansion) or drug diffusion (polymer movement between regions of different concentration), through assignment of probability values to possible outcomes. Assigned probabilities are tied to the description of the *relative speed* of the modelled phenomena and are usually derived from information on diffusion coefficients of different materials.

The basis of our model can be briefly described in terms of the following criteria:

- Three-dimensional discrete space allows for models of various geometries and device compositions
- Internal state variations are set randomly according to the desired probability distribution using Direct Monte Carlo methods: this, since local distributions of device features, e.g. drug density or polymer composition at a given site, are unknown
- Different dissolution phenomena are modelled independently: we can separately investigate the influence of polymer swelling while ignoring coating erosion and vice-versa
- Efficient large scale simulation: this, in order to explore molecular and/or small-scale level effects
- This specific model implementation simulates the dissolution of a single bead represented in a three-dimensional space. The space itself is represented as a lattice, divided into discrete cells, with each being described by a compound state $\Psi_{(i,j,k,t)}$ depending on the aggregate condition of the bead structure at discrete space coordinates *(i, j, k)* and at a discrete point in time *(t)*. Cellular Automata rules of the type:

$$\Phi(\sigma) : \Psi_{(i,j,k,t)} \rightarrow \Psi_{(i*,j*,k*,t+\Delta t)} \qquad (1)$$

  are defined for each cell and are applied to the compound state (Ψ) of either the current cell - *(i*, j*, k*)* $\equiv$ *(i, j, k)* - or any eligible cell within the Moore neighbourhood - *(i*, j*, k*)* $\equiv$ *(i$\pm$1, j$\pm$1, k$\pm$1)* - in order to re-evaluate it after each time increment (Δt) of the simulation. When a particular rule is applied to the cell state, it considers the state of all neighbouring cells (σ) in the 3D space (26 in all, corresponding to a 3D Moore neighbourhood), before defining its effect on that particular cell or on its neighbour. The rules applied can model any distinct process that causes change in the bead state, or its transfer to the neighbouring cell, including drug diffusion, chain-scission of polymers, erosion or swelling of device core.

The bead itself is represented as an idealised sphere, constructed of two layers - a coating layer which consists of entangled polymer chains and a core layer, which contains the drug molecules dispersed within the polymeric carrier. The possible states of each cell represent different structural forms of the bead that affect the behaviour of those molecules as they diffuse out of the sphere (Figure 1).

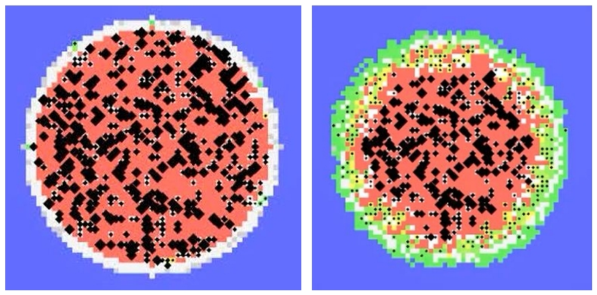


Figure 1: Visualisation of Different Dissolution Stages in the CA Framework. **(Left)** Initial Coating and Undegraded Core; **(Right)** Coating Polymer Chain Disentanglement and Expansion due to Water Influx.

3. PARALLELISATION SCHEMES

In order to develop an efficient parallelisation algorithm for the CA model presented, we need to understand first the fundamental types of cell communications and cell transfers occurring within it. Traditional synchronous CA models usually follow a standard pattern where a cell state is passively dependent on its neighbours and, based on their state values, is updated itself in the following iteration. These kinds of models should fit into the category of so called ”embarrassingly parallel” problems, where parallelisation of the problem space is straightforward [8]. However, differences in the rule types in our case mean that certain types of shared state and synchronisation between different parallel processes is necessary.

3.1. Rule Types

In the presented model, each cell has a state describing the current process of several different physico-chemical phenomena:

- Amount of degradation of polymer chains
- Amount of expansion potential of polymer chains
- Amount of drug particles present in the cell

These sub-states are mutually independent and can be updated separately from each other. Thus, we cannot treat whole cell state updates as ”atomic”, only sub-state ones. Therefore, every single cell has potential to be updated several times during a single simulation iteration.

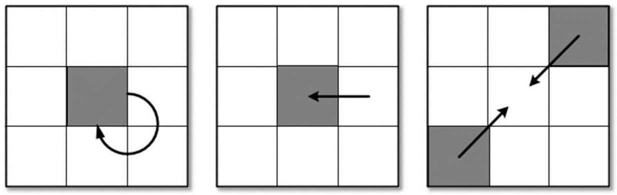


Figure 2: Elementary Types of CA Rules Present in the Model.

Model rules that affect those sub-states can generally be classified into three fundamental types (Figure 2):

- Temporal rules which change the state of a single cell without neighbour influence. This type of rule is always deterministic (e.g. polymer degradation over time)
- Neighbourhood-dependent rules which change the state of a single cell, based on the combination of states from neighbouring cells. This type of rule can either be deterministic (polymer erosion) or probabilistic (polymer wetting)
- Neighbourhood-dependent rules which change both the state of the cell and the state of one of its neighbours. This type of rule is always probabilistic (e.g. drug diffusion, polymer swelling).

The particular obstacle to overcome for parallelisation of the model to be efficient is effective communication of the third type of rule across parallelisation boundaries. These rules were introduced to realistically simulate the physico-chemical processes which have an active spatial span, as opposed to being passively affected by the neighbour state. However, the ability of the cell to affect the neighbour state in a probabilistic manner leads to the possible occurrence of multiple updates of a single cell from two or more of its neighbours (Figure 2). To resolve possible conflicts several approaches may be used:

- The neighbour cell can be marked as updated for all sub-states and no further state changes permitted
- The neighbour cell can be marked as updated for a single type of sub-state change (e.g. polymer swelling), but other types of state changes (e.g. drug diffusion) may be permitted
- The neighbour cell state can be updated multiple times and all types of state changes may be permitted. In this case, changes are considered to be additive.

In order to preserve physical process realism, the model generally assumes the third approach; however there can be cases where the neighbouring cell is precluded from a certain type of update (e.g. due to change of state). To illustrate the point let us consider two examples. In the first example, the drug diffusion algorithm selects the same target cell for drug packet movement from two different neighbours. In order to correctly describe the resulting movement, the destination cell will contain packets from both sources. In the second example, polymer expansion into the coating layer has caused a neighbour state change which prevents any further updates to the destination cell. In general, for the latter kind of updates, we assume a priority approach based on which cell caused the first change to the destination cell state. Any subsequent attempts to change the state are ignored. However, from the parallel implementation perspective, this implies both the need for cells to communicate their state across to that of the destination cell, as well as some kind of locking access to the destination cell during the process, in order not to cause partial or incorrect updates and thus loss of data.

3.2. Main Model Algorithm

The main model algorithm is implemented as multiple pass visits of each cell in the 3D matrix. The cells are visited in random order, using a shuffle approach, to avoid any potential bias or rule cycle. In the first pass, when a cell is visited, all possible behaviours defined for the particular cell type are evaluated, and applied based on calculated probabilities. The resulting state is then updated in the secondary matrix. When the first pass is finished, matrices are swapped, and a second pass is performed to calculate the resulting global state of interest to the modellers, such as new dissolution boundary radii, and overall amount of drug dissolved. Since only one pass occurs at any one time, there are no conflicts in cell updates. The algorithm pseudo-code is described as Algorithm 1:

Algorithm 1 *Update*(Matrix M)

```
1: Secondary = M
2: shuffled_cell_vector = Shuffle(M)
3:     for each c in shuffled_cell_vector do
4:         for each rule in rule_list do
5:             Apply(c, rule) ⇒ Secondary
6: M = Secondary
7: ComputeRelease(M)
```

3.3. Thread-level Parallelism

The first approach we consider, for solution of the efficiency problem, involves thread-level parallelisation on a shared multi-processing (SMP) architecture. The main matrix is kept as is, and the secondary matrix is divided into logical regions of equal height, where updates in each region are controlled from a separate execution thread (Figure 3). Since the matrix state is shared in memory and visible to all threads, we now have the potential for conflicts in cell state updates. In order to minimise the negative effects of explicit thread synchronisation, cell state locking is considered only at region boundaries. Namely, two boundary layers on each side of the region, except in the case of the first and last region, are considered to be ”shared” and locking is enforced for any state changes in that region during the single update period.

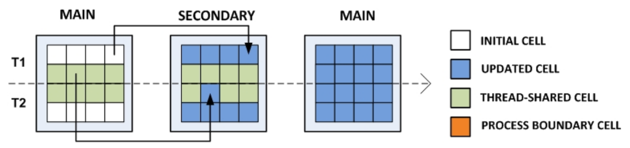


Figure 3: Parallelisation Strategies - OpenMP.

However, other threads are permitted to make any updates that are considered legal by the rule set, after the lock is lifted. The pseudo-code for the algorithm modification is given as Algorithm 2. The standard OpenMP API primitives have been used as a means of implementing the parallelisation of the ”for” loops in the main algorithm.

Although the approach provides a straightforward algorithmic solution to the problem, SMP architectures are generally limited by the number of available execution cores, and performance gain from this type of parallelisation has a ”hard ceiling” set by the underlying architecture. Hence, in order to improve the execution times further, we have to cross the process and machine boundaries.

Algorithm 2 *UpdateByThreads*(Matrix M)

```
1: Secondary = M
2: parallel for each region in M.Regions
3:     shuffled_cell_vector = Shuffle(region)
4:     for each c in shuffled_cell_vector do
5:         if(IsShared(c) == true) then lock(Secondary(c))
6:         for each rule in rule_list do
7:             Apply(c, rule) ⇒ Secondary(region)
8:         if (IsShared(c) == true) then unlock(Secondary(c))
9: #end parallel
10: M = Secondary
11: ComputeRelease(M)
```

3.4. Process-level Parallelism

In order to split the algorithm execution across the different execution nodes, the focus shifts to solving the communication problem instead of the synchronisation one. In distributed computing platforms, such as MPI, a state has to be explicitly shared by sending and receiving over the communication network. This network now introduces an additional variable in the model performance, and introduces the need to define a communication pattern between different model regions.

Algorithm 3 *CrossProcessUpdate*(Matrix M, int rank)

```
1: Secondary = M
2: n = Secondary.Regions.Count
3: Matrix region
4: if(rank == MASTER) then
5:   Scatter(M.Regions) ⇒ region
6: for each c in Shuffle(region) do
7:   if(IsBoundary(c) == true) then skip
8:   ApplyRules(c, rule_list) ⇒ Secondary(region)
9: ExchangeBoundary(rank, rank + 1)
10: for each c in Shuffle(region.boundary) do
11:   ApplyRules(c, rule_list) ⇒
12:       Secondary(region.boundary)
13: if(rank == MASTER) then begin
14:   Gather(M.Regions) ⇐ Secondary(region)
15:   ComputeRelease(M)
16: end
```

Generally, CA model space can be efficiently sent to separate execution nodes using a ”scatter-gather” algorithm. Regions of equal size are sent (scattered) from a ”master” process, responsible for overall state management, to two or more ”slave” processes, which are solely responsible for the simulation of a single region. After the regions have been simulated, their state is returned (gathered) back by the master process, which has the responsibility of assembling them back together and calculating the resulting output. It is interesting to note that, due to the fact that the master process is idle during the period between the scatter and gather phases, it also assumes the role of the slave, by assigning itself as owner of one of the regions to update.

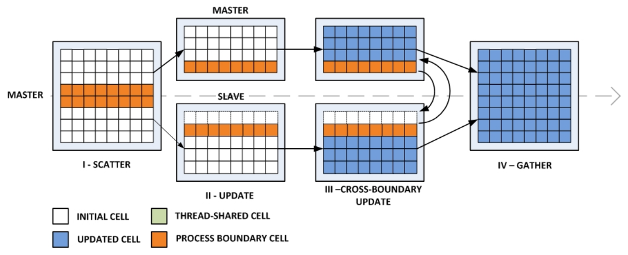


Figure 4: Parallelisation Strategies - MPI.

In our particular case, scattering of the regions is not enough because regions share logical space of the CA framework, so that a mechanism to send boundary cell states across this logical space needs to be present. Although MPI allows for asynchronous data communication that can be used on a per cell basis, i.e. a cell sends out updated state information as soon as it is changed, this places a prohibitive performance strain, both in terms of the amount of micro-communications that need to be made and in view of the fact that any boundary cell update would need to wait on neighbour boundary state updates in order to continue with the execution. To avoid this, we choose to serialise boundary layer updates to the end of the region update cycle. Similar to the concept of thread-level shared layers, we introduce *process-level shared layers*. These are constructed of boundary layers in each region. As illustrated in Figure 4, the update of the boundary layers is now the sole ownership of just one of the neighbouring process regions that share those layers. Upon completion of region updates, which can be done without the need for data from neighbouring processes, boundary layer information is sent to the "owner" process which completes the region update cycle, and returns the updated state back to the initial process. The main cycle then completes through the slave processes sending back the resulting state to the master. The resulting pseudo-code is presented in Algorithm 3: for brevity, the function *ApplyRules* replaces the behavioural "for loop" from previous code examples, which was responsible for rule applications.

Although this approach allows a much larger parallelisation ceiling, the fact that it involves cross-process communications, usually made across a physical network, means the bandwidth speed is the main bottleneck that impacts performance. Even when executing on the same node, using inter-process communication, the overhead can be particularly large, especially considering the amount of data present in the model.

3.5. Hybrid Parallelism

The advantages and disadvantages of the proposed parallelisation strategies naturally lead to the solution where both are combined in order to minimise the negative impacts of each [5]. On the one hand, we want to avoid the "hard ceiling" present in the SMP solution by leveraging cross-process communication, while on the other, we want to minimise the communication overhead present in the message-passing paradigm. In order to achieve this, we execute one process per node via MPI, but implement the threading within each region by using OpenMP. As a result, the process-level communication is reduced to an absolute minimum, while the limit to the amount of cores that can be used is drastically increased. Of course, communication overhead and non-parallelisable elements of the model will result in diminishing returns as the number of cores increase. Also the model geometry will be an ultimate limit, as the number of cores approaches the number of layers present. The resulting hybrid pseudo-code algorithm is shown as Algorithm 4 and its schematics in Figure 5.

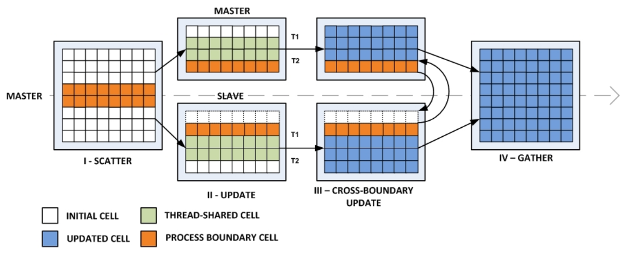


Figure 5: Parallelisation Strategies - Hybrid.

4. RESULTS

This section presents the performance results obtained by using the different parallelisation strategies. Model runs were performed using the Irish Centre for High End Computing (ICHEC) Stokes supercomputer. Stokes is an SGI Altix ICE 8200EX cluster with 320 compute nodes. Each compute node has two Intel (Westmere) Xeon E5650 processors and 24GB of RAM. The nodes are interconnected via two planes of ConntextC Infiniband. Our model was implemented in C++ (using gcc 4.3.0 64bit compiler), OpenMPI version 1.4.3 and GNU OpenMP (GOMP) version 3.1. In all cases, 1440 iterations of the simulation were run, with one iteration configured to represent one minute of real time. This choice was based on the need to correlate the data obtained to the *in vitro* experiments which usually run for 24 hours. The bounding volume of the simulation space was 300 x 300 x 300 cells, resulting in around 2.7 GB of model data.

Algorithm 4 *HybridUpdate*(Matrix M, int rank)

```
1: Secondary = M
2: n = Secondary.Regions.Count
3: Matrix region
4: if(rank == MASTER) then
5:   Scatter(M.Regions) ⇒ region
6: parallel for each slice in region.slices
7:     for each c in Shuffle(slice) do
8:       if(IsBoundary(c) == true) then skip
9:       if(IsShared(c) == true) then lock(c)
10:      ApplyRules(c, rule_list) ⇒ Secondary(region)
11:      if(IsShared(c == true) then unlock(c)
12: #end parallel
13: ExchangeBoundary(rank, rank + 1)
14: for each c in Shuffle(region.boundary) do
15:   ApplyRules(c, rule_list) ⇒
16:       Secondary(region.boundary)
17: if(rank == MASTER) then begin
18:   Gather(M.Regions) ⇐ Secondary(region)
19:   ComputeRelease(M)
20: end
```

4.1. Thread-level Parallelism

The focus of the first set of runs was on performance gain achievable using OpenMP parallelisation only. Simulations were run for 1-8 cores and as well as the execution times, we observed speedup of different code segments and their relative participation in overall program execution. Figure 6 (top) shows the improvement in execution times, which is significant and near-linear. At the low end of the tested spectrum, the initial execution time for non-parallelised code is almost 11 hours, while at the opposite end (8 cores) we gain a 6-fold wall-time improvement in execution speeds. From the other two measurements, we can see that the potential for speeding up parallel sections of the code can bring further time reductions, as parallel code still represents a larger portion of the total executed code. However, we are bound by the limitation of the SMP architecture, which prevents expanding the thread-parallel approach further.

4.2. Process-level Parallelism

Further, dividing the algorithm across compute nodes allows us to lift the restrictions imposed by the thread-level paradigm, using the MPI framework. We investigated the performance of pure-MPI, (i.e. no threading), solution over 2-32 cores, with the goals, both of comparing the performance to OpenMP as well as examining speedup potential and overhead for local and global communication on simulation speed and solution scalability. Results obtained in Figure 6 (middle) show that, for this model size, communication overhead has a significant impact on simulation times, although the potential of the MPI solution is ultimately greater as it does not suffer from ”ceiling” limitations.

Interestingly, due to expensive communication, the MPI solution is not as efficient as a comparable OpenMP one. Since the ”scatter and gather” operations incur a certain overhead (from 1-2 seconds per iteration in our experiments), the sequential portion of the simulation is increased. For the case of 8 cores, relative performance is almost 40% improved for thread-level parallelism. However, the real benefit occurs for a larger number of cores, where the MPI solution benefits from practically unlimited infrastructure.

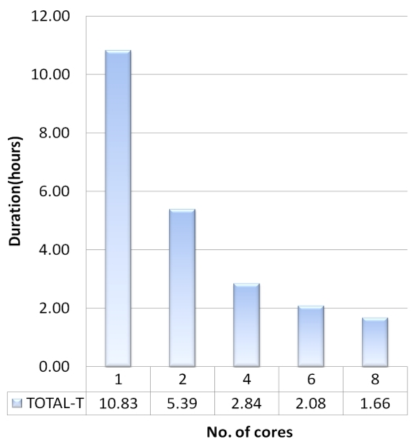
4.3. Hybrid Parallelism

Combining the above two approaches was the ultimate objective of this experiment. By using the fine-grained parallelism within a compute node, and thus avoiding the communication price, we expected to gain some advantage over the coarser-grained approach that MPI provides [9]. Figure 6 (bottom) presents the comparison of the two. From Figure 6 (middle), even with MPI for increasing number of cores, there is an increase in the relative sequential execution time. This is in agreement with findings presented in Figure 6 (bottom), where total improvement in execution speeds reaches a plateau after a certain number of cores. Above a certain point, at around 64 cores, the performance of either MPI or hybrid approach reaches the limit for the model.

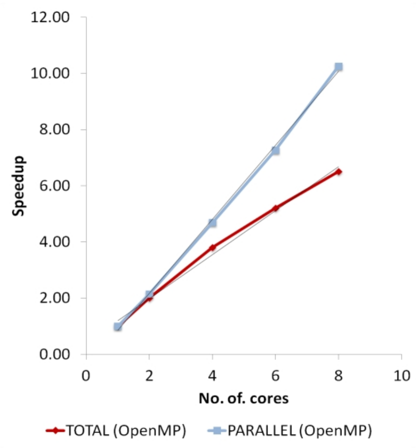
Finally, we investigated the average time taken to simulate each of the parallel slices and conclude that further optimisation is possible by applying an appropriate load balancing solution. For future work, some refinement of the approach is indicated enabling current benchmarks to be re-evaluated for different model-space division strategies.

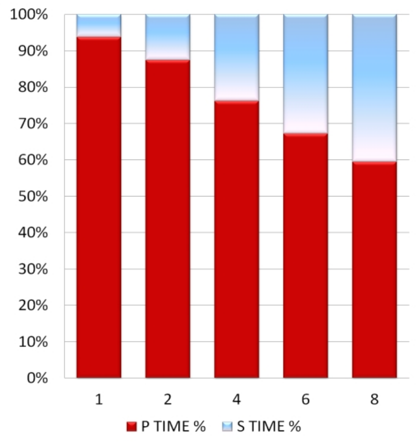
5. CONCLUSION

We have presented several parallelisation strategies, aimed at improving performance of the CA frameworks, used as a modelling basis for pharmaceutical investigations. Each strategy is viable, depending on speedup requirements and available infrastructure. In general, parallelisation platforms permit in-depth simulation of complex drug formulations, as well as evaluation of a wide parameter range through reduction of *in silico* experiment time.

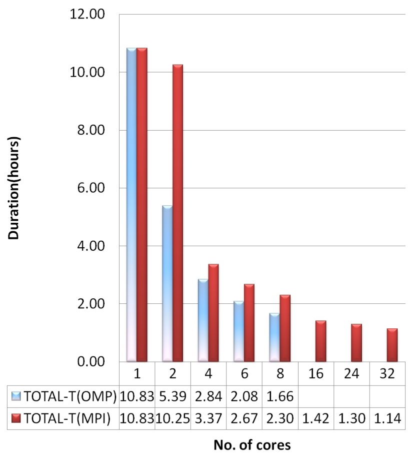


	1	2	4	6	8
▨ TOTAL-T	10.83	5.39	2.84	2.08	1.66

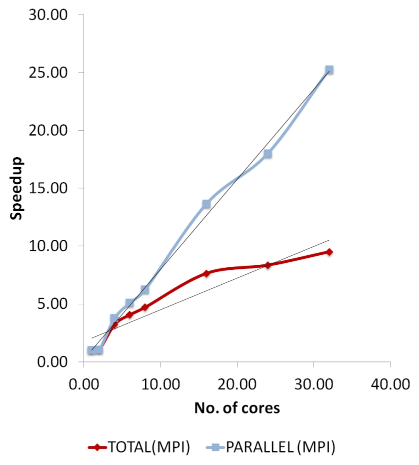


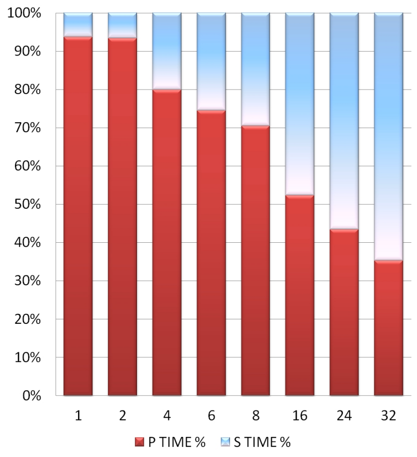


(Left) Execution Times (OpenMP); **(Centre)** Parallel vs. Total Speedup (OpenMP); **(Right)** Sequential vs. Parallel Execution (OpenMP)

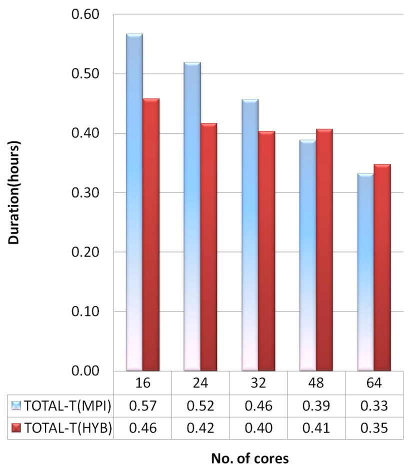


	1	2	4	6	8	16	24	32
▨ TOTAL-T(OMP)	10.83	5.39	2.84	2.08	1.66			
🟥 TOTAL-T(MPI)	10.83	10.25	3.37	2.67	2.30	1.42	1.30	1.14

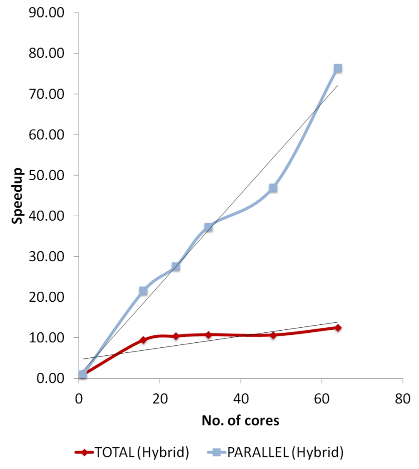


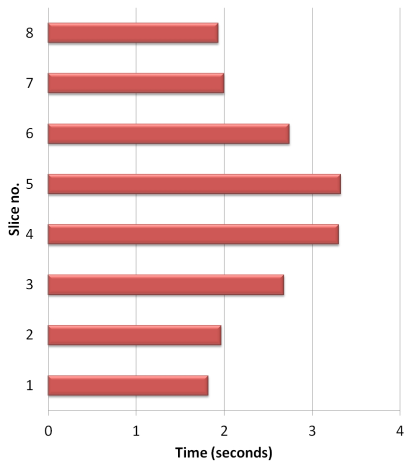


(Left) Execution Times (MPI vs. OpenMP); **(Centre)** Parallel vs. Total Speedup (MPI); **(Right)** Sequential vs. Parallel Execution (MPI)



	16	24	32	48	64
▨ TOTAL-T(MPI)	0.57	0.52	0.46	0.39	0.33
🟥 TOTAL-T(HYB)	0.46	0.42	0.40	0.41	0.35





(Left) Execution Times (Hybrid vs. MPI); **(Centre)** Parallel vs. Total Speedup (Hybrid); **(Right)** Per-Slice Simulation Times (1x8)

Figure 6: Performance Results: OpenMP **(top)**, MPI **(middle)** and Hybrid **(bottom)**

The results show that the hybrid approach offers the best performance, followed closely by the pure MPI based solution (i.e. no threading) over 8-32 cores. With the promised advances in MPI implementation, this gap may reduce further. Further improvements may be possible and can be made by experimenting with different load balancing methods, in order to close the gap observed in individual thread/process execution times. Finally, it would be interesting to see the applicability of the proposed solutions when run as part of commodity cloud computing infrastructures, which already promise a flexible and cost-effective solution platform [10] for the types of research questions of core importance to the pharmaceutical industry.

ACKNOWLEDGEMENTS

This work is funded by the Irish Research Council for Science, Engineering and Technology (IRCSET), through an 'Enterprise Partnership Scheme' postgraduate scholarship. The authors also wish to thank ICHEC for provided HPC resources and Mr. Gilles Civario for valuable advice on MPI during model development.

REFERENCES

[1] A. Barat, H. J. Ruskin, and M. Crane. "Probabilistic models for drug dissolution. Part 1. Review of Monte Carlo and stochastic cellular automata approaches", *Simulation Modelling Practice and Theory*, Vol.14, No.7, (2006), pp.843-856

[2] A. Barat, H. J. Ruskin, and M. Crane. "3D Multi-agent models for protein release from PLGA spherical particles with complex inner morphologies", *Theory in Biosciences*, Vol.127, (2008), pp.95-105

[3] M. Bezbradica, H. J. Ruskin, and M. Crane. "Modelling drug coatings: A parallel cellular automata model of ethylcellulose-coated microspheres", *Proceedings of ICBBB 2011*, Vol.5, (2011), pp.419-424

[4] A. Göpferich and R. Langer. "Modeling of polymer erosion", *Macromolecules*, Vol.26, (1993), pp.4105-4112

[5] H. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, and B. Chapman. "High performance computing using MPI and OpenMP on multi-core parallel systems", *Parallel Computing*, Vol.37, No.9, (2011), pp.562-575

[6] J. A. Kimber and S. G. Kazarian, and F. Štěpánek. "Microstructure-based mathematical modelling and spectroscopic imaging of tablet dissolution", *Computers & Chemical Engineering*, Vol.35, No.7, (2011), pp.1328-1339

[7] T. J. Laaksonen, H. M. Laaksonen, J. T. Hirvonen, and L. Murtomäki. "Cellular automata model for drug release from binary matrix and reservoir polymeric devices", *Biomaterials*, Vol.30, No.10, (2009), pp.1978-1987

[8] F. Massaioli, F. Castiglione, and M. Bernaschi. "OpenMP parallelization of agent-based models", *Parallel Computing*, Vol.31, No.10-12, (2005), pp.1066-1081

[9] B. J. Pope, B. G. Fitch, M. C. Pitman, J. J. Rice, and M. Reumann. "Performance of hybrid programming models for multiscale cardiac simulations: Preparing for petascale computation", *IEEE Transactions on Biomedical Engineering*, Vol.58, No.10, (2011), pp.2965-9

[10] M. C. Schatz, B. Langmead, and S. L. Salzberg. "Cloud computing and the DNA data race", *Nature Biotechnology*, Vol.28, No.7, (2010), pp.691-693

[11] J. Siepmann, N. Faisant, and J.-P. Benoit. "A new mathematical model quantifying drug release from bioerodible microparticles using Monte Carlo simulations", *Pharmaceutical Research*, Vol.19, No.12, (2002), pp.1885-1893

[12] K. Zygourakis. "Development and temporal evolution of erosion fronts in bioerodible controlled release devices", *Chemical Engineering Science*, Vol.45, No.8, (1990), pp.2359-2366

[13] K. Zygourakis and P. A. Markenscoff. "Computer-aided design of bioerodible devices with optimal release characteristics: A cellular automata approach", *Biomaterials*, Vol.17, No.2, (1996), pp.125-135