

# Parallelized Simulations of Grain-Surface Chemistry

William Burke, Drake Deuel, Esmail Fadae, Jamila Pegues

May 2019

## 1 Motivation

### 1.1 Background

The chemical composition and atmospheres of exoplanets are the target of new and upcoming telescope missions (e.g., NASA’s TESS Mission, Ricker et al., 2014, launched in April 2018). This is in large part because exoplanets offer the opportunity to find other worlds that are habitable to, and may even host, Earth-like life. However, in order to understand the chemical composition of these exoplanets, and to be able to distinguish any biosignatures from abiotic false positives (Ballard, 2019), we need to understand how these exoplanets formed their chemical compositions in the first place. This understanding requires studying chemistry on much, much smaller scales: on the surfaces of dust grains. Asteroids, planetesimals, and planets form from the collision and accumulation of microscopic dust grains; therefore grain-surface chemistry, through which complex molecules can form and even build up in layers on a grain’s surface, plays a crucial role in the chemical history of these planetary bodies.

Unfortunately, grain-surface chemistry is extremely difficult to study. Laboratory experiments would be the ideal method for exploring grain-surface chemistry; however, even just experimentally approximating the cold, dark, sparse conditions of space requires monetarily expensive laboratory setup and equipment. Computational simulations of grain-surface chemistry offer the potential to be extremely powerful tools for robustly exploring grain-surface chemistry in different astronomical environments. In practice, however, these simulations are also expensive, for several reasons: (1) chemistry happens quite quickly relative to grain growth, meaning that the simulation timesteps must remain short even as the timespan of the overall simulation grows; (2) atoms are small ( $\sim 10^{-10}\text{m}$ ) relative to the radii of typical dust grains ( $\sim 10^{-7}\text{m}$ ), such that the elements within the simulation must be small compared to the physical area represented by the simulation; and (3) in suitable physical conditions, atoms/molecules can accumulate on the grain surface over time, and the simulation must devote resources to keeping track of all of these particles at each timestep as the simulation progresses. These factors combine such that simulations of grain-surface chemistry quickly become computationally and temporally expensive as the size of the grain and the length of the simulation increase. To perform these simulations within reasonable time frames and resource budgets, we need an approach that minimizes the expenses without sacrificing simulation accuracy.

The work of Chang et al. (2005) shows that the major processes of grain-surface chemistry can be described in a Poisson framework. This means that grain-surface simulations can treat the movement of particles as independent events. Chang et al. (2005)

Parameter	Value	Reason for Value
$E_{bind,a}$	$5.14982 \times 10^{-21} \text{ J}$	Binding energy of H to the grain surface
$E_{surf,a}$	$3.957 \times 10^{-21} \text{ J}$	Diffusion barrier of H on the grain surface
$m_a$	$1.6738234 \times 10^{-27} \text{ kg}$	Mass of H
$n_a$	$10^6 \text{ m}^{-3}$	Typical H number density of diffuse ISM
$N_s$	$2 \times 10^{18} \text{ m}^{-2}$	Surface density of sites on the grain surface
$r_{gr}$	$0.5 \sqrt{\frac{S}{\pi N_s}}$	Radius of the grain; scales with grain size
$s_a$	1	Sticking efficiency
$S$	<i>Variable</i>	Total number of sites on the grain
$T_{gas}$	<i>Variable</i>	Temperature of the gas
$T_{grain}$	<i>Variable</i>	Temperature of the grain

Table 1: Simulation parameters and values. The grain surface is assumed to be homogeneous olivine; all grain-specific values ( $E_{bind,a}$ ,  $E_{surf,a}$ ,  $N_s$ , and  $r_{gr}$ ) are based on the values used for this surface type by Chang et al. (2005). The environment is assumed to be the diffuse interstellar medium (abbreviated as ISM).

used this framework to construct a continuous-time, random-walk simulation that explored the formation of molecular hydrogen ( $\text{H}_2$ ) from hydrogen atoms (H) on a grain surface under different physical conditions. Though their publication was released over a decade ago, the approach of Chang et al. (2005) is still used in studies today; however, the scales of these studies have remained small in order to avoid computational and temporal expenses at larger problem sizes (e.g., Chang and Herbst, 2014; Cuppen et al., 2017; Wakelam et al., 2017).

In this work, we present a new grain-surface simulation code that simulates the formation of  $\text{H}_2$  on grain surfaces. We take advantage of the Poisson framework of Chang et al. (2005) to optimize and parallelize the simulation. The code achieves a speed-up of  $\times 10$ , and its performance  $\times 10$  better than the serial case. The code can be expanded to handle more complex atoms, reactions, and grain-surface scenarios with little or no increase in computational and temporal cost.

## 2 Methodology

### 2.1 Grain-Surface Processes

Within typical grain-surface chemistry models, there are four major processes that a particle (i.e., an atom or molecule) can undertake (e.g., Cuppen et al., 2017). We include these processes within our simulation and briefly summarize them below. All processes are illustrated in Figure 1. We use the general equations as presented in the recent grain-surface chemistry review by Cuppen et al. (2017), but we focus on hydrogen and simulate a similar environment to that of Chang et al. (2005) to allow easy and consistent comparison between our results and their original work. Note that the purpose of this work was not to produce any new or accurate scientific results, but instead to develop an efficient baseline grain-surface simulation code, which can later be expanded to handle complex scientific grain-surface scenarios at drastically reduced computational and temporal costs.

All parameter values used for our simulation are listed in Table 1.

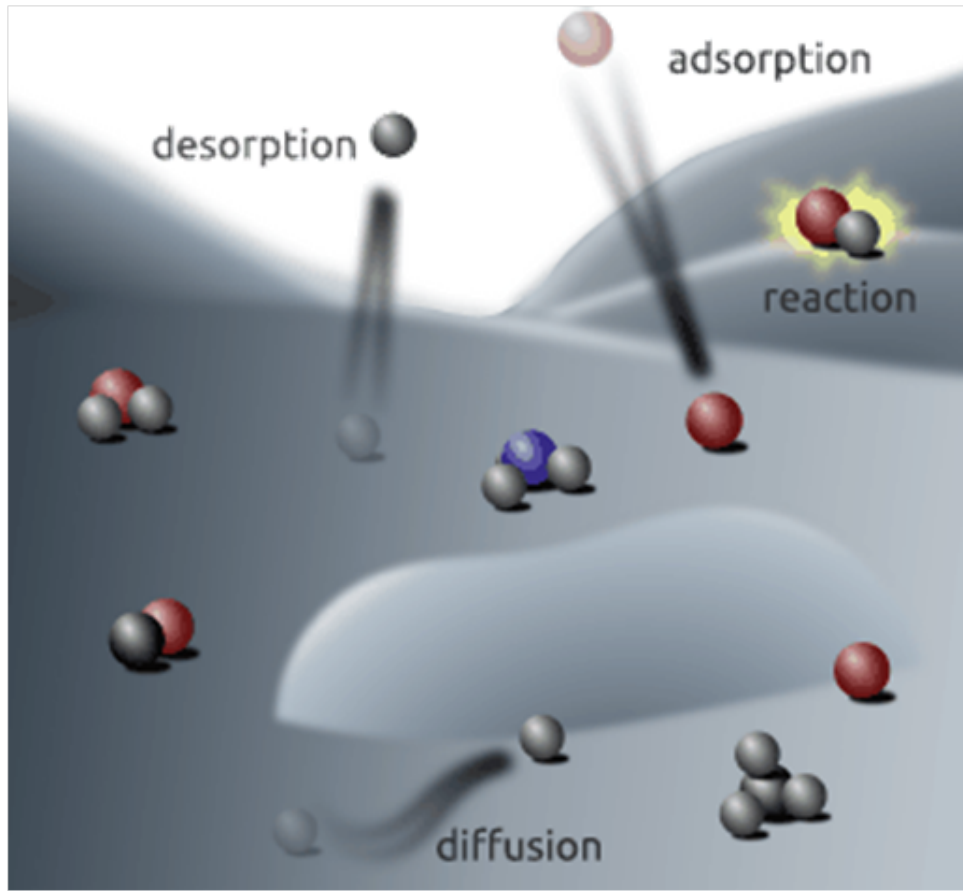


Figure 1: An illustration of the four major grain-surface processes included in our simulation. Reproduced from Figure 1 of Cuppen et al. (2017).

### 2.1.1 Adsorption

This is the process of a particle joining to a grain surface. The rate of this process (in units of  $[\text{time}^{-1}]$ ) is as follows:

$$R_{ads} = s_a v_a n_a \pi r_{gr}^2, \quad (1)$$

where  $s_a$  is a sticking efficiency  $\in [0, 1]$  that describes how likely the particle  $a$  will stick to the grain;  $n_a$  is the number density of  $a$  in the surrounding gas;  $r_{gr}$  is the radius of the grain; and  $v_a$  is the average thermal velocity of  $a$  in the gas:

$$v_a = \sqrt{\frac{8kT_{gas}}{\pi m_a}}, \quad (2)$$

where  $T_{gas}$  is the temperature of the gas,  $k$  is the Boltzmann constant, and  $m_a$  is the mass of the species  $a$ .  $T_{gas}$  and  $n_a$  are naturally environment-dependent (e.g., the interstellar medium will have very different temperatures and densities compared to a dense molecular cloud core).

### 2.1.2 Desorption (Thermal)

Thermal desorption is the process of a particle on a grain leaving the grain's surface. The rate of this process is given as:

$$R_{des} = \nu \exp\left(\frac{-E_{bind,a}}{kT_{gr}}\right), \quad (3)$$

where  $E_{bind,a}$  is the binding energy of a species  $a$  to the grain surface;  $k$  is again the Boltzmann constant;  $T_{gr}$  is the temperature of the grain; and  $\nu$  is the characteristic attempt frequency, given as:

$$\nu = \sqrt{\frac{2N_s E_{bind,a}}{\pi^2 m_a}}, \quad (4)$$

where  $N_s$  is the surface density of sites, or cells, on the grain surface; and  $m_a$  is the mass of  $a$ .

### 2.1.3 Diffusion

Diffusion is the process of a particle moving from one site on a grain to an adjacent site on a grain. Similarly to desorption, the rate of this process is given as:

$$R_{diff} = \nu \exp\left(\frac{-E_{surf,a}}{kT_{gr}}\right), \quad (5)$$

where  $E_{surf,a}$  is the diffusion barrier, which depends on the surface of the grain itself. Similarly, the characteristic attempt frequency is

$$\nu = \sqrt{\frac{2N_s E_{surf,a}}{\pi^2 m_a}}, \quad (6)$$

### 2.1.4 Reaction

Reactions occur when two particles are in the same cell on the surface. There is a particle-dependent probability that indicates how likely the product of a reaction (such as  $\text{H}_2$  from  $\text{H} + \text{H}$ ) will then desorb from the surface after the reaction. This is a parameter in our model. For the results below, we assumed that the probability of  $\text{H}_2$  desorption was 1.

## 2.2 Poisson Framework

Chang et al. (2005) describe grain-surface chemistry within a Poisson framework, which treats the adsorption, diffusion, and desorption of particles as Poisson events, which are completely independent of each other. Generically, the amount of time  $t$  that passes between two Poisson events follows an exponential curve, which Chang et al. (2005) calls a ‘waiting-time distribution’ (WTD):

$$\psi_a(t) = R \exp(-Rt), \quad (7)$$

where  $R$  is the rate of a process (e.g., desorption), and therefore  $R^{-1}$  is the average time that passes between the process described by  $R$ . The time  $t$  between two events of this process can then randomly be sampled as:

$$t = -\frac{\ln X}{R}, \quad (8)$$

where  $X$  is a random number sampled from a uniform distribution in  $(0, 1)$ .

This framework could be used to calculate the time between events on the grain within a grain-surface simulation. However, there is an issue with the simplifying assumption made by Chang et al. (2005). Plugging in the equation for  $t$ , we find

$$\psi_a(t) = R \exp(-Rt) = R \exp\left(R \frac{\ln X}{R}\right) = RX, \quad (9)$$

This amounts to sampling the uniform distribution when calculating the inter-event times, which does not preserve the properties of a Poisson process that 1) the number of events in disjoint intervals are independent and 2) the number of events in a fixed interval  $t$  follows a Poisson distribution  $\text{Pois}(Rt)$ .

Using Chang et al. (2005) as inspiration, we applied the Poisson framework in a way that more easily lends itself to parallelization and does not require any simplifying assumptions. For a given process with rate  $\lambda$ , the probability that the number events  $k=0$  in a timestep  $t$  is as follows from Blitzstein and Hwang (2015)

$$P(k=0) = \frac{e^{-\lambda t} (\lambda t)^k}{k!} = e^{-\lambda t}, \quad (10)$$

This gives the probability of at least one event

$$P(k > 0) = 1 - e^{-\lambda t}, \quad (11)$$

This applies directly to adsorption, but particles on a grain surface may undergo desorption and diffusion. The probability of at least one event happening is then a function of the combined rate

$$P(k > 0) = 1 - e^{-(\lambda_{\text{desorption}} + \lambda_{\text{diffusion}})t}, \quad (12)$$

Conditioning on an event happening, Blitzstein and Hwang (2015) tell us that the probability of is proportional to the rate

$$P(desorbtion|event) = \frac{\lambda_{desorbtion}}{\lambda_{desorbtion} + \lambda_{diffusion}}, \quad (13)$$

$$P(diffusion|event) = \frac{\lambda_{diffusion}}{\lambda_{desorbtion} + \lambda_{diffusion}}, \quad (14)$$

Thus, to accurately model the complex processes occuring on a grain surface, we need only choose a timestep such that the probability for of two events for a given poisson process is exceedingly low, or the probability of no events is very high. Since the probabilities are a nonlinear function of temperature and interaction energy, we enter the desired probability as a parameter to our model, in this case 99.99%. From this parameter, we calculate the timestep and then precompute the event probabilities, which are held constant for the rest of the execution. Each time an event occurs, we select what type of event it is by sampling the uniform distribution using the probabilities above.

The advantage of our approach is that we have preserved the properties of the Poisson process and made it much easier to parallelize our code. We can use fixed timesteps and treat the events in each timestep independently. There is a 99.99% probability that only 1 event will happen at each time step, so we can treat our tiny event probabilities as entries in a fixed probability transition matrix. Moreover, each particle in our system can be treated independently at each step, allowing us to divide the computational work among many cores.

## 3 Software Design

### 3.1 Serial Workflow

Algorithm 1 describes the general algorithm

```

Data: grain size, time, temperature, binding energies, gas concentrations
Result: simulation of chemistry reactions on grain surfaces
calculate transition probabilities;
create position matrix P;
for  $t = 0$  to  $T$  do
  for  $i = 1$  to  $n^2$  do
    if  $rand < tol$  then
       $P(i) \leftarrow P(i) + \text{desorbed molecule}$ 
    end
    if  $P(i)$  contains multiple molecules then
      if  $rand < tol$  then
         $P(i) \leftarrow P(i) + \text{products} - \text{reactants}$ 
      end
    end
  end
  for  $i = 1$  to  $n^2$  do
    if  $P(i)$  contains multiple molecules then
      for each molecule do
        if  $rand < tol$  then
           $s \leftarrow \text{new state}$ 
           $P(i) \leftarrow P(i) - \text{molecule}$ 
           $P(s) \leftarrow P(s) + \text{molecule}$ 
        end
      end
    end
  end
end

```

**Algorithm 1:** Serial algorithm for the grain-surface simulation.

### 3.2 Optimization and Parallelization Techniques

"Technical description of the parallel application, programming models ..."

"Technical description of the software design, code baseline, dependencies, how to use the code, and system and environment needed to reproduce your tests"

"Description of advanced features like models/platforms not explained in class, advanced functions of modules, techniques to mitigate overheads, challenging parallelization or implementation aspects..."

"Links to repository with source code, evaluation data sets and test cases"

All of our simulation code is available on our public git repository for the course: <https://github.com/ddeuel/CS205>

This was largely a compute focused project, as the only major data need was the storage of our position matrix. We used a hybrid model with distributed memory in MPI and shared memory in OpenACC. We divided our matrix among our eight nodes, and each node used GPU compute to state transformations using the transition probabilities and then update the position matrix. Finally, we used message passing between nodes to maintain the matrix globally and account for molecules diffusing across node divisions.

Our fast approach was possible because we carefully chose our timestep in such a way that we were able to guarantee strict locality. The probability of a particle moving between states more than once in a given timestep can be set to be arbitrarily close

to 0 given sufficient computational resources. We chose 0.01% for practical reasons, but this could be reduced even further if more resources were available. Adjusting this probability did not measurably affect the outcome of our simulation with H molecules, but it might become important for simulations involving more complex interactions than those considered here, in which case we would set the value of this variable using a convergence test.

Locality has a huge effect on the resources required for our simulation. If we had implemented a full transition matrix and used matrix multiplication, the time complexity of our algorithm at each step would have been  $O(n^6)$ , where  $n$  is the number of available sites along each edge of a square grain surface. The total number of sites is  $n^2$ , the transition matrix would have been  $n^4$ , roughly a terabyte for our grain size, and a fully coated grain with  $n^2$  particles would have required multiplications in  $O(n^6)$ . Instead, the locality guarantee allows us to iterate through an  $n^2$  position matrix, treating each particle as independent and considering only its current site, the 8 adjacent diffusion sites, and desorption as possible transitions. In our final implementation, the time complexity at each step is then  $O(n^2)$ .

Since the number of particles on the grain surface can vary from 0 to  $n^2$ , we felt that a data structure with memory requirements proportional to the number of particles would create problems with load balancing. Instead, we opted for a position matrix  $n^2$  elements, each of which stores the interaction energies of the particles at that site. Physically, the number of particles at each grain surface site is limited by space and the fact that particles in the same place will react with one another, so we were able to put an upper bound on the storage required for each matrix element. This fixed the size of our position matrix.

We took three approaches to parallelization on this code. The simplest approach was an OpenACC implementation of a single node multi-threaded CPU, shared memory parallel design. We decided to use OpenACC since this would allow for easy portability to a GPU parallel model with a simple flag change hopefully being the only thing required. Our file `grain_acc_cpu.c` uses an OpenACC pragma in our `hop()` function to parallelize the simulation of the diffusion and desorption processes on the grain. This process is represented in code as a nested for loop running over the occupancy matrix and executing probabilistic changes based on sampling to the grain matrix representation. We found that the CPU implementation was simplest to code because we could use the standard C `random()` functions to generate random numbers for sampling. We were able to achieve great scaling on this implementation with almost linear scaling as thread count increased, as seen below. We also achieved near linear speedup from the serial version of our code. There was overhead introduced through using OpenACC, as running with a single thread on OpenACC/PGI compilation and framework was significantly slower than running our serial version itself. This overhead based on `pgprof` analysis was largely due to fork, waits, and synchronization as we would expect. Using shared memory parallelization allowed us to not worry about separate copies of our matrix becoming out of sync.

The next parallel strategy was to then port our code over to be usable with GPU compute. This turned out to be less simple than simply removing the `-ta=multicore` CPU compilation tab for `pgcc` to compile. Instead we

Another approach was parallelization over shared memory using MPI. In this approach, we partitioned the simulated grain surface over multiple processors, so that the resulting submatrices could be operated upon simultaneously. This was a particularly challenging approach, for several reasons: (1) the particles had to be able to traverse the



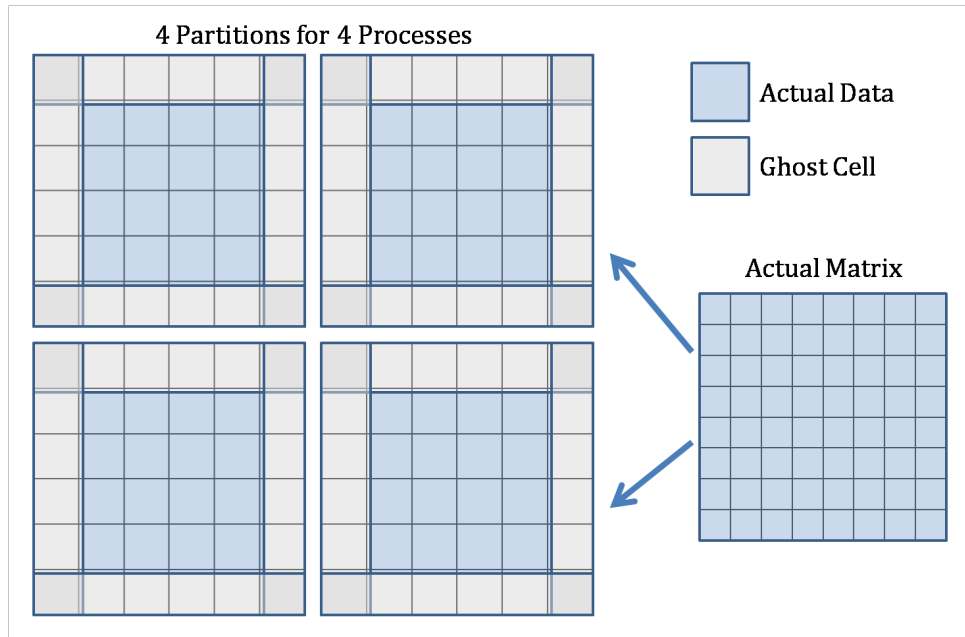


Figure 2: Illustration of our partitioning approach to shared-memory parallelization using MPI

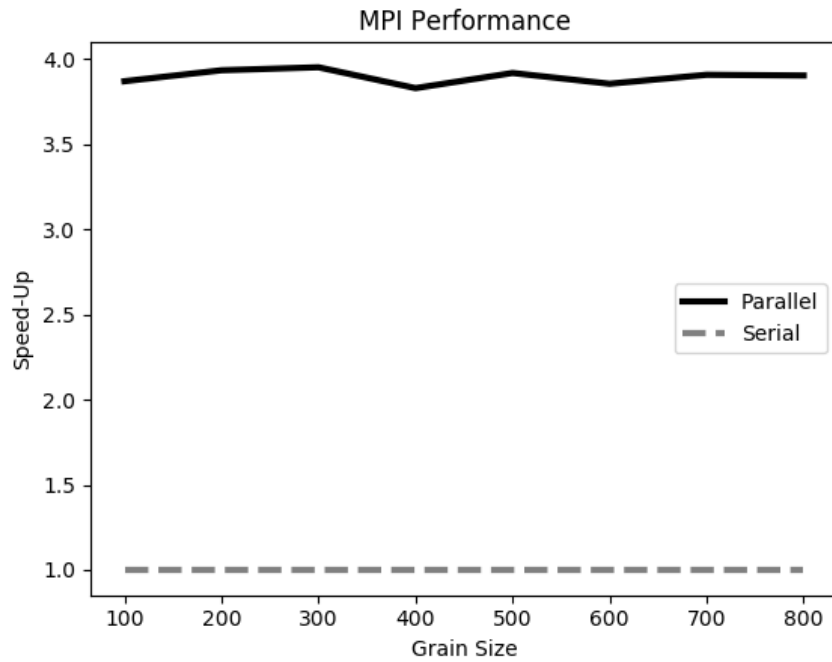


Figure 3: Parallel vs. serial speed-up for the simulation using four processors in MPI

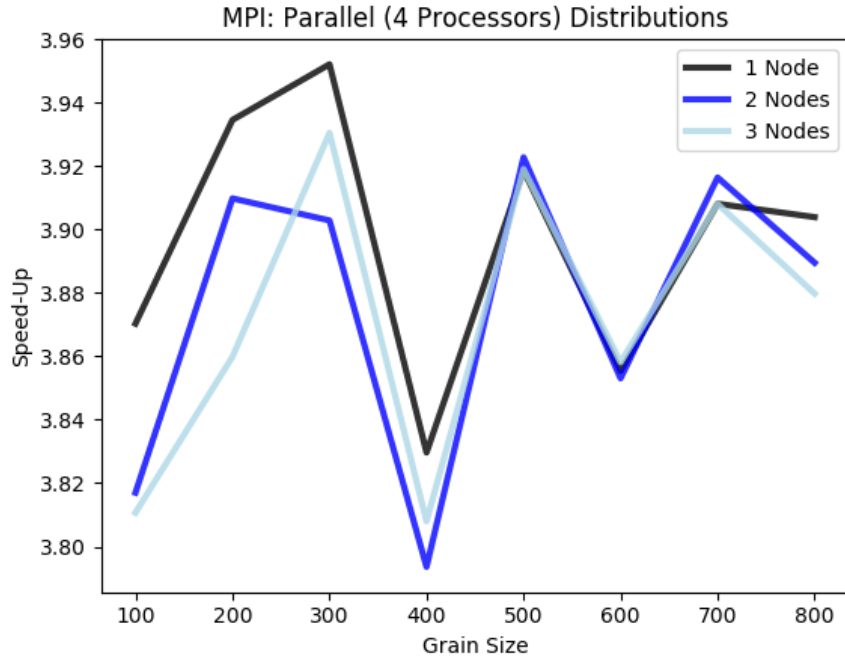


Figure 4: Parallel speed-up for the simulation using four processors in an MPI cluster

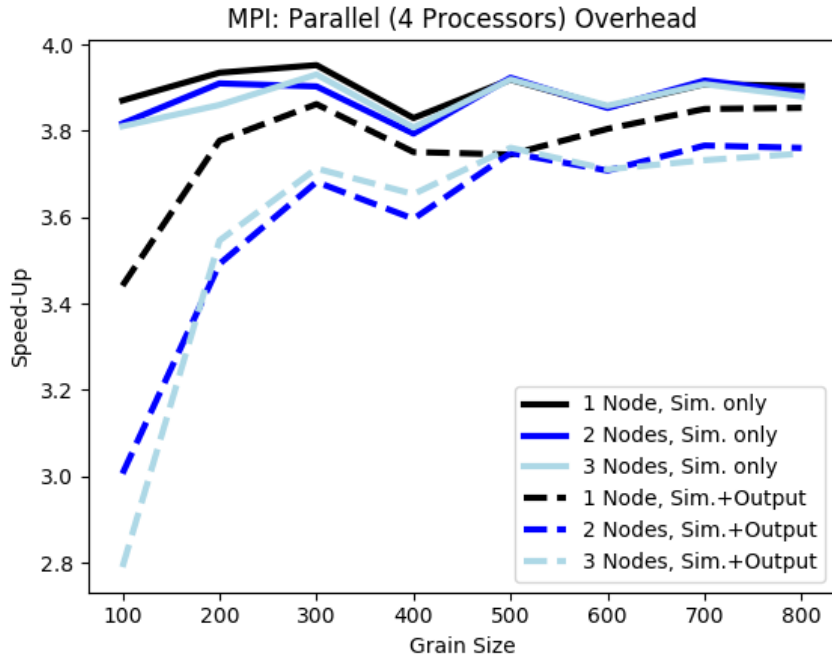


Figure 5: Overhead associated with synchronizing and outputting results across the different submatrices

entire grain, which meant that we had to facilitate communication (i.e., message-passing) across submatrices; (2) since the particles could roll over the grain edge and appear on the other side, we had to choose the moments of message-passing carefully, to avoid conflicts from overlapping diffusion; and (3) we had to employ both local and global techniques to update the submatrices at each timestep and then synchronize their results at the end of the simulation.

We used ghost cells as the medium of communication across the edges of submatrices. Figure 2 illustrates this approach. In summary, we partitioned the square grain surface among a given number of processors (restricted so that the grain surface could be split evenly in both length and width among the processors). We then padded the top, bottom, left, and right of each submatrix with ghost blocks, where no actual data was stored. At each timestep, each processor scanned the submatrix for particles and any diffusion or desorption events; whenever the processors reached the second-to-last or last cell within a row representing actual data, then the edges of each submatrix were updated by the ghost cells.

Figure 3 shows the speed-up of the simulation as a function of problem size for four processors on a single MPI node. Figure 4 shows the effect on the simulation speed-up given by distributing processors over a cluster of MPI nodes. Figure 5 demonstrates the overhead associated with synchronizing and outputting the results from each submatrix. We see that the speed-up relative to the serial case for the simulation is quite significant, and is roughly constant with problem size. Distributing the work over multiple nodes in a cluster does not drastically alter performance for the simulation alone. However, the overhead associated with post-simulation synchronization and output for multiple nodes is not negligible. Notably this overhead is most detrimental to performance for small problem sizes, and decreases in significance as problem size increases.

### 3.3 System Setup

All code was written in C. We used a g3.4xlarge instance for serial, CPU, and GPU timings to keep the timings as consistent as possible for cross referencing speedup. We followed the infrastructure guide i5 on OpenACC on AWS provided by the course for instance configuration and compiler set up except that we used the latest 19.4 version of PGI.

To configure the shell environment to use OpenACC and compile with pgcc run these commands must be run on each new instance shell session:

```
export PGI=/opt/pgi;
```

```
export PATH=/opt/pgi/linux86-64/19.4/bin:$PATH;
```

The compilation lines used for generating executables was:

```
serial: gcc -DUSE_CLOCK grain.c timing.c -o grain -lm
```

```
multi-threaded CPU: ACC_NUM_CORES=2 pgcc -Minfo -ta=multicore -DUSE_CLOCK grain_acc.c
```

```
GPU: pgcc -acc -Minfo -DUSE_CLOCK grain_acc.c timing.c -o grain_acc -lm
```

To execute we have a format of:

```
./executable n NUM_STEPS TIMING_ONLY_FLAG
```

Where `n` is an integer value for the grain size as `n x n`, `NUM_STEPS` is the integer number of time steps, and `TIMING_ONLY_FLAG` is a integer of 0 or 1 for 0 as verbose output, and 1 as timing value output only.

An example run input would then be:

```
./grain 300 100000 0
```

For MPI, we used the t2.2xlarge AWS instance(s). We compiled and ran the code as follows:

```
serial compiler: mpicc -DUSE_CLOCK grain_serial.c timing.c -o grain_serial -lm
```

```
parallel compiler: mpicc -DUSE_CLOCK grainMPI.c timing.c -o grain_serial -lm
```

```
serial: mpirun -np 1 ./grain_serial size numpsteps 0
```

```
parallel (4 processors, 1 node): mpirun -np 4 ./grainMPI size numsteps 0
```

```
parallel (4 processors, MPI cluster with 2 nodes): mpirun -np 4 -hosts main,node1 ./g
```

```
parallel (4 processors, MPI cluster with 3 nodes): mpirun -np 4 -hosts main,node1,nod
```

Where *grain\_serial.c* and *grainMPI.c* are the serial and parallel files, *size* is the grain size, *numsteps* is the number of steps, 0 is a flag that prints out a summary of the performance, and *main*, *node1*, and *node2* were the arbitrary names given to the nodes in the cluster.

When running our serial implementation, timings showed exponential growth with the size of our grain. This is to be expected as we are operating on a square matrix giving at least  $O(n^2)$

Timing our code when we executed in serial, we found that for the serial portions of our code took ???????? seconds. The parallelizable portions took ???????? seconds. The fraction,  $p$ , of parallelizeable runtime to total runtime is then ????????. According to Amdahl's law, this means that our theoretical maximum speedup  $S_{max}$  for a fixed problem size is

$$S_{max} = \frac{1}{1 - p} = ????????$$

In practice, we found the following strong scaling speedups for a fixed problem size:

Previously, we hypothesized that each time step would consume roughly the same amount of resources so that the overall execution time would be proportional to the runtime of the individual timesteps,  $O(n^2)$ . Execution times are plotted in 1 as a function of the side length of the grain,  $n$ . In practice, we observed the following weak scaling speedups as we increased our problem size:

## 4 Discussion

"Final discussion about goals achieved, improvements suggested, lessons learnt, future work, interesting insights" Citations

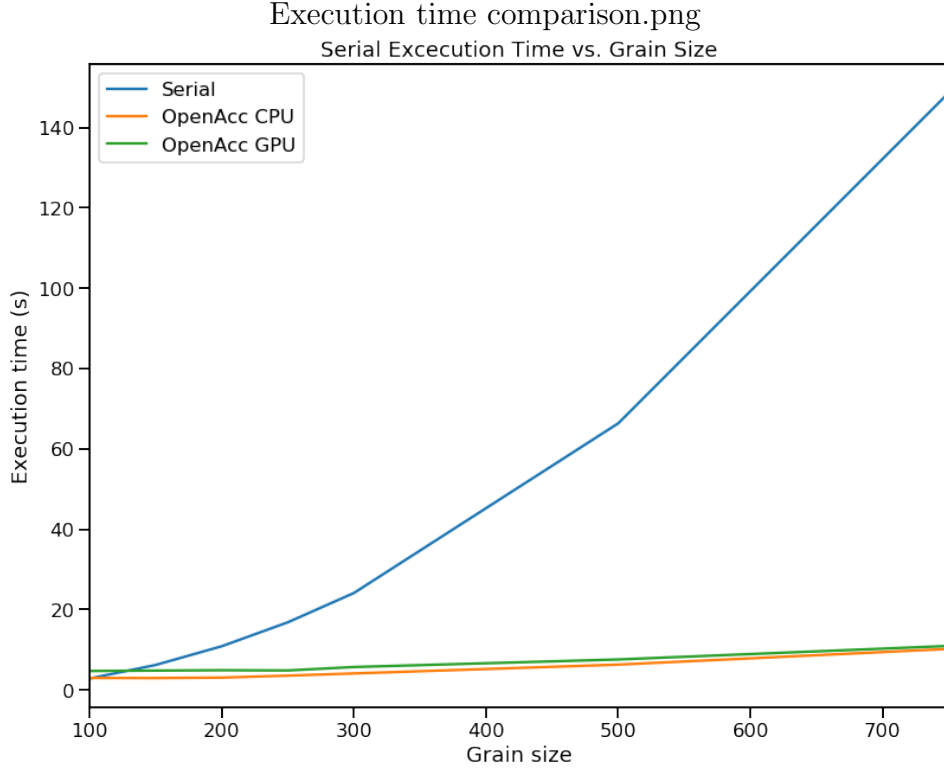


Figure 6: Serial execution times of our surface simulations for square grains with  $n$  sites on each edge, or  $n^2$  sites in total

## References

- Sarah Ballard. Predicted Number, Multiplicity, and Orbital Dynamics of TESS M-dwarf Exoplanets. , 157(3):113, Mar 2019. doi: 10.3847/1538-3881/aaf477.
- J. K. Blitzstein and J. Hwang. *Introduction to Probability*. CRC Press, Taylor Francis Group, 2015.
- Q. Chang, H. M. Cuppen, and E. Herbst. Continuous-time random-walk simulation of  $H_2$  formation on interstellar grains. , 434(2):599–611, May 2005. doi: 10.1051/0004-6361:20041842.
- Qiang Chang and Eric Herbst. Interstellar Simulations Using a Unified Microscopic-Macroscopic Monte Carlo Model with a Full Gas-Grain Network Including Bulk Diffusion in Ice Mantles. , 787(2):135, Jun 2014. doi: 10.1088/0004-637X/787/2/135.
- H. M. Cuppen, C. Walsh, T. Lamberts, D. Semenov, R. T. Garrod, E. M. Penteado, and S. Ioppolo. Grain Surface Models and Data for Astrochemistry. , 212(1-2):1–58, Oct 2017. doi: 10.1007/s11214-016-0319-3.
- George R. Ricker, Roland Kraft Vanderspek, David W. Latham, and Joshua N. Winn. The Transiting Exoplanet Survey Satellite Mission. In *American Astronomical Society Meeting Abstracts #224*, volume 224 of *American Astronomical Society Meeting Abstracts*, page 113.02, Jun 2014.

Serial	OpenAcc CPU	OpenAcc GPU	Grain size
2.832046	2.969	4.702021	100
6.208168	2.947	4.808163	150
10.86386	3.055	4.904208	200
16.822279	3.554	4.84202	250
24.115229	4.1	5.694505	300
66.341999	6.294	7.574742	500
148.588536	10.197	10.941632	750

Figure 7: Serial execution times of our surface simulations for square grains with  $n$  sites on each edge, or  $n^2$  sites in total

Valentine Wakelam, Emeric Bron, Stephanie Cazaux, Francois Dulieu, Cécile Gry, Pierre Guillard, Emilie Habart, Liv Hornekær, Sabine Morisset, Gunnar Nyman, Valerio Pirronello, Stephen D. Price, Valeska Valdivia, Gianfranco Vidali, and Naoki Watanabe.  $\text{H}_2$  formation on interstellar dust grains: The viewpoints of theory, experiments, models and observations. *Molecular Astrophysics*, 9:1–36, Dec 2017. doi: 10.1016/j.molap.2017.11.001.