condition calculation every iteration, or should one define an interval (say after e.g. every 5 or 10 iterations) that one checks for convergence? Explain why, and propose a clever algorithm yourself.

Hint : this depends of course heavily on the grain size of your parallel computation and the relative cost of the stopping condition as compared to the cost of a parallel iteration. So, take that into account in your discussion. Clearly, for this you need the results from assignment (A) and (D) of the previous section.

(B) Using your parallel stopping condition, implement the parallel Jacobi iteration (re-using the software for the parallel time dependent simulation) and test it. Especially test the influence of the number δ on the number of iterations and the quality of the solution (the linear dependence of the concentration on the *y*-coordinate, see Eq. 7).

3.3.2 The Gauss-Seidel Iteration

An improvement over the Jacobi iteration is the Gauss-Seidel iteration, where during the iteration a new value is used as soon as it has been calculated. Assuming that the iteration proceeds along the rows (i.e. incrementing *i* for fixed *j*), the Gauss-Seidel iteration reads

$$c_{i,j}^{(k+1)} = \frac{1}{4}\left[c_{i+1,j}^{(k)} + c_{i-1,j}^{(k+1)} + c_{i,j+1}^{(k)} + c_{i,j-1}^{(k+1)} \right]. \qquad (17)$$

Although the Gauss-Seidel iteration is not a big improvement over the Jacobi iteration (in terms of the amount of iterations needed for convergence) and only is a first step in introducing the Successive Over Relaxation method (SOR, see next section), the Gauss-Seidel iteration *does* change an enormous amount in the parallel algorithm. The inherent locality in the Jacobi iteration is presumably lost in Eq. 15. A re-ordering of the calculations is needed to regain the necessary data locality.

(A) Describe this new re-ordering, the Red/Black order that allows a parallel implementation of the Gauss-Seidel iteration. What is the consequence of this ordering in terms of the communication overhead per iteration? Adapt your time complexity model for this new situation. What happens?

(B) Implement the parallel Gauss-Seidel iteration using Red/Black ordering, test the correctness, scalability, and convergence behavior.

3.3.3 The Successive Over Relaxation

Now that you have the parallel Gauss-Seidel iteration in place, it is easy to take the next and final step. The Gauss-Seidel iteration did not provide a huge improvement over the Jacobi iteration. A next improvement comes from the Successive Over Relaxation (SOR). SOR is obtained from Gauss-Seidel by an over-correction of the new iterate, in formula

$$c_{i,j}^{(k+1)} = \frac{\omega}{4}\left[c_{i+1,j}^{(k)} + c_{i-1,j}^{(k+1)} + c_{i,j+1}^{(k)} + c_{i,j-1}^{(k+1)} \right] + (1-\omega) c_{i,j}^{(k)}. \qquad (18)$$

This method converges only for $0 < \omega < 2$, and for $\omega < 1$ the method is called under-relaxation. For $\omega = 1$ we recover the Gauss-Seidel iteration. It turns out that for our diffusion problem the optimal ω (that minimizes the number of iterations) lies somewhere in the interval $1.7 < \omega < 2$. The exact value depends on the grid size *N*.

(A) Implement the parallel SOR. Test the quality of the result. How does the number of iterations depend on ω and *N* ? What is the optimal ω. Does the number of iterations also depend on *p* ? What about the performance of the code ?

So far we have only looked at diffusion in a rather dull domain. Now that we have an efficient parallel iterative solver available, it's time to start including some object into the domain. So, now we assume that within our computational domain we include say a square object. We assume that the object is a *sink* for the diffusion concentration, that is, the concentration is zero everywhere on the object.

(B) Implement the possibility to include objects into the computational domain. The objects should be sinks. Implement this in such a way that again you are independent of the number of processors. Furthermore, in general objects can be located in the computational domain in such a way that processor boundaries are crossed.

Clearly the presence of objects will influence the load balancing. For now you may ignore this problem. Just use exactly the same decomposition as you have used before. We will come back to this issue in the final assignment on data decomposition.

Hint: For the iterations, the presence of the objects is not complicated. If a point (i,j) is part of an object, the concentration is just 0, and an iteration is not necessary (i.e., the new value is also 0). Therefore, you must implement some easy encoding of the object in the computational grid, and during the iterations simply test if the grid point that you are updating is part of the object or not. If not, you apply the SOR rule, if yes, just put the new value to zero. The easiest encoding is just an extra array of integers, where e.g. a one-value would code for the presence of an object, and a zero value for the absence of an object.

(C) Experiment a little bit with some objects in the computational domain (e.g. a square, or a few squares, …). What is the influence on the number of iterations. What about the optimal ω, is it influenced by the presence of objects? Look at the resulting concentration fields, and try to interpret what happens.

3.4 Diffusion Limited Aggregation

The diffusion-limited aggregation (DLA) is a growth model based on diffusing particles. The growth is started with a single seed (just a small square object in a single lattice point, at the bottom of the computational domain). As described in detail in the lecture notes, DLA can be modeled by solving the time independent diffusion equation (i.e. the Laplace equation), locating growth candidates around the cluster, and assigning a growth probability for each growth candidate, as a function of the concentration of diffusing nutrients at that growth location. Next, the growth candidates are added to the cluster with probability p_g. After this growth step, the diffusion equation is again solved, and the process is iterated for a large number of growth steps.

You currently have almost all the tools available for a parallel simulation of the DLA growth model. Let us take a closer look at the growth model itself, allowing you to insert this into your parallel programs. Figure 12 shows a possible configuration on an object (the filled dots) and the growth candidates (the open circles). A growth candidate is basically a lattice site that is not part of the object, but whose north -, or east -, or south -, or west neighbor is part of the object.

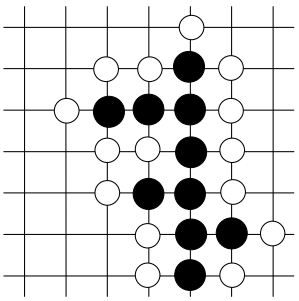

Figure 12: the object and possible growth sites

The probability for growth at each of the growth candidates is calculated by

$$p_g\left((i,j)\in\circ \rightarrow (i,j)\in\bullet\right)=\frac{\left(c_{i,j}\right)^{\eta}}{\sum_{(i,j)\in\circ}\left(c_{i,j}\right)^{\eta}}. \qquad (19)$$

The parameter η determines the shape of the object. For $\eta = 1$ we get the normal DLA cluster. For $\eta < 1$ the object becomes more compact (with $\eta = 0$ resulting in the Eden cluster), and for $\eta > 1$ the cluster becomes more open (and finally resembles say a lightning flash).

Modeling the growth is now a simple procedure. For each growth candidate a random number between zero and one is drawn and if the random number is smaller than the growth probability, this specific site is successful and is added to object. In this way, on average just one single site is added to the object.

The assignment of growth candidates is completely local, and can be done in parallel (we assume that the object is located in the domain, such that it crossed processor boundaries). However, to calculate the probabilities, as in Eq. 17, we need to find the normalization constant (the summation in the denominator). This implies a global communication routine.

(A) Implement the growth model in parallel, paying special attention to the calculation of the growth probabilities.

Hint: You need random numbers, you can use the `drand48()` function to generate them. You should provide each processor with a different seed, in order to make sure that each processor generates a different set of random numbers (i.e. to avoid any correlation between processors).

Now, with the growth model in place you are ready to perform growth simulations.

(B) Run a number of growth simulations. Take large domains (say 512^2) and investigate the influence of the η parameter. Can you still optimize by setting your ω parameter in the SOR iteration to a specific value? How is the speedup of your final parallel simulation of the DLA growth? Is this what you would have expected, based on all the previous assignments?

Hint : the SOR iteration is run over and over again on a slowly growing object. As the growth step is constructed in such a way that on average only one lattice site is grown to the object, the concentration fields will hardly change. Therefore, it is advantageous to start a new SOR iteration with the solution of the previous growth step.