In this thesis we have presented some Lagrangian particle models for absolute and relative dispersion of passive tracers and reactive species. In Lagrangian particle models many particle trajectories are computed to make accurate predictions of the mean concentration and the concentration fluctuations. The more particles, the more accurate the predictions. Calculating the trajectories of many particles requires much computing time; therefore, the need for parallelisation is obvious. One way of introducing parallelism is to use several processors each including a program control unit, an arithmetic logic unit and several memory and I/O modules: multiprocessors. Two distinct multiprocessor systems exist; shared memory and distributed memory (Moldovan, 1993; Petkov and van der Steen, 1993). The processors in a shared-memory machine communicate via a common memory, which forms a bottle-neck if all processors want to access the memory concurrently. A distributed memory multiprocessor system consists of several independent modules (processors with an own memory unit) and an interconnection network. Properties such as simplicity and scalability make distributed memory multiprocessors prime candidates for very large systems. For our application we purchase massive parallelism, therefore, we focus on the implementation of the particle-models on a distributed memory system. The implementation of an algorithm on a parallel computer is called parallelisation. More high performance computing terminology is given in (Petkov and van der Steen, 1993).

In Section 8.1 we briefly point out the problems of implementing Eulerian models on parallel computers. Next the implementation of one-particle and two-particle models for the turbulent transport of passive tracers is discussed. We shortly point out the parallelisation possibilities of Chock and Winkler’s particle-grid model and the pair-description of Komori et al.. We present an efficient method to parallelise the new two-particle model for turbulent transport of reactive species.
8.1 Parallel Eulerian models

In Chapter 2 we gave an overview of models for turbulent transport and chemical reactions from an Eulerian point of view. In incompressible flows the problem can be described by a set of differential equations; the advection-diffusion-reaction equations, the incompressibility equations and the Navier-Stokes equations. In Direct Numerical Simulations (DNS) these equations describe the instantaneous concentrations and velocities. In Reynolds Averaged Navier-Stokes (RANS) models and Large Eddy Simulations (LES) the equations are averaged in time or space and describe averaged concentrations and velocities. Differential equations are solved numerically by discretising the equations in space and time.

Time discretisation of the equations leads to simple schemes to compute the velocity of the turbulent flow and the concentration field at time $t_n$ from the values of the velocity and concentration at the previous time step $t_{n-1}$. These schemes are called explicit.

If one discretises the problem such that it leads to a set of implicit equations, a larger time-step size can be used than in the explicit methods, therefore implicit methods are often preferred. However, the implicit approach requires the solution of large sets of non-linear equations, which in the case of many chemical reactions is computationally very expensive. Methods that can be used to solve these sets of non-linear equations are for example the Newton approaches, multi-grid or the Gauss-Seidel iterative methods (Verwer and Blom, 1995).

In air pollution research one commonly applies operator splitting where the transport and chemistry processes are treated separately (McRae et al., 1982) such that the computational requirements are smaller. We will briefly point out the problems which occur when parallelising numerical algorithms for solving the differential equations by considering the advection-diffusion part of the problem. The chemistry part may be solved using Ordinary Differential Equation (ODE) solvers (Verwer and Simpson, 1995; Sandu et al., 1996) and can be parallelised in a similar way. In (Spee et al., 1996) the parallelisation of the operator splitting methods are discussed in more detail.

An implicit discretisation of the advection-diffusion equation is an iterative solution algorithm where in each time-step a large system of linear equations has to be solved. For each grid-point there is one equation, hence for a grid of 1000 points in each direction, one obtains a system of $10^6$ equations whereas each equation contains in the order of 10 terms. We may formulate this problem mathematically as follows. Solve $x_n$ in

$$ Ax_n = b(x_{n-1}, t_{n-1}) $$

where the matrix $A$ and the vector function $b(x_{n-1}, t_{n-1})$ follow from the set of discretised differential equations. The vector $x_n$ consists of elements equal to the concentration field and the velocity at the spatial grid-points at time $t_n$. The matrix $A$ typically consists of much less than 1% non-zeros. Solving these large sparse systems directly is nearly impossible due to the enormous size of
the systems and is inefficient because these direct methods do not profit from the sparse structure of the matrix. Therefore a lot of research is being conducted on iteratively solving these systems. For symmetric matrices $A$ the conjugate gradient method (CG) can be used. Non-symmetric but real valued systems can be solved with the general minimal residual method (GMRES). Many other methods and combinations of methods are being developed. A good overview of iterative methods for solving linear systems is given in (Barrett et al., 1994; Sleijpen and van der Vorst, 1993; Petkov and van der Steen, 1993). Due to the large size of the matrices, an enormous amount of computer storage and time are needed. A lot of effort is put into the implementation of these iterative solvers on parallel computers. Parallelisation of the iterative methods for solving linear systems is based on domain-decomposition, where the computational domain is divided in sub-domains such that each processing node performs the computations for a small part of the domain (Keyes et al., 1995). In the mathematical matrix formulation this means that the matrix $A$ and the vector $b$ are split up and distributed over the nodes such that each node solves its part of the problem. The processors can not work independently because the turbulent transport in one part of the domain depends on the evolution of the transport in neighbouring domain parts. Accordingly each node needs to know what other processors are doing and thus communication between processing nodes is needed. The problem of balancing the computation and communication load per processor is difficult and the optimal solution differs per physical problem (Keyes et al., 1995). We refer to (Barrett et al., 1994; Crone and van der Vorst, 1993; Crone, 1994; de Sturler, 1994; Crone, 1995; Hoffmann et al., 1993) for more information on the parallelisation of the CG method, of GMRES and of iterative solvers on parallel computers in general. For more specific information about the parallelisation of LES, we refer to (Leith, 1993; Dannevik, 1993).

Solving the Eulerian models for turbulent transport and chemical reactions on multi-processor computers is not simple. This is valid for the DNS as well as for the LES and RANS models. The biggest problem in parallelising these models is that the optimal domain-decomposition for one specific problem may be a very bad solution for the next. The problems are even bigger if one considers complex domains or unstructured and irregular grids for the discretisation of the differential equations.

8.2 Parallel Lagrangian models

Lagrangian models are very different from Eulerian models. Instead of solving a set of differential equations, one generates the trajectories of many particles or particle pairs. The accuracy of the predictions depends mainly on the number of particles. Implementing Lagrangian models on multi-processor systems is therefore completely different from parallelising Eulerian models. In Section 8.3 we present the parallelisation of particle models for the transport of passive tracers. Next, the implementation of the models for turbulent transport and chemistry on
parallel systems is discussed. We have seen that the generation of random numbers represents a large part in the calculation of particle trajectories. Therefore, in this section we give an overview of random number generators on computers in general and on parallel machines in particular.

8.2.1 Random number generation on parallel computers

A computer calculates random numbers from a deterministic algorithm; therefore such a random number is often called a pseudo-random number.

A simple classical and good algorithm to generate pseudo-random numbers from a uniform distribution is the Linear Congruential Method (LCGM) which was first introduced by Lehmer in 1948. The model generates a sequence of numbers \( x_n \) by the recursive rule

\[
x_{n+1} = a \cdot x_n + c \mod M
\]

where \( M \) represents a large positive number, \( a \) denotes an integer chosen such that \( \text{gcd}(M, a) = 1 \) and \( c \) is an integer from the set \( \{0, 1, 2, \ldots, M - 1\} \). Often \( M \) is chosen close to the largest integer that can be represented by the computer, for example \( M = 2^{32} \) or the Mersenne prime number: \( M = 2^{31} - 1 \). The sequence of random numbers \( x_n \) is periodic with period smaller than \( M \). Most frequently used choices of the parameters \( a, c \) and \( M \) and the performance of the resulting LCGM based generators are discussed in (Knuth, 1981; James, 1990). The LCGM generator is initialised by choosing an initial value \( x_0 \) also called the seed of the random number generator.

Random number generators used in stochastic processes, for example in Lagrangian models, should have a very large period such that many uncorrelated numbers can be generated. An extra requirement is posed on generators which are implemented on parallel computers: each series of random numbers generated on one computing node must be independent and uncorrelated from the series generated on the other nodes.

The LCGM computes only one series. However, a LCGM can be used on a parallel computer. If \( P \) processors participate in the calculation, the random numbers used on processor \( p \) can be derived from the sequential LCGM sequence by using the sequence \( \{x_p, x_{p+1}, x_{p+2}, \ldots\} \). Such a sequence can be generated on a parallel computer in two ways. The first method consists of dedicating one processor to generating random numbers and making sure that all other processors which perform the calculations, get their random numbers in time. In the other method, each processor computes his own sequence \( \{y_n\} \) from the sequential sequence
\{x_n\} using the following recursive rule

\[y_0 = x_p = (a^p x_0 + c \sum_{j=0}^{p-1} a^j) \mod M\]

\[y_n = x_{p+n^P} = (a^P y_{n-1} + c \sum_{j=0}^{P-1} a^j) \mod M\]

where \(P\) denotes the total number of processors participating in the calculation and \(p\) is the local number of the processor (Fox et al., 1988; Petkov and van der Steen, 1993). In the choice of parameters \(a, c\) and \(M\) of the LCGM, one has to take care that the \(P\)-tuples of random-numbers are independent. Not all generators have this property, see (Knuth, 1981, page 90) for an example of a random generator which gives good random numbers but if considering the random numbers in pairs, one gets a well-ordered grid-pattern. The initial seed \(x_0\) determines where to start the sequence, but a LCGM generates always the same numbers in the same order. On massively parallel computers where very large problems may be solved, the period of the LCGM may be too small.

Marsaglia et al. (Marsaglia et al., 1990) were the first to develop an algorithm, RANMAR which is portable, has a very long period and most of all, it generates many independent disjoint sequences of random numbers. The algorithm uses a Fibonacci method in which the series of random numbers is derived from

\[x_n = (x_{n-p} \circ x_{n-q}) \mod M\]

where \(\circ\) denotes a binary or logical operation, and \(p\) and \(q\) are integers with \(p > q\). That is, a new random number is computed by adding, subtracting, multiplying or performing another operation on two earlier generated random numbers, \(x_{n-p}\) and \(x_{n-q}\). Obviously, for the initialisation of these Fibonacci generators the first \(p\) numbers of the sequence have to be generated in order to start the recursive algorithm. For more details about this algorithm we refer to (James, 1990; Marsaglia et al., 1990). In RANMAR the Fibonacci generator is used with \(p = 97\) and \(q = 33\) and operation \(\circ = \text{“subtraction plus one, modulo one”}\). In (Marsaglia et al., 1990) more details about the algorithm RANMAR are given. In our simulations we used a slightly modified version of RANMAR given by James (James, 1990) which can be started with one integer seed. For each seed between zero and \(9.10^8\) an independent uncorrelated sequence of uniformly distributed random numbers is generated. Each sequence has a period of \(10^{34}\). These two properties of making many independent sequences with very long periods make RANMAR a suitable generator for implementation on parallel computers. In our test-simulations we used the random number generator RANMAR which was started with a different seed on each processor (for example with seed equal to the processor identity number), to yield uncorrelated sequences of random numbers.
8.3 Parallel one-particle models

Lagrangian one-particle algorithms for modelling absolute dispersion of passive tracers are generally well suited for parallelisation. The trajectories of particles can be computed independently because they represent independent realisations of the turbulent flow. Accordingly, many nodes can participate in the calculation without informing other nodes about their computations. However, one data transfer between all nodes (a global communication) per simulation is needed to compute the mean concentrations and to store them for post-processing purposes.

The parallelisation of a particle model can be illustrated by computing the mean outcome of throwing a dice. One person can throw one dice. He will have to throw it $M$ times, say to get the desired accuracy in the mean outcome. If 10 people participate in the experiment, each person throws $M/10$ times a dice and averages his own results. Then the average over $M$ throwings is calculated by accumulating the results of the ten persons. Like throwing the dice, computing particle trajectories is an independent stochastic process and thus parallelisation of the particle models can be done easily and efficiently.

Problems may arise however, if one wants to know the concentration statistics at several times, say every hour or every minute, as soon as they are computed. For this case where the computed concentration fields need to be calculated and stored at intermediate times, we studied the parallel implementation of particle models in more detail. We distributed the particles evenly over all the processors; if we have $P$ processors, each processor computes $M/P$ particle trajectories. This part is fully parallel as mentioned before in the comparison with throwing a dice. Whenever the mean concentration field has to be calculated, each processor computes its own (sub-)concentration field by averaging over its own particles. The total concentration field is equal to the average of the sub-results. We now face two problems which are closely related: how to average these sub-results stored in the processing nodes and how to store the total result in a file for post-processing purposes. Depending on the features of the parallel system, there are different methods to deal with these problems which are referred to as input and output (I/O) problems.

The most simple solution to the I/O problems is just let each processor write its own results to a central disc. Note that before any post-processing like plotting can be performed, we have to compute the total concentration field from the sub-results which have been stored in file(s). This method encounters problems due to hardware restrictions because often it is not possible to access file-systems from multiple processors simultaneously. On most multi-processor systems the disc-access is handled more or less sequentially. We refer to this method as the direct filing method.

The transport of data via the interconnection network is often much faster than disc-access, therefore instead of letting each processor write to disc, we could also sum the sub-concentrations, gather the total concentration field in one pro-
Implementation of Lagrangian models on parallel computers

cessor and let only this processor access the disc. Such a gather operation is called
a global sum. The costs of a global sum strongly depend on the network-topology
and the communication-algorithm used. We take a closer look at several pro-
cessor topologies: a hypercube, a 2d-grid and a linear array with $P$ processors
(see (Moldovan, 1993)). On the hypercube the distance between two arbitrary
nodes is at most $2 \log P$, hence the global sum takes at least $2 \log P$ neighbour-
neighbour communications no matter which processor gathers the result. In a
2d-grid topology where the nodes are arranged in a 2-dimensional equidistant
grid, a global sum where a central processor collects the answer, takes at least
$\sqrt{P}$ communications and a global sum in a corner node $2(\sqrt{P} - 1)$ data-transfers.
A global sum at the end node of a linear array takes at least $P - 1$ neighbour
communications. Hence, one global sum is cheapest on a hypercube and most
expensive on a linear array (Petkov and van der Steen, 1993).

A third method to file the concentration fields, is using a master-slave hier-
archy where the master gathers all the sub-results, computes the average and
files it while all the other processors calculate particle trajectories and send their
sub-concentrations to the master one after the other (Petkov and van der Steen,
1993).

To measure the performance of a certain implementation on $P$ processors, we
use the notion of speed-up:

$$\text{speed-up} = S_P = \frac{\text{run-time on 1 processor}}{\text{run-time on } P \text{ processors}}$$

Here, of course, the run-time of the most efficient implementation is used. The
speed-up would be perfectly equal to $P$, if we only considered the calculation of
the particle trajectories, but the I/O requirements of our application may lower
the performance.

The parallel implementation of one-particle models can be split into three
parts: a fully parallel part (the calculation of the particle trajectories), a sequential
part (the filing of the concentration fields) and an extra part due to the imple-
mentation of the model on a multiprocessor system: the parallelisation overhead.
The first two I/O strategies (direct filing and filing with a global sum) differ from
each-other in the overhead part. The parallelisation overhead depends on the
possibilities of overlapping communication with computation and on some fea-
tures of the application itself.

When running the application, at first the processors operate synchronously,
but after the first filing operation they fall out of step because processors can not
access the disc simultaneously or because processors do not participate in the
same communication actions. The processors will stay out of step until the next
filing operation where in most cases it is favourable that the processors are not
working synchronously, for example, if all processors have to access the disc. In
the best circumstances (if the computational work is large enough) all processors
can file their fields immediately without delay and causing no further costs. We
Parallel one-particle models

illustrate this phenomenon with an example. In Figure 8.1, the task-diagram is shown of an implementation of a one-particle model on a linear array with 8 nodes. We see that in the first global sum, processor with id-number eight has to wait a long time before it gets the results from processor seven which gets results from processor six, and so forth. After the first global sum and store, the nodes do not work synchronously anymore. This is an advantage for the next global sum because all nodes now receive the results from their neighbours immediately after they have completed the calculations for the next concentration field. In this case, after the first global sum, the idle time per node is reduced to a minimum.

For all implementation methods discussed here we can derive the maximum and the minimum parallelisation overhead for the case with no overlap and with maximum overlap of communications with computations. The overlap possibilities depend on the number of processors, $P$ and the number of filings, $N_s$. Other factors influencing the overhead are $T_{disc}(N)$, the time needed to file $N$ bytes, $T_{send}(N)$ the time needed to send a message of $N$ bytes to a neighbour processor and $N_p \frac{N_s}{N_p} T_{traject}$, the time needed to compute the trajectories of $N_p$ particles for the $\frac{N_t}{N_p}$ time-steps laying between two filing actions ($N_t$ equals the total number of time-steps). In table 8.1 the maximum and minimum overhead are presented for the implementation strategies discussed above.

For the direct filing method, the overlap possibilities are optimal if all processors are able to file their results within the time needed to compute the traject-
Implementation of Lagrangian models on parallel computers

<table>
<thead>
<tr>
<th>Implementation method</th>
<th>Parallelisation overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct filing</td>
<td>$(P - 1)N_sT_{disc}(N)$</td>
</tr>
<tr>
<td>Hypercube</td>
<td>$2\log P N_s T_{send}(N)$</td>
</tr>
<tr>
<td>2d-grid, mid-node</td>
<td>$\sqrt{PN_s}T_{send}(N)$</td>
</tr>
<tr>
<td>2d-grid, corner-node</td>
<td>$(2(\sqrt{P} - 1)N_sT_{send}(N)$</td>
</tr>
<tr>
<td>Linear array, end-node</td>
<td>$(P - 1)N_sT_{send}(N)$</td>
</tr>
</tbody>
</table>

Table 8.1: Parallelisation overhead of different filing strategies for no overlap of communications with computations and for maximum overlap.

ories for the next field, hence if $PT_{disc} < N_p \frac{N_s}{N} T_{trajected}$. All global sum methods perform best if the total task (summing and filing) of the gather-processor is larger than the global sum operations of the other processors.

If we assume that $T_{disc}(N) > T_{send}(N)$, then the direct-filing strategy has the largest overhead but it is independent of the filing frequency, which can be very large. For long simulations where the concentration has to be filed very often, the linear array attains the best performance. If the application is short or if the user is only interested in the final result, the best performance is obtained by using a hypercube.

The master-slave method can not be split in the same three components, because the computational part is divided into $P - 1$ equal parts instead of $P$ parts. A slave needs for the application at least $N_pN_sT_{trajected} + (N_s + P)T_{send}(N)$ time and the master ($(P - 1)T_{send}(N) + T_{disc})N_s$. The best performance is attained if the load of the master and the slaves are nearly equal.

8.3.1 Tests on the Parsytec GCel and the PowerXplorer

In the previous section, we analysed the performance of the parallel one-particle model on several multiprocessor topologies. To substantiate the theoretical analysis, we implemented a one-particle model for absolute diffusion in the atmospheric boundary layer on two parallel computers, the Parsytec GCel and the PowerXplorer. In the test case, we assumed turbulence to be homogeneous and stationary in the horizontal directions and inhomogeneous in the vertical direction, see Chapter 4 for the formulation of the equations of motion for the particles. We calculate the mean concentration in grid-points of a $21 \times 21 \times 25$ grid. The time-step needed to make accurate predictions is equal to $0.01 T_L = 10$ seconds.
The concentration fields are calculated every hour, thus every 360 time-steps. All tests cover a 10 hour time-period. The test-case is discussed in detail in (Crone et al., 1996).

We implemented this particle model on two message-passing systems located at the Interdisciplinary Centre for Complex Computer facilities Amsterdam (IC² A): a Parsytec GCel with 512 T805 transputers and a PowerXplorer with 32 nodes, consisting of a PowerPC as computation unit and a transputer for the communication. Both machines operate under the PARIX (parallel extension to UNIX) programming environment. We coded the simulation in the ANSI-C language in combination with PARIX library routines to handle the communication. The network topologies were set up with routines from the virtual topology library. More information about these two systems can be found in (Hoekstra et al., 1996).

The T805 transputer has a low sustained floating point performance of approximately 1.2 MFlop/sec and a simulation with only 2560 particles took nearly eight hours on one single processor. Therefore, we first compared the performance of multiple processors with the performance of a single processor for this small application. We calculated the theoretical speedups with the formula's given in table 8.1 where the number of flops needed for the calculation of one time-step for one particle was estimated from the algorithm. The time needed for one neighbour-neighbour communication was approximated by

\[ T_{\text{send}}(N) = \tau_{\text{setup}} + \tau_{\text{send}} N \]

where \( \tau_{\text{setup}} \) represents the time needed to set up the communication (for example for freeing buffer space) and \( \tau_{\text{send}} \) denotes the time to send one byte. The values for the flop rate, \( \tau_{\text{setup}} \) and \( \tau_{\text{send}} \) were taken from the GCel and PowerXplorer tests described in (Hoekstra et al., 1996). We performed some tests to determine \( T_{\text{disc}}(N) \). In Figure 8.2 the theoretical and measured speed-ups on 4, 16, 64 and 256 processors are shown for the different implementation methods described earlier. It was not possible to make a master-slave hierarchy with more than 32 processors. In Figure 8.2, we see that even for this small application the performance is good for all processor numbers. The speed-up deteriorates on 256 processors because the computational work is very small: only 10 particles per processor. For accurate predictions, about 2.5 million particles (about 2000 particles per grid-cell) should be used, making the computational part and thus the parallel part of the application much bigger, which leads to even better speed-ups.

We studied the scalability of the problem by measuring the execution times for a constant number of particles per processor, thus for a constant computational load per processor. In figure 8.3 the results are presented. We see that the methods using the interconnection network scale very well. The direct filing method performs well as long as the computational work is larger than the time needed for all processors to access the disc. On 256 processors the computational
Implementation of Lagrangian models on parallel computers

Figure 8.2: Theoretical and measured speed-ups on the Parsytec GCel of parallel implementations of a one-particle model for turbulent transport of passive tracers.

Figure 8.3: Timings on the Parsytec GCel of parallel implementations of a one-particle model for turbulent transport of passive tracers: for a constant computational load per processor (100 particles/processor)
load is not large enough and we see that the performance decreases rapidly. The master-slave implementation on only 4 processors suffers a load-imbalance, the master’s task is much smaller than the tasks of the slaves. We used the theoretical analysis to predict the performance of the master-slave method for larger numbers of processors and observed that when using 256 processors the performance deteriorates due to the increasing task of the master.

In Figure 8.4(a), the speedup on the PowerXplorer is shown. The PowerXplorer is computationally faster than the Parsytec but has comparable speed for communication and disc-access. For a specific application the PowerXplorer has a worse speed-up than the Parsytec, because the communication between processors is relatively expensive compared to the computational work. On the PowerXplorer, we studied the scalability with 1000 particles per processor, Figure 8.4(b). We, again, see that the direct filing and master-slave methods do not scale as well as the global sum methods.

We conclude that the direct filing and the master-slave strategies do not scale good enough. Best performances are attained when the filing of the concentration fields is done by a global sum operation using the interconnection network. The choice of the network topology depends on the number of filings in one application, for low numbers the hypercube is best and for many filings the linear array. Overall, we may say that one-particle models are well-suited for parallelisation.
There are many possibilities to overlap the communication and I/O operations with computations, hence the idle-time of processors can be minimised.

8.4 Parallel two-particle models

We have seen that the parallelisation of one-particle models for turbulent transport of passive tracers is simple and efficient. The parallelisation of a two-particle model which describes the first and second moment of the concentration is similar to the parallel implementation of one-particle models because the backward pair-trajectories can also be computed independently. Hence, these models for relative dispersion are best parallelised by distributing the pairs over the processing nodes. The backward trajectories are independent and can be computed on nodes without data-transfers with other processors. A user may want to be informed about the concentration statistics once they are computed, hence I/O problems like in the parallelisation of one-particle models may occur. Like in the one-particle model, these I/O requirements do not lower the efficiency of the parallelisation markedly. Comparable simulations to the ones performed in the previous section, showed that parallelisation of the pair model is simple and efficient.

8.5 Parallel Lagrangian models for turbulent transport and chemistry

Two Lagrangian models for turbulent transport of reactive species have been presented and we discussed these models, Chock and Winkler’s particle-grid model (Chock and Winkler, 1994a; Chock and Winkler, 1994b) and Komori et al.’s pair model (Komori et al., 1991) in Chapter 5. In Chapter 6 we presented a new two-particle model for transport and chemistry interactions. All these models are based on the computation of particle or pair trajectories, therefore they seem to be well suited for parallelisation. However, the chemistry part makes the implementation of these models on distributed memory computers more complex than the parallelisation of models for passive tracers. Next, we briefly discuss the parallelisation of Chock and Winkler’s and Komori et al.’s model. Furthermore, we consider the implementation of the new two-particle model on parallel computers and present the results of several tests.

8.5.1 Chock and Winkler’s model

It is a tedious task to parallelise Chock and Winkler’s particle grid model presented in (Chock and Winkler, 1994a; Chock and Winkler, 1994b) and discussed in Chapter 5. The turbulent transport is modelled by computing particle trajectories, hence distributing the particles over processors seems to be the best method
for parallelisation. However, the chemistry is calculated on a spatial grid, for
which domain-decomposition is the most suited solution.

If we distribute the computational work such that each processor computes
the trajectories of a small number of particles, in each time-step communication
is needed for the chemistry part. That is a global sum of the concentration field and
a global broadcast is needed to make sure that each processor has at its disposal
the concentration averages over all $M$ particles to calculate the chemistry in the
grid-cells.

If the work is distributed by means of domain decomposition, particles move
through domains randomly. Accordingly in each time-step communication is
needed to make sure that each processor has at its disposal the information of the
particles in its sub-domain. These data are used to compute the mean concentra-
tions needed in the chemistry part of the model.

Accordingly, one way or the other, many data transfers per time-step are
needed in the parallelisation of the particle-grid model.

8.5.2 Komori et al.’s model

In Komori et al’s model, the transport of particle-pairs and the chemistry between
two particles within a pair are independent of all other pairs. Henceforth, the
parallelisation of their model is similar to the parallelisation of pair models for
passive tracers. The implementation of Komori et al.’s model is simple and effi-
cient.

8.5.3 New two-particle model

The new two-particle model for turbulent transport of reactive species presented
in Chapter 6, is based on the same independent particle trajectories as the pair
models for passive tracers. The chemistry is based on the statistics of all particles.
In particular the reaction probabilities of a particle at intermediate times are com-
puted from the mean concentrations of all particles. Accordingly, parallelisation
is not obvious.

In the parallel implementation of the new model, the backward trajectories
from $(x, t_n)$ are distributed over the processors. Each processor computes the
trajectories of $M/P$ pairs, where $M$ denotes the total number of pairs and $P$ the
number of processors. The chemistry between pairs at $x$ in the time interval
$[t_{n-1}, t_n]$ ($t_{n-1} = t_n - \Delta t$) depends on the composition of the particles in the pair
and on the reaction probability of an “AB”-pair given by equation (6.11), i.e.

$$p(\Delta t|AB) = \frac{\Delta t \ k \ a_0}{1 + \Delta t \ k \ a_0}.$$}

Thus, the reaction of a pair in the period $[t_{n-1}, t_n]$ is independent of the reactions
of all other pairs and can be computed in parallel.
The reaction of a particle in the period \([t_0, t_{n-1}]\) is modelled by the probabilities \(q(t_0, t_{n-1} | A)\) and \(q(t_0, t_{n-1} | B)\) given in equations (6.15) and (6.16), i.e.

\[
q(t_0, t_{n-1} | A) = \frac{\langle c_C(x, t_{n-1}) \rangle}{\langle c_A(x, t_n) \rangle},
\]

\[
q(t_0, t_{n-1} | B) = \frac{\langle c_C(x, t_{n-1}) \rangle}{\langle c_B(x, t_n) \rangle}.
\]

Accordingly, in the parallel implementation the ensemble means \(\langle c_C(x, t_{n-1}) \rangle\), \(\langle c_A(x, t_n) \rangle\) and \(\langle c_B(x, t_n) \rangle\) have to be computed by a global sum and communicated to all processors before the nodes can compute the reaction probabilities and thus the reaction of particles in the period \([t_0, t_{n-1}]\).

The parallel version of the algorithm of the new transport/chemistry model does not change considerably from the sequential version. On each computational node, Algorithm 9 is performed with \(M/P\) pairs of particles. In step (2) the average over all \(M\) pairs is computed from the sub-results in all processors by means of a global sum communication. Next a global broadcast is performed to make sure that each processor has at its disposal the mean concentrations without reaction \(\langle c_A(x, t_n) \rangle\) and \(\langle c_B(x, t_n) \rangle\), and the mean concentration of species C at the previous time \(t_{n-1}, \langle c_C(x, t_{n-1}) \rangle\). In this way, each processor can go on with the next step, (3) where the reaction probabilities \(q(t_0, t_{n-1} | A)\) and \(q(t_0, t_{n-1} | B)\) are computed. Another communication step is needed whenever the concentration statistics have to be computed and stored for post-processing purposes. Accordingly, to handle step (5) of Algorithm 9 another global communication is needed. This problem is similar to the I/O problem of the one-particle model for absolute dispersion, we refer to the discussion in Section 8.3 for more details about handling this problem.

We tested our parallel version of the new two-particle model by implementing it on an IBM-SP2 located at the Academic Computing Services centre in Amsterdam. This SP2 is a collection of 76 very powerful IBM RS/6000 processor units (‘nodes’). The data transfer between nodes is handled by the ‘switch’, a high-speed network. More about the SP2 in general and the SP2 at the centre in Amsterdam can be found at the World Wide Web pages HTTP://WWW.SARA.NL/.

We programmed the model in ANSI-C in combination with MPI-F, IBM’s library implementation of the standard portable message-passing library definition, MPI (Message Passing Forum, 1994).

We have performed measurements for a typical run of \(2.10^5\) particle-pairs, \(\Delta t = 0.01\ T_L\), reaction-rate \(k = 2\) and a total simulation time equal to \(5\ T_L\). The test is similar to the ones discussed in Chapter 7 in Figure 7.2. The results are shown in Figure 8.5. We have seen previously that in each time-step a global sum and broadcast is needed, hence the computations are not fully parallel. Nevertheless, we can see in Figure 8.5 that the speed-up of such a parallel implementation on an IBM-SP2 is good. On 32 processors the speed-up of 30 corresponds to an efficiency of 93 per cent. On 64 processors, the time needed to compute the evol-
8.6 Conclusions

We have seen that the Lagrangian particle models for turbulent transport of passive tracers are easy to parallelise: calculating particle trajectories is fully parallel, the only problem is the filing of the concentration fields. The parallel implementations on distributed memory multiprocessor systems are efficient and scale very well.

In comparison with the large communication requirements of parallel Eulerian models, the data transfers needed in parallel Lagrangian models are negligible. Eulerian models require not only data transport between nodes for the I/O but also several data transfers between nodes in each time-step.

The Chock and Winkler’s particle-grid model is not suited for parallel implementation. Komori et al.’s model is equivalent to a pair model for passive tracers in the sense that pairs disperse and react independently from other pairs, hence parallelisation of their model is simple and efficient. We have seen that in spite of the global sum and broadcast communications in each time-step, the new two-particle model for transport and chemistry can be implemented on a multi-processor computer in a simple and efficient way.