Appendix A

Parallelization of wave packet propagation programs

A.1 Concepts of parallel programming

The latest generation of supercomputers are the so-called massively parallel machines. They consist of a large number of *processing elements*, or nodes — individual high speed processors with their own local memory. The processing elements are connected to a high speed internal network, which enables efficient exchange of data between them (see Figure A.1). A typical massive parallel computer has several hundred nodes. The data to be operated upon is distributed between the processing elements, which then operate on their locally stored data, and exchange it when required by the algorithm.

There exist two fundamentally different programming concepts for such architectures, namely *MIMD* and *SIMD*. *MIMD* stands for *Multiple Instruction Multiple Data*. Within this approach, different processing elements operate on different data, and run different instructions on it. A possible application of this technique for quantum dynamical problems would be to use several processing elements for the

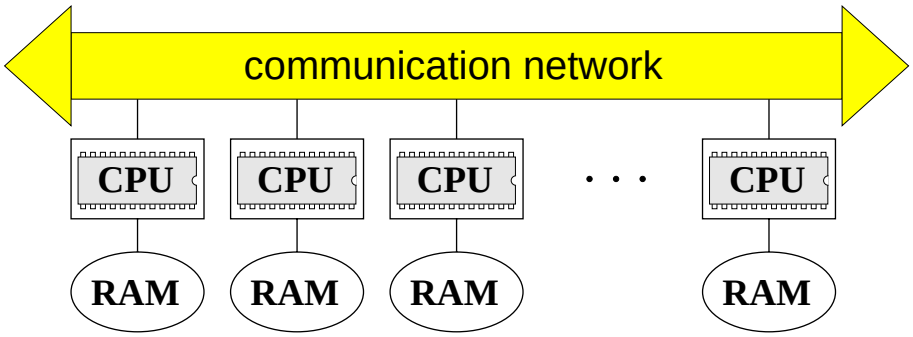

Figure A.1: A scheme of a generic parallel computer. Each processor (CPU) has its own local memory (RAM), and is connected to a high-speed internal communication network.

propagation of the wave packet, and dedicate the others to analysis of the generated wave functions.

SIMD, or *Single Instruction Multiple Data* approach requires that all the processing elements run the same program on different data portions, which are evenly distributed between them. This method is primarily used for numerical mathematics problems, especially for matrix operations and for the solution of systems of differential equations. This is the approach used here for the solution of the Schrödinger equation.

Every program which runs on a parallel machine, consists of the sequence of blocks of the following structure [158]:

- Local operations
- Data exchange
- Synchronization

First the individual processors operate on the data available locally to each. After a certain point, to continue the calculation, the processing element has to get additional data and/or provide its own to a different processing element. The processing

element then makes requests for input-output (I/O), which are put into designated queues. No data exchange takes place until the final step, the synchronization, when all the processors are halted, and the queued I/O requests are carried out. Upon completion of the exchange, the processing elements simultaneously resume operation, working on the newly acquired data. These three fundamental steps comprise one *superstep* of computation. A parallel program consists of at least one superstep.

Care should be taken to minimize the number of supersteps in an algorithm, because the synchronization is a relatively time-consuming operation, the more so, the greater the number of processing elements used. Since synchronization is required for data exchange, it follows that the algorithms, which do not require movement of large blocks of data benefit the most from parallelization.

To implement the above mentioned techniques in actual programs, several code libraries can be used. In this work the *Oxford BSP Library* [159, 160] has been used, which provides the subroutines for data exchange and synchronization for the *Fortran* and *C* programming languages.

A.2 Parallelization of three-dimensional FFTs

The techniques described below were developed by Bisseling and Sundermann [75, 159, 160]. Their successful applications to to several three-dimensional wave packet propagation problems can be found in this thesis.

The heart of a wave packet propagation program is the fast Fourier transformation, and therefore the main task of parallelization of these programs is to enable the efficient evaluation of the transformation of data distributed between the processing elements. For the three-dimensional problems, the technique of *slab decomposition* has proven to be effective. In this approach the three-dimensional wave function in coordinate space is divided into P slices, or slabs, along one of the coordinates (designated as z). The slices are then distributed between P processors, each of which has all the data necessary for the Fourier transformation along the other two

(x and y) coordinates (see Figure A.2).

To complete the transformation to the momentum space, one has to perform the FFT along the z coordinate. For this, the abovementioned coordinate space distribution is unsuitable. Therefore, the data are rearranged into P columns of equal size by slicing along x and y coordinates. After the rearrangement, the FFT in the z coordinate can be performed, and the desired wave function in momentum space can be obtained (Figure A.2).

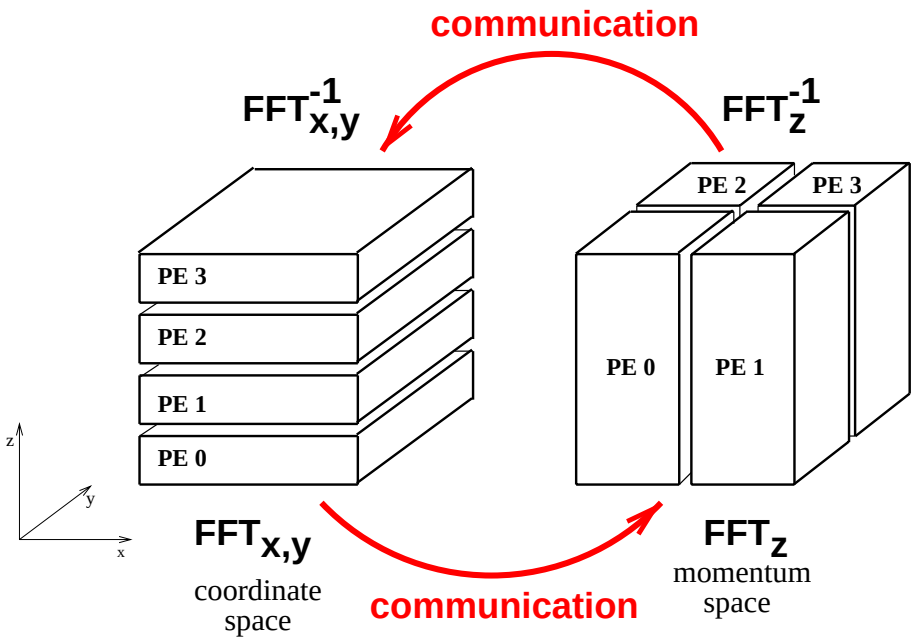

Figure A.2: Two types of distribution of a three dimensional wave function. $P = 4$

For the inverse transformation, the procedure is reversed. First, the inverse FFT along the columns of the momentum space wave function is performed, then the data is rearranged to the coordinate space representation, where the inverse two-dimensional FFT is applied. In contrast to the sequential propagation programs, the transformation to/from coordinate space cannot be achieved "in place", without allocating additional storage. The parallel propagation programs require separate data structures for the coordinate and momentum space wave functions.

For a detailed discussion of the problems of parallelization of the wave packet propagation, as well as for formal description of the data exchange algorithm, the reader is referred to the work of Sundermann [75].

A.3 Notes on the efficiency

The three-dimensional calculations of this work were performed on the Cray T3E massively parallel supercomputer of the Konrad-Zuse-Zentrum für Informationstechnik, Berlin (ZIB). Typically, the number of processing elements used varied from 16 for test calculations to 128 for production runs. The scaling properties of the algorithms with respect to the number of processors employed are illustrated in Figure A.3, where the speedup factor (the ratio of the time required for a benchmark run on one processor to the run time of the same program on P processors) as a function of the number of processors used. In the case of infinitely fast communication, this ratio will be equal to P (solid line on Figure A.3). The measured performance

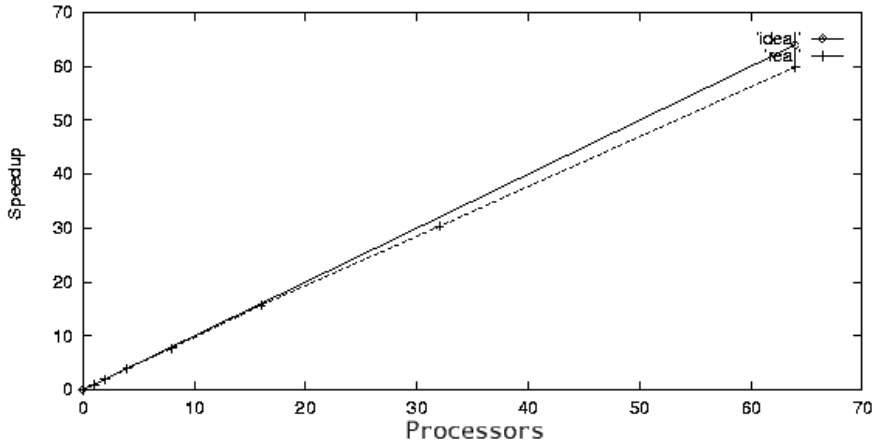

Figure A.3: Dependence of the performance gain on the number of processors utilized for an exemplary propagation of Na_3 molecule (adopted from [75]).

(Figure A.3, dashed line) was found to be very close to the linear scaling limit, indicating that the parallelization technique employed is quite adequate. Even if

it did not prove to be the case, increasing the number of processing elements used is unavoidable for very large scale problems, since the massively parallel computers allow these to use the amounts of memory otherwise unavailable.

In order to give the feeling of the actual times required for the propagation programs to run, it is worth mentioning, that the propagation of the wave packet for Ag_3 (Chapter 3) on 8,000,000 element grid for 2 ps, involving 10,000 time steps, took approximately two hours of computation time on a Cray T3E using 128 processing elements.