Fine Grain Parallelism for Discrete Variable Approaches to Wavepacket Calculations

Daniele Bellucci[1], Sergio Tasso[1] and Antonio Laganà[2]

1 - Dipartimento di Matematica e Informatica, Università di Perugia, 06123 Perugia, Italy
2 - Dipartimento di Chimica, Università di Perugia, 06123 Perugia, Italy

Abstract. The efficiency of some parallel models and structures when applied to wavepacket reactive scattering calculations is discussed by revisiting some existing time dependent quantum procedures. The achievement of computational efficiency was found to be closely related to the parallel model adopted with the fine grain being always less efficient than the coarser grain ones. In spite of this the fine grain parallel model was found to be useful for dealing with excessively large matrices.

1 Introduction

Scattering properties of reactive systems can be evaluated using different approaches[1]. Among them those based on classical mechanics are naturally parallelized at large grain since each trajectory calculation is an independent computational task. On the contrary, approaches based on quantum mechanics are difficult to parallelize because of the spread nature of the wavefunction describing the quantum system. The spread nature of the quantum representation of the system shows up in the use of a large basis set or of a large pointwhise representation of the wavefunction.

Quantum wavepacket methods differ from time-independent quantum methods in that they integrate in time the time-dependent Schrödinger equation starting from a known quantum state of the reactants.

In the numerical procedure we use (TIDEP), only the real part of the wavepacket is propagated [2]. For the generic atom-diatom collinear (two mathematical dimensions) reactions A + BC(v, j) the initial wavepa-cket is set up by expressing the wavefunction in terms of the initial diatomic molecule BC wavefunction and its analysis is performed at a cut corresponding to a large fixed B-C vibrational coordinate [2]. To start the propagation, the initial wavepacket (the vibrational, v, and rotational, j, wavefunction of the diatomic reactant times a normalized Gaussian function and a phase factor giving a relative momentum towards the interaction region[2]) is placed in the entrance channel. The method can be implemented in a way that only the real component of wavepacket can be explicitly propagated [2] and a collocation method can be used. Accordingly, the potential and the wavefunction are represented by their values on a regular grid that must be large enough to contain the initial wavepacket, the region where the analysis line is drawn, and the interaction region. The grid must also be fine

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2331, pp. 918–925, 2002.

enough to accurately describe the structure of the wavefunction. The real part of the wavepacket is propagated in time until it has mainly been absorbed near the edge of the grid. This reduces the calculation to a continuous manipulation of a certain number of multidimensional matrices.

As already pointed out elsewhere[3], a coarse grain model is better suited to parallelize TIDEP. Coarse grain parallel implementations of the code and related advantages and disadvantages are discussed in section 2. Finer grain parallelization models are, under certain respects, more interesting. They act, in fact, at the innermost level of the matrices implying a highly repetitive execution of some operations that is a favourable case for parallelism. In addition, the bigger dimensionality of the matrices used in these approaches makes more likely the possibility that the memory limits of the machine used are hit. In section 3, we discuss several details of a fine grain parallel implementation of TIDEP.

2 The coarse grain parallelization of TIDEP

The parallelization tool used in our work is MPI. In TIDEP, calculations are performed for a given range of energy, at a fixed value of the vibrotational quantum number (vj) of the reactant diatom and at a single value of the total angular momentum quantum number J and parity p. Therefore, the coarsest grain level of parallelism that can be adopted is the one distributing the calculation for a given quartet of v, j, J, p quantum numbers. In this case, due to the increasingly (with J) different characteristics of the various tasks, the best choice is to adopt a task farm organization dynamically assigning the computational workload [3]. This very coarse grain approach was fruitfully implemented on a cluster of powerful workstations. In this case, however, the highest J calculations require increasingly longer times to run, make the check for convergency with partial waves difficult and the imbalance of the load rapidly growing.

A next lower level of parallelization of TIDEP is the one based on the combined distribution of fixed J, p and Λ calculations. As it has been already pointed out above, while it is natural to distribute fixed J calculations (these calculations are fully decoupled) the decoupling of Λ is not natural since one has to introduce physical constraints of the centrifugal sudden type (*i.e.* the projection of J on the z axis of the body fixed frame is kept constant during the collision). This allows to perform separately the step-propagation of the wavepacket for blocks of fixed Λ values and recombine the various contributions only at the end of each propagation step. This is a key feature of the adopted computational scheme since it allows a decomposition of the domain of the wavepacket in J blocks of size equivalent to that of block $J = 0$. This converts a request for increasing the node memory proportionally to J into a request for increasing the number of nodes proportionally to J while keeping the node memory constant.

To carry out the calculations the O(^{1}D)+HCl atom diatom reaction [4, 5] was taken as a case study. Accordingly, the mass values were chosen to be 15.9949 amu, 1.00783 amu and 34.96885 amu (for O, H and Cl, respectively). The energy range covered by the calculation was approximately 1 eV, the initial vibrotational

state used for the test was $v = 0$ and $j = 0$. The potential energy surface used for the calculations is described in ref. [4, 5] where other details are also given. Two sizes ((a) 127 × 119 points, (b) 251 × 143 points) were used for the dimension of the R' and r' matrices while the angular part was expanded in a set of basis functions (80 in both (a) and (b) cases). Time propagation was carried out for about 40000 steps to properly diffuse the wavepacket. Production runs took about 3 weeks on a single node of a Sylicon Graphics PowerChallenge supercomputer at $J = 0$. A first version of the parallel code was run [5] on the Cray T3E at the EPCC of Edinburgh (UK) for the simplest case of $J = 0$ and $J = 1$ in which only three pairs of J and Λ values are considered and only three nodes are used. Measured speedups are 2.6 for the propagation grid (a) and 2.5 for the propagation grid (b).

The model was then generalized to higher J values. In this generalized model node zero was exclusively devoted to act as a master and the I/O was decentralized. At the same time, the feature of carrying out fixed J calculations in pairs by associating one high J value with its complement to the maximum value of J was adopted in order to keep the number of processors used constant. To evaluate the performance of this model, the calculations were carried out on the Origin 3800 at Cineca (Bologna, I) using the same set of parameters adopted for the grid (a) test described above yet reducing the basis set expansion for the angular part to 10. Table 1 shows the value of the percentual increase of the node computing time with respect to that of the node carrying out the $J = 0$.

Table 1. Percentual time increment (with respect to $J = 0$)

J	1	2	3	4	5	6	7	8	9	10	11	12	13
% time	0.9	1.8	3.2	6.4	6.4	8.2	15.5	17.3	16.4	19.1	20.0	21.8	22.7

As clearly shown by the results reported in the Table 1 the computing time per node (averaged over the various values of Λ) increases with J up to about 20%. This indicates that, although one has to pay an extra price to increase the maximum allowed value of J, for this parallel model the increase of communication time associated with an increase in the number of allowed Λ only marginally reduces the advantage of having distributed the calculations over several computing nodes.

3 The fine grain parallelization of TIDEP

The key feature of TIDEP is the iterated use of a time propagator which is characterized by a determined and recursive structure of matrix operations, such as the fast Fourier transform, which could allow a re-use of the resources. This

requires, however, that the matrix operations (multiplications, transpositions, Fourier transforms) of the algorithmic sequence are performed in a proper way.

At fine grain level, this means to focus the parallelization work on the routines BLAS DCOPY and DAXPY. In fact TIDEP calls these routines more than hundred thousands times per propagation. If use is made of propagation techniques involving a continuous transformation between coordinate and momentum space, the use of the BLAS routines is accompanied by the use of a Fast Fourier Transform routine that makes the computational burden even heavier.

An alternative approach is that based on the Discrete Variable Representation (DVR) method. This is based on the reiterated application of operations like

$$\mathbf{H} = \mathbf{A} \cdot \mathbf{C} + \mathbf{C} \cdot \mathbf{B}^{\mathbf{T}} + \mathbf{V} \odot \mathbf{C} \tag{1}$$

where $\mathbf{A}$ and $\mathbf{B}$ are the matrix representations of the two terms (one for each dimension) of the Laplacian operator, $\mathbf{C}$ is the collocation matrix of the wavefunction, $\mathbf{V}$ is the matrix representation of the potential operator (accordingly $\mathbf{V} \odot \mathbf{C}$ is the direct product of the single component $\mathbf{V}$ matrix with $\mathbf{C}$).

```
LOOP of iv from 1 to nv
  LOOP of ir from 1 to nr
    h(ir, iv) = 0
  END loop of ir
END loop of iv
LOOP of iv from 1 to nv
  LOOP of i from 1 to nr
    LOOP of ip from 1 to nr
      h(i, iv) = h(i, iv) + a(i, ip) · c(ip, iv)
    END loop of ip
  END loop of i
END loop of iv
LOOP of i from 1 to nr
   LOOP of iv from 1 to nv
     LOOP of ivp from 1 to nv
       h(i, iv) = h(i, iv) + c(i, ivp) · b(iv, ivp)
     END loop of ivp
   END loop of iv
END loop of i
LOOP of iv from 1 to nv
   LOOP of i from 1 to nr
     h(i, iv) = h(i, iv) + v(i, iv) · c(i, iv)
   END loop of i
END loop of iv
```

Fig. 1. Pseudo code for the section of the *av* routine associated with eq. (1)

In the reduced dimensionality version of TIDEP used for the parallelization, above matrix operations are performed inside the routine *av*. Inside *av*, two ma-

trix times vector and one vector times vector operations are performed according to the computational scheme given in Fig. 1.

When all the matrices involved are distributed per (groups of) rows among a certain number of nodes, all the operations sketched above imply a quite significant amount of communication to allow the nodes have the updated version of the matrices involved.

As already mentioned, the fine grain approach has the advantage of allowing an increase of the size of the involved matrices beyond the capacity of the node memory. In this approach, in fact, the request of memory is drastically reduced by partitioning the space (and momentum) domain. The choice we made was to partition the representation domain by rows and to adopt a management of the memory that takes into account the hierarchy of the memory including the I/O levels.

Accordingly, out of eq. (1) one obtains

$$Row(i,H) = \sum_{k=1}^{nr} A(i,k) \cdot Row(k,C) + Row(i,C) \cdot B^T + Row(i,V) \odot Row(i,C)$$

whose algorithmic structure is given in Fig. 2.

In this algorithm the matrix C is always handled by rows. The parallel model adopted is a task farm that performs the calculation for the first two operations of the right hand side of eq. (1) at worker level and leaves the third one with the master. Each worker has access to a local (unshared) secondary memory in which the elements of the proper partitions of A, B and C are stored. In the startup phase the C matrix is distributed to the workers (this avoids possible subsequent I/O conflicts). In the same phase the first row of A is distributed by the master that, after reading sequentially the rows of C, forwards the pairs $\langle A(1,k), Row(k,C)\rangle$ to the workers using a *roundrobin* policy. This implies the use of a buffer of nr elements in which, at each time, the C row is stored. The dimension of the buffer is determined by the number of workers (M). Accordingly, the ith worker is assigned the pairs $\langle A(1,k), Row(k,C)\rangle$ with $k \equiv i \bmod M$.

Rows of matrix A and B are stored in the same local secondary memory. Each worker after receiving the row vector of C performs its multiplication

$$\sum_{k \in \mathcal{D}_i} Row(k,C) \cdot A(1,k) \tag{2}$$

where $\mathcal{D}_i = \{x \in N | x \equiv i \bmod M\}$.

The product $Row(1,C) \cdot B^T$ is then computed by multiplying inside each node the first row of C by the related partition of rows of B (avoiding so far the transposition). These terms are then summed to the quantity (2). The sum of the vectors computed by the workers and the master determines the first row of H. This is performed via a *reduce* called by all the farm processes in which the master deals with $Row(1,V) \odot Row(1,C)$ and the workers the computed vectors. The sum is saved into the logical space of the master that stores it into the secondary memory space assigned to the matrix H.

```
{Let nr = nv}
LOOP of i from 1 to nr
   ReadFromFile Row(i, A)
   ReadFromFile Row(i, V)
   ReadFromFile Row(i, C)
   LOOP of j from 1 to nv
      Temp(j) = 0
   END loop of j
   LOOP of k from 1 to nr
      ReadFromFile Row(k, C)
      LOOP of j from 1 to nv
         Temp(j) = Temp(j) + A(i, k) · C(k, j)
      END loop of j
   END loop of k
   LOOP of j from 1 to nv
      Temp(j) = Temp(j) + V(i, j) · C(i, j)
   END loop of j
   LOOP of w from 1 to nr
      ReadFromFile Row(w, B)
      LOOP of j from 1 to nv
         Temp(w) = Temp(w) + C(i, j) · B(w, j)
      END loop of j
   END loop of w
   LOOP of j from 1 to nv
      H(i, j) = Temp(j)
   END loop of j
   WriteToFile Row(i, H)
END loop of i
```

Fig. 2. Sequential version of the section of the *av* routine associated with eq. (1)

To minimize the worker *idle* time, the master *broadcasts* to all workers the pair $\langle Row(2, A), Row(2, C)\rangle$ before entering the state of waiting for the completion of the *reduce*. This allows the worker to immediately resume their calculations after executing the *reduce*. This has the effect of overlapping (and therefore masking) the time needed for the completion of the *broadcast* through the computing time of the workers. It is worth noting that the master performs the *broadcast* while no other process of the farm attempts an access to the communication channel. As a result, the performance is not affected by possible network access conflicts.

Then each worker W_i can access at the same time its own block of C rows stored in the startup phase with no conflicts and alike in the previous startup phase performs the sum of scalar products by taking from the vector $Row(2, A)$ the proper elements of index i modulus the number of workers (M). In a similar way, the C rows of index greater than 1 are generated.

Test runs have been performed on a Beowulf made of 8 monoprocessor (Pentium III 800 MHz) nodes having 512 MB of central memory using square matrices

of size 512, 768 and 1024. As shown by Table 2, the elapsed times measured for the parallel version are definitely smaller than those of the serial version. As a result, speedups are significant and the advance apparent especially if a comparison is made between with the previous version for which the elapsed time of a five processor parallel run would hardly break even with that of a single processor sequential run[3]). It is particularly worth noting also that, in the investigated range of matrix sizes, the speedup is constant.

Table 2. Elapsed times and speedups

Matrix size	Seq. time/s	Par. time/s	Speedup
512x512	1549.1	231.1	6.7
768x768	5193.2	768.3	6.8
1024x1024	12271.0	1814.2	6.8

4 Conclusions

The need for pushing the parallelization of wavepacket reactive scattering codes to a fine level in order to deal with matrices of large dimensions associated with the solution of problems of high dimensionality has been discussed. The code considered by us for parallelization makes use of a collocation method and a discrete variable technique. Then the domain is decomposed in a way that the sequence of matrix operations can be performed by minimizing the time needed for the reorganization of the matrices during the operations and by overlapping communication to execution. The study has shown that in this way it is not only possible to deal with systems whose collocation matrices are too large to be accomodated in the local memory of the nodes but it is also possible to achieve a significant parallel speedup.

5 Acknowledgments

This research has been financially supported by MIUR, ASI and CNR (Italy) and COST (European Union).

References

1. Laganà, A., Innovative computing and detailed properties of elementary reactions using time dependent approaches, Computer Physics Communications, **116** (1999) 1-16.

2. Balint-Kurti, G.G., Time dependent quantum approaches to chemical reactions, Lecture Notes in Computer Science, **75** (2000) 74 - 88.
3. V. Piermarini, L. Pacifici, S. Crocchianti, A. Laganà, Parallel models for reactive scattering calculations, Lecture Notes in Computer Science **2110** (2001) 194 - 203.
4. V. Piermarini, G. Balint-Kurti, S. K. Gray, G.F. Gogtas, M.L. Hernandez, A. Laganà, and M.L. Hernandez, Wave Packet Calculation of Cross Sections, Product State Distributions, and Branching Ratios for the O(^{1}D)+ HCl Reaction, J. Phys. Chem. A **105(24)** (2001) 5743-5750.
5. V. Piermarini, A. Laganà, G. Balint-Kurti, State and orientation selected reactivity of O(^{1}D)+ HCl from wavepacket calculations, Phys. Chem. Chem. Phys., **3** (2001) 4515-4521.