Parallel Models for a Discrete Variable Wavepacket Propagation

D. Bellucci[2], S. Tasso[2], and A. Laganà [1]

[1]Department of Chemistry, University of Perugia, Via Elce di Sotto, 8,
06123 Perugia, Italy
lag@dyn.unipg.it

[2] Department of Mathematics and Informatics, University of Perugia,
Via Vanvitelli, 1, 06123 Perugia, Italy
daniele@dyn.unipg.it, sergio@unipg.it

Abstract. The parallelization of a code carrying out the discrete variable propagation of a wavepacket is discussed. The performances obtained from a Task Farm model are compared with those obtained from a Pipeline model.

1 Introduction

Atom diatom reactive scattering calculations are often carried out using wavepacket techniques. The difficulty of carrying out these calculations lies in the fact that as the total angular momentum increases the matrices involved become rapidly so large that they make the computer code quite inefficient. In fact, the algorithms usually utilized to carry out the time propagation of the wavepacket require at each time step the use of all the elements of the matrices. This makes it inconvenient to apply a fine grain parallelization of the code.

As an alternative, for our time dependent code (in which the wavepacket is propagated in the *AV* routine by dealing only with its real component[1]) we adopted a more localized Discrete Variable technique that at each time step (τ) performs a series of matrix operations that can be schematized as follows:

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

Eq (1) is recursive since the value of C is taken at time $\tau - 1$ while that of G is taken at time τ (the value of C at time τ, in turns, depends on that of G at the same time τ even though, in order to simplify the notation, we have dropped the time subindex). For simplicity we also assume that all the matrices involved in the calculation are square matrices of order nr. Such a constraint, however, can be easily removed with no prejudice for the results.

The starting point of our study is the sequential version of *AV* (see Fig. [1] for its pseudocode). By adopting a domain representation *by rows* the j-th row of matrix G can be expressed as follows:

$$Row(j,G) = \sum_{k=1}^{nr} A(j,k) \cdot Row(k,C) + Row(j,C) \cdot B^T + Row(j,V) \odot Row(j,C). \tag{2}$$

To optimize the usage of space all the matrices were stored on the secondary memory minimizing the occupation of the main memory. When adopting a domain representation by rows it is easy to show that:

- Only a row vector is needed to carry out the calculations of Eq (2);
- The first term of Eq (2) ensures that accesses to I/O are optimized by storing each matrix *by row*;
- The *transpose* operation is avoided by multiplying $Row(j,C)$ by B rows since they are the columns of B^T [2].

```
Do $j = 1, n$
   $Row(j,G) = 0_n$
   ReadFromFile ($Row(j,A)$)
   Do $k = 1, n$
      ReadFromFile ($Row(k,C)$)
      $Row(j,G) = Row(j,G) + A(j,k) \cdot Row(k,C)$
      If ($k = j$) then
         Keep in main memory $Row(j,C)$
      EndIf
   EndDo
   Do $h = 1, n$
      ReadFromFile ($Row(h,B)$)
      $G(j,h) = G(j,h) + Row(j,C) \cdot Row(h,B)$
   EndDo
   ReadFromFile ($Row(j,V)$)
   $Row(j,G) = Row(j,G) + Row(j,V) \odot Row(j,C)$
   WriteToFile ($Row(j,G)$)
EndDo
```

Fig. 1. Pseudocode of the sequential version of the *AV* routine.

2 The Task Farm Model

The first attempt to parallelize *AV* was performed using MPI and a Task Farm model [3]. In the startup phase the Master process distributes the rows of C using a cyclic policy. This implies that in the startup phase if $j \equiv i \bmod M$ the vector $Row(j,C)$ is sent to the Worker W_i (with M being the number of scheduled Workers). At the end of the startup phase each Worker has stored the rows received from the Master in a local (unshared) secondary space storage hereafter called $Dataset(W_i, C)$. In the same way, all the elements of the matrices A and B referred by the Worker W_i are stored in a similiar *Dataset* during the wavepacket inizialization (immediately before the first step $\tau = 0$).

To carry out the subsequent operational phase the Master process adopts a scheduling policy to *broadcast* $Row(j, C)$ (that is needed to perform the calculation of the second term of Eq (1)) to each Worker. Each scheduled Worker loads from a local (unshared) secondary memory space all the elements needed to carry out the calculation of:

– $w_i = \sum_{k \in D_i} A(j,k) \cdot Row(k, C)$
– $w_i(h) = w_i(h) + Row(j, C) \cdot Row(h, B) \qquad \forall h \in E_i$

where:

– i is the index of the Worker process;
– j is the index of the Row of G to be calculated;
– $D_i = [i]_M$ with M being the number of scheduled Workers;
– E_i is the set of contiguous indices assigned to Worker W_i allowing an optimum load balancing among the Workers.

Through a reduce operation the Master reassembles the resulting rows of matrix G. The pseudocodes of the Master and the Worker processes are given in Figs. [2],[3] respectively.

```
Process Master
Do $i = 1, nr$
    ReadFromFile ($< Row(i,V), Row(i,C) >$)
    MPI_Bcast (Every Worker, Row(i,C))
    $T = Row(i,V) \odot Row(i,C)$
    MPI_Reduce (Master, T, $Row(i,H)$)
    WriteToFile ($Row(i,H)$)
EndDo
```

Fig. 2. Pseudocode of the operational phase of the Master Process of the Task Farm model.

Several speedup measurements were performed using a Cluster of Linux Workstations on which the Gnu compiler and the Lam-MPI library were made available. The size of the matrices was varied from 512 to 1024. Speedups achieved when using the Task Farm Model are shown in Fig. [4]. They scale as the ideal speedup indicated for comparison in the same figure.

3 An Improved Task Farm Model

A first improvement of the simple Task Farm model (hereafter called STF) was introduced by changing the scheduling policy. The cyclic policy introduces,

```
**Process** $W_i$
**Do** $j = 1, nr$
   **MPI_Bcast** (From Master, $Row(j, C)$)
   $w_i = 0$
   **Read** ($\{A(j, k) | k \in D_i\}$ from $Dataset(W_i, A)$)
   **ForEach** $k \in D_i$
      **Read** ($Row(k, C)$ from $Dataset(W_i, C)$)
      $w_i = w_i + A(j, k) \cdot Row(k, C)$
   **EndFor**
   **ForEach** $h \in E_i$
      **Read** ($Row(k, B)$ from $Dataset(W_i, B)$)
      $w_i(h) = w_i(h) + Row(j, C) \cdot Row(h, B)$
   **EndFor**
   **MPI_Reduce** (Master, $w_i$)
**EndDo**
```

Fig. 3. Pseudocode of the i-th Worker process operational phase.

in fact, a certain amount of overhead due to the fact that the communication system has to be initialized everytime a block of data is sent.

The improved Task Farm (ITF) minimizes such an overhead adopting a block scheduling policy. Each Worker has to receive at least a matrix partition during the startup phase. Thus the Master process needs at least M steps to complete the scheduling operation (remember that M is the number of Workers used). The block policy consists of sending a block of contiguos rows of C to each Worker.

To estimate the advantage gained in terms of a reduction of comunication time when using the block policy of the ITF model instead of the cyclic one we plot in Fig. [5] the amount $Dt = t_{STF} - t_{ITF}$ defined as the difference between the STF and ITF the elapsed times (t).

As apparent from Fig. [5] the *block* scheduling policy progressively outperforms the cyclic one on the four node Beowulf as the size of the matrices grows. Such an advantage slowly decreases as the number of processes increases although it remains substantially constant in percentage with respect to an increase of the size of the matrices and of the number of processes. This result, however, exploits also the fact that the A, B and V matrices remain unchanged during the calculation. Accordingly, in the ITF model there is not need to repeat their distribution to the Workers at every time step.

Related speedups are given in Fig. [6]. The figure shows that the ITF model, in addition to improving over the STF one, still scales as the ideal curve when the dimension of the matrices increases.

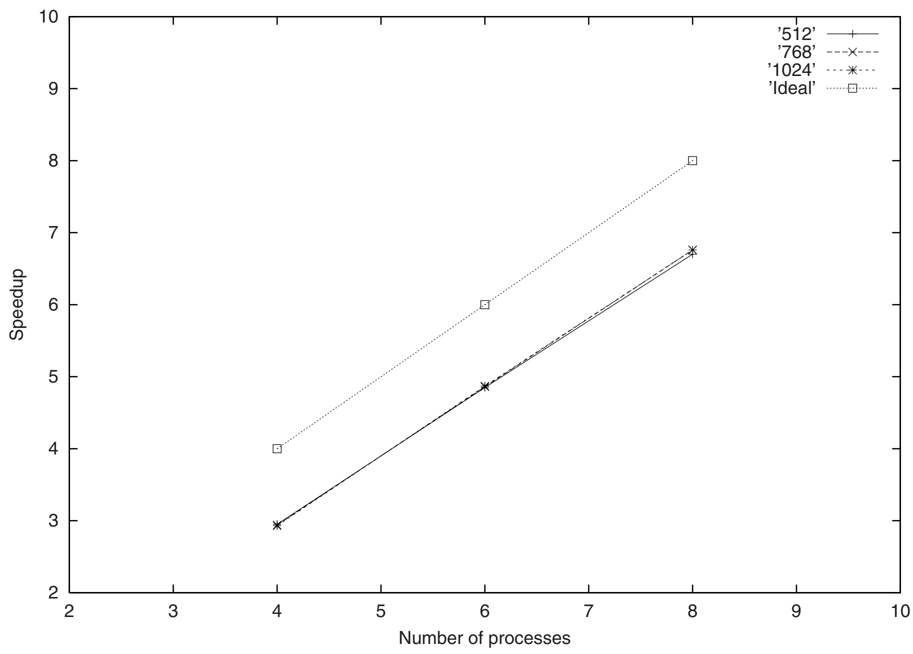

Fig. 4. Speedups obtained for the STF model on the 8 node Beowulf plotted as a function of processes.

4 The Pipeline Model

During the operational phase of the Task Farm the calculation of $A \cdot C + C \cdot B^T$ is performed by all available Worker processes in a way that overlaps the calculation of $V \odot C$ performed by the Master process. In this way related sequentiality is suppressed and possible limiting effects on the speedup are avoided.

The Pipeline solution differs from the Task Farm ones in that the calculation of G is partitioned among the scheduled processes. Therefore, in an M stage Pipeline the i-th process[1] W_i calculates the following vector:

- $w_i = \sum_{k \in D_i} A(j,k) \cdot Row(k,C)$
- $w_i(k) = w_i(k) + Row(j,C) \cdot Row(k,B) + V(j,k) \odot C(j,k) \qquad \forall k \in D_i$

where

- j represents the index of the row of G to be calculated;
- D_i is a set of contiguous indices assigned to process W_i allowing an optimum load balancing among the stages.

[1] Each process is now a stage of the pipe.

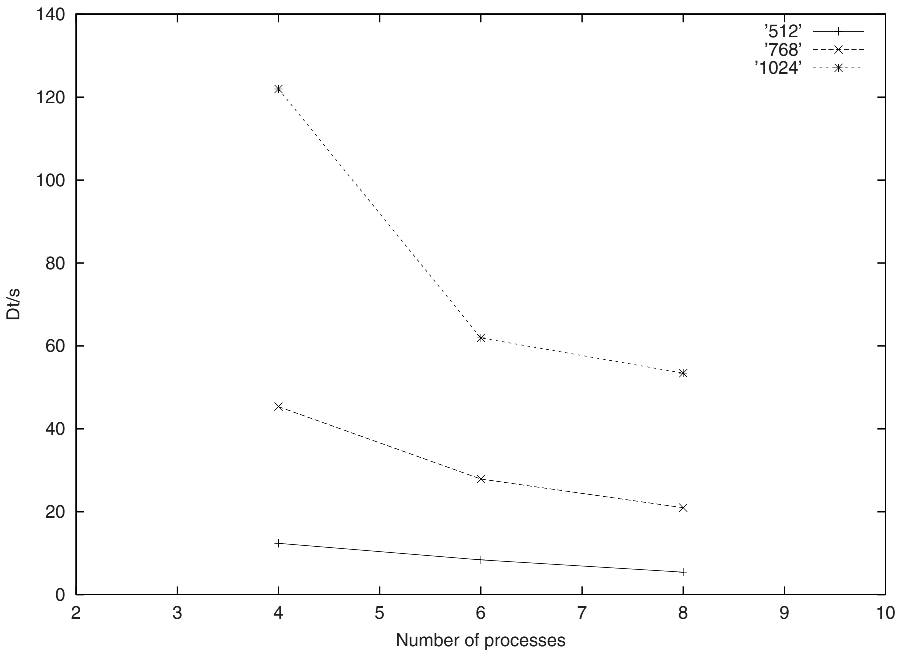

Fig. 5. Time difference between cyclic and block policy.

Distributed calculations of $Row(j, G)$ can be described as follow: the W_1 stage calculates w_1, loads $Row(j+1, C)$ and sends the t-uple $< Row(j+1, C), w_i >$ to W_2. For each $i = 2, \cdots, M-1$ stage W_i receives the t-uple from W_{i-1}, performs the sum $w_i = w_i + w_{i-1}$ and then sends the new t-uple $< Row(j+1, C), w_i >$ to W_{i+1}. The last stage carries out the final calculation of $Row(j, G)$ summing its vector (w_M) to the second term of the t-uple received from W_{M-1} which contains $\sum_{i=1}^{M-1} w_i$. Like in the Task Farm model each vector needed to calculate w_i by stage W_i is loaded from a local (unshared) secondary memory storage to avoid possible conflicts.

A possible stencil for stage W_i is illustrated in Fig. [7]. In our work we made use of the persistent comunication requests [4] to avoid overheads caused by the repeated initialization of the comunication system.

Speedups obtained for this approach are plotted in Fig. [8]. A comparison with the speedups calculated for the Task Farm model (see Fig. [6]) shows that the Pipeline model is less efficient though the size of the matrices does not affect the speedup obtained. The efficiency obtained for the Pipeline solution is constant (percentual efficiency is 46% irrespective of the number of processes used and of the size of the matrices employed) implying that the Pipeline model

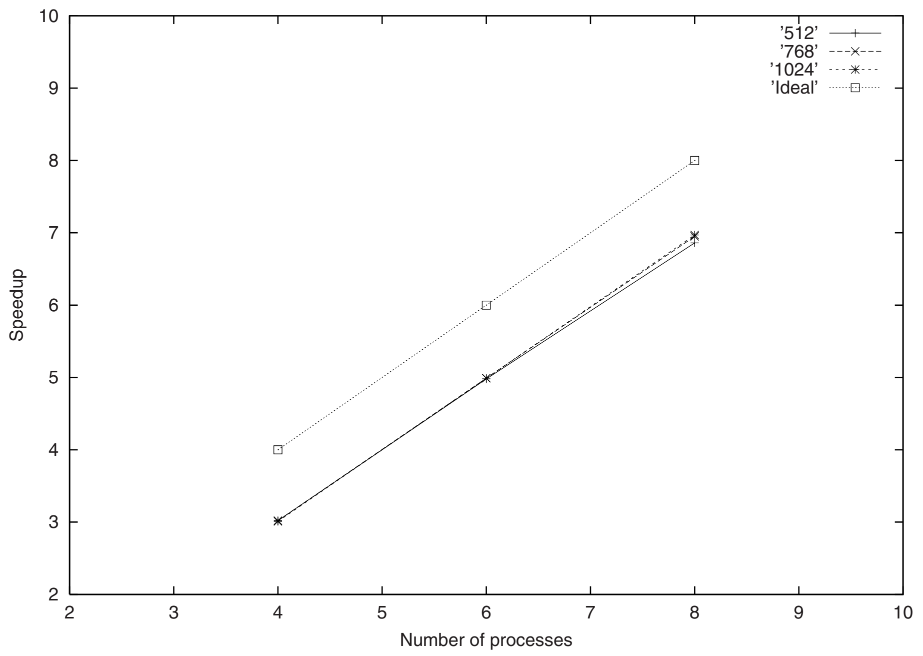

Fig. 6. Speedups obtained for the *ITF* model.

is well balanced. This suggests that a possible evolution of this model is a Pipeline made of Task Farms. Work on such a hybrid model is in progress.

5 Conclusion

Discrete variable approaches to wavepacket propagation techniques require to compute the following recursive matrix operation

$$G = A \cdot C + C \cdot B^T + V \odot C$$

where the value of the elements of C depend on the previous value of the elements of G.

This calculation engages a significant amount of computing resources. In this paper we have discussed how the request of computing resources can be reduced. To this end we have proposed a Task Farm model. Our study shows that a Task Farm model is appropriate to this end and that further reduction in both computing time and memory occupation can be obtained by approaches based on models accounting for the memory hierarchy of MIMD concurrent architectures.

```
$\cdots$ calculation of $Row(j,G)$ $\cdots$
$w_i = 0_n$
Read $(\{A(j,k) | k \in D_i\}$ from $Dataset(W_i, A))$
ForEach $k \in D_i$
    Read $(Row(k,C)$ from $Dataset(W_i, C))$
    $w_i = w_i + A(j,k) \cdot Row(k,C)$
    Read $(Row(k,B)$ from $Dataset(W_i, B))$
    $w_i(k) = w_i(k) + Row(j,C) \cdot Row(k,B) + V(j,k) \odot C(j,k)$
EndFor
If $(i == 1)$ then
    ReadFromFile $(Row(j+1,C))$
Else
    MPI_Recv (From $W_{i-1}$, $< Row(j+1,C), w_{i-1} >$)
    $w_i = w_i + w_{i-1}$
EndIf
If $(i \neq M)$ then
    MPI_Send $(W_{i+1}, < Row(j+1,C), w_i >)$
Else
    $\{Row(j,G) = w_i\}$
    WriteToFile $(Row(j,G))$
EndIf
$\cdots\cdots$
```

Fig. 7. Pseudocode of the i-th stage of the Pipeline model.

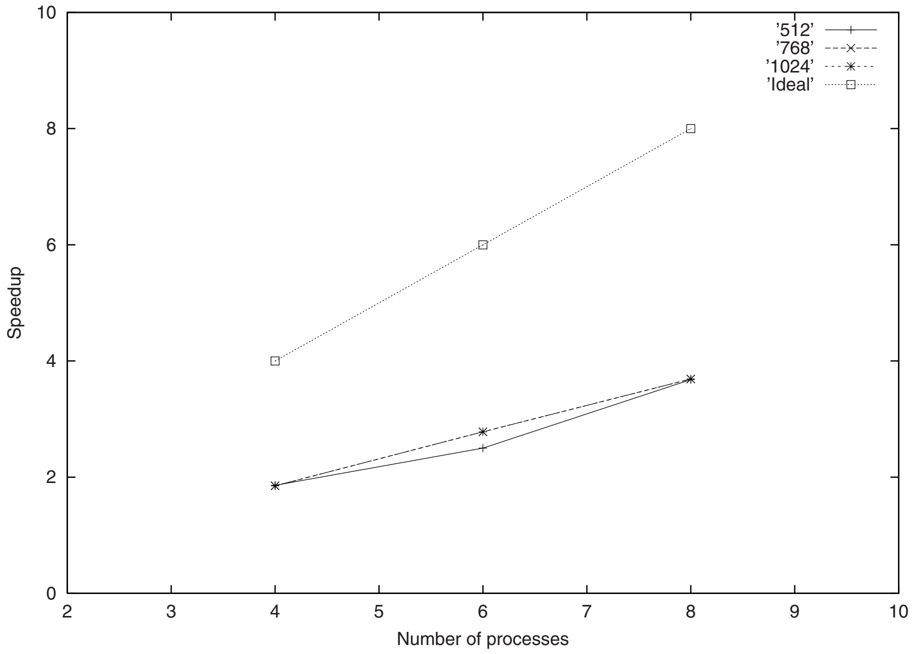

Fig. 8. Speedups obtained for the Pipeline model on the 8 node Beowulf plotted as a function of the number of processes.

This has led to an improved Task Farm model that adopts a block scheduling policy. Speedups reached by this solution are satisfactory and are not affected by an increase of the matrix size (in the size interval of our investigation). Further benefits could be ensured by the use of a *thread safe* communication library [5–7] that could reduce the need for collective communication.

We have also investigated the performances of a Pipeline model. These are lower than those of a Task Farm model though being characterized by a constant value of the efficiency with respect to the number of activated processes and to the size of the matrices. This suggests a possible use of a Hybrid Model in which each stage of the Pipeline is made of a Task Farm.

Acknowledgments.
This work has been financially supported by MIUR, ASI, CNR and COST in Chemistry (Action D23).

References

1. G. G. Balint-Kurti, Time dependent quantum approaches to chemical reactions, Lectures Notes in Chemistry, **75** (2000) 74 - 88.
2. G. D'Agosto, Parallel approaches to integrate the Schrödinger equation using a time dependent technique, Diploma thesis, (2000) Perugia.
3. D. Bellucci, S. Tasso, A. Laganà, Fine grain parallelism for discrete variable approaches to wavepacket calculations, Lecture Notes in Computer Science, **2331** (2002) 918 - 925.
4. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, MPI: The Complete Reference, MIT Press, Cambridge Massachusetts, (1996).
5. A. Skjellum, B. Protopov, S. Hebert, A thread taxonomy for MPI, Proc. of MPIDC, (1996).
6. A. Chowdappa, A. Skjellum, N. Doss, Thread-safe message passing with P4 and MPI, Tech. Rep. TR-CS-941025, Computer Science Department and NSF Engineering Research Center, Mississipi State University, (1994).
7. M. Danelutto, C Pucci, A compact thread-safe communication library for efficient cluster computing, Lecture Notes in Computer Science, **1823** (2000) 407 - 416.