CHAPTER 3

Vectorization of Fast Vortex Methods

Vortex methods, have attracted the interest of many researchers for the study of unsteady incompressible flows at relatively high Reynolds numbers. In such flows the vortical structures, which are the key to describing the whole flow field, tend to be confined in limited regions of the domain and vortex methods inherently adapt to resolve those regions. In this method the vorticity field is described by a set of N particles. The velocity field is computed then at each of the particles, resulting in a set of ODE's that needs to be integrated in time to determine their trajectories. This may alternatively be considered as an N-body problem for the particles. The simplest method for computing the velocities on the particles requires work that is proportional to N^2 as all pairwise interactions need to be computed. This renders the method prohibitive if one wants to use large numbers of particles to resolve all spatial scales in the flow.

The N-body problem appears in a diverse number of scientific fields (astrophysics, plasma physics, etc.) and techniques for the reduction of the computational cost have been addressed in many different perspectives. Solution methods to this problem may be classified in two broad categories: The *hybrid methods* (particle-mesh) and *direct* (particle only) methods. An extensive survey on *hybrid methods* may be found in the book of Hockney and Eastwood (1981). The cost of the hybrid methods is of $\mathcal{O}(\mathrm{N} + \mathrm{M}\log\mathrm{M})$ (where M is the number of mesh points), but their performance degrades when the clustering of the particles is highly non-uniform and they tend to introduce numerical diffusion. The direct methods compute the approximate velocity field as induced by clusters of particles using a certain number of expansions for each cluster. This number is determined by the accuracy required in the solution of the problem and the type of the field that is approximated. A hierarchical (tree) data structure is associated with the particles and is implemented in order to determine when those expansions may be used instead of the exact pairwise interactions to preserve the accuracy of the solution. The tree is used to establish interaction lists for the particles.

Originally the *direct* technique was developed for the simulation of gravitational problems by Barnes and Hut (1986) and Appel (1985) developed a data structure to facilitate such computations. The application of direct schemes in vortex methods has been primarily exploited by Greengard and Rohklin (1987) and Van Dommelen and Rudensteiner (1989) .They may be referred to as the box-box *(BB)* and particle-box *(PB)* algorithms respectively. Their computational cost theoretically scales as N or $N\log N$, respectively, but their applicability and efficient use of available super-computer architectures (vector/parallel) is what really determines the cost of these methods. The issue that arises then in the development of these codes is the question of parallel or vector implementation ? Although this question is usually answered by the availability of resources it seems worth exploring the possibilities. The nature of the algorithm was originally thought to be such that it would lend itself easier to a parallel implementation (Barnes and Hut, 1986) the main constraint being the recursive descent of the data structure. A naive implementation of this algorithm on a vector computer would require that this recursive scheme is unrolled into a sequence of iterative procedures. Such an implementation however would result in inefficient vectorization and no advantage would be taken of the pipelining capabilities.

The parallel implementation of the methods has been addressed at various degrees of sophistication and for a diverse number of problems (see for example Pépin (1990), Katzenelson (1989) and Salmon (1991)). Vectorization of the method has also been successfully addressed in works by Van Dommelen and Rudensteiner (1989) for vortex methods using the *PB* algorithm. Recently (1990) Hernquist, Makino, and Barnes published a series of papers demonstrating that the tree descent can indeed be vectorized. They applied their strategy to the Barnes - Hut algorithm obtaining orders of magnitude increase in the computational speed of the algorithm. Usually the most time consuming part of these schemes has been the building and descending of the tree in order to establish interaction lists. Once these have been established the velocity field of the particles may easily be determined. The vectorization of the *BB* algorithm for vortex methods is addressed in this work and its efficiency is compared to that of a *PB* scheme.

A data structure is implemented which lends itself to vectorization, in the construction and descent level, combining some of the ideas presented in the works of Barnes, Hernquist and Makino. It is applied to both (*PB* and *BB*) algorithms and comparisons are made as to what efficiencies may be obtained. The timings compare favorably with those reported in the past (Van Dommelen and Rudensteine,

1989) and it is demonstrated that an efficient implementation of the *BB* scheme can outperform the *PB* one for numbers of particles more than a few thousand.

3.1 Multipole Methods for the Velocity Evaluation

In the context of vortex methods the vorticity field is discretized by a set of elementary vortices whose individual field is expressed by a cut-off function $\phi(x)$. The superposition of these fields determines the vorticity ω, at any location $\mathbf{x}$, as :

$$\omega(\mathbf{x}) = \sum_{n=1}^{N} \Gamma_n \phi(\mathbf{x} - \mathbf{x}_n) \tag{3.1}$$

If $\phi(\mathbf{x})$ is a δ-function or we are interested in the velocity field (u, v) in a location (x, y) away from the particles, the velocity field is expressed in complex form as:

$$\boxed{V(Z) = \frac{i}{2\pi} \sum_{n=1}^{N} \frac{\Gamma_n}{Z - Z_n}} \tag{3.2A}$$

where $Z = x + iy$ and $V = u - iv$. In order to compute the velocity at each one of the particles the above sum implies $\mathcal{O}(N^2)$ operations. To reduce this computational cost the geometrical distribution of the vortices is exploited. The key observation is that the velocity induced by a cluster of particles need not be computed directly from its individual members. Instead the velocity field induced by a group of M particles clustered around a centre Z_M may be approximated by a finite number (P) of multipole expansions. At distances greater than the radius R_M of this cluster this approximation converges geometrically. This is the basis for the *PB* scheme. The *BB* scheme introduces one more step. The expansions of a certain cluster may be translated and computed with the desired accuracy at the center of another cluster. Subsequently those expansions are used to determine the velocity of the particles in the second cluster. The derivation of those expansions is based on the following two identities of complex numbers :

$$\frac{1}{1-z} = \sum_{n=0}^{\infty} z^n; \qquad \text{for } |z| \leq 1 \tag{$*$}$$

$$\sum_{k=0}^{P} \alpha_k (Z - Z_0)^k = \sum_{l=0}^{P} \Big(\sum_{k=l}^{P} \alpha_k \binom{k}{l} (-Z_0)^{k-l} \Big) Z^l. \tag{$**$}$$

Adding and subtracting Z_M on the denominator of Eq. 3.2A the velocity field induced by the cluster may be expressed as :

$$V(Z) = \frac{i}{2\pi}\frac{1}{Z - Z_M}\sum_{m=1}^{M}\frac{\Gamma_m}{1 + (Z_M - Z_m)/(Z - Z_M)}$$

By expanding now the denominator inside the sum using $(*)$ and keeping (P) terms in the expansions the velocity is determined by :

$$\boxed{V(Z) = \frac{i}{2\pi}\frac{1}{Z - Z_M}\sum_{k=0}^{P}\frac{\alpha_k}{(Z - Z_M)^k}} \qquad (3.2B)$$

The complex coefficients α_k express the moments of the discrete vorticity distribution in the cell and are computed by :

$$\boxed{\begin{gathered}\alpha_k = \sum_{m=1}^{M}\Gamma_m(Z_m - Z_M)^k \\ k = 0, \ldots, P\end{gathered}} \qquad (3.2C)$$

To make the computations more efficient the coefficients of boxes that belong to coarser levels of the hierarchy are not constructed directly from the particles. Instead they are obtained by a shifting of the expansions of their descendants. To obtain then the expansions of a parent box from those of its children we use identity $(**)$ to get :

$$\boxed{\alpha_l^{\text{parent}} = \sum_{k=0}^{l}\binom{l}{k}\alpha_k^{\text{children}}(Z_M^{\text{children}} - Z_M^{\text{parent}})^{l-k}} \qquad (3.2D)$$

We may observe now from Eq. 3.2B, that at a distance R from the center of the cluster the rate of convergence of the expansions would be proportional to $(R_M/R)^{P+1}$ and, for example, if $R \geq 2R_M$ this rate is proportional to $(1/2)^{P+1}$, implying the geometric convergence of the series.

The above expansions are the main tools for the *PB* algorithm. However one may proceed one step further and recast the Laurent series into Taylor series and consider the interactions between groups of particles. These interactions take place in the form of shifting the center of expansions of one cluster (M) to the center of another (G). Those expansions are then used to compute the velocity of the particles in each box. This results in eliminating the logN factor from the work count of the scheme.

To obtain these expansions we add and subtract Z_G in the denominator of the expression Eq. 3.2A so we have that the velocity induced by the group M at a point Z in the neighborhood of Z_G is equal to :

$$V(Z) = \frac{i}{2\pi} \sum_{k=0}^{P} \frac{\alpha_k}{(Z_G - Z_M)^{k+1}(1+\xi)^{k+1}}$$

where $\xi = (Z - Z_G)/(Z_G - Z_M)$. Expanding this expression for ξ using $(*)$, the velocity field induced by particles in the cluster M, at a point Z in the neighborhood of Z_G is:

$$\boxed{V(Z) = \frac{i}{2\pi} \sum_{l=1}^{P} \delta_l (Z - Z_G)^l} \qquad (3.2E)$$

where the coefficients δ_l are determined by :

$$\boxed{\begin{gathered}\delta_l = \frac{(-1)^{l+1}}{(Z_G - Z_M)^l} \sum_{k=0}^{P} \binom{l+k-1}{k} \frac{\alpha_k}{(Z_G - Z_M)^k} \\ l = 1, \ldots, P\end{gathered}} \qquad (3.2F)$$

Interactions are computed at the coarsest possible level while satisfying the accuracy criterion. However the velocities of the individual particles are computed from the expansions of the finest boxes in the hierarchy. Hence the expansions of the coarser level boxes have to be transferred down to their descendants as follows :

$$\boxed{\delta_l^{\text{children}} = \sum_{k=l}^{P} \binom{k-1}{k-l} \delta_k^{\text{parent}} (Z_M^{\text{children}} - Z_M^{\text{parent}})^{k-l}} \qquad (3.2G)$$

This shifting uses (**) and, as in Eq. 3.2C, does not introduce any additional errors. One may observe here the different use of the coefficients α_k and δ_l when computing velocities on the particles. The coefficients δ_l of a certain box are used to compute the velocity on the particles of the same (childless) box whereas the α_k's are used to compute the velocities due to particles that belong to well separated boxes.

Proofs for the above results may be found in a more rigorous and complete form in Greengard and Rohklin (1987). They show that the series converges if the distance between interacting boxes and/or particles is at least twice as large as the radius of the cluster involved. By using the expansions when clusters are separated by larger distances one may reduce the number of expansions that are necessary to obtain the same level of accuracy. This trade-off may be optimized by linking the number of expansions employed to the distance between the interacting pairs dynamically (Salmon, 1991). However this may result in increased cost for the construction of the interaction lists and it would complicate the algorithm and reduce its vectorizability.

3.2 The Data Structure

The two-dimensional space is considered to be a square enclosing all computational elements. We apply the operation of continuously subdividing a square into four identical squares until each square has only a certain maximum number of particles in it (see Fig.3.1 for an example) or the maximum allowable level of subdivisions has been reached (the latter requirement seems obligatory when one programs in FORTRAN and has to predefine array dimensions). This procedure for a roughly uniform particle distribution results in $\mathcal{O}(\log_4 \mathrm{N})$ levels of squares.

The two fast algorithms under discussion, *PB* and *BB*, exploit the topology of the computational domain each with a different degree of complexity and efficiency. The hierarchy of boxes defines a tree data structure which is common for both algorithms. However its key ingredients and address arrays are implemented in a different way as explained below for the two algorithms. The tree construction proceeds level by level starting at the finest level of the particles and proceeding upwards to coarser box levels.

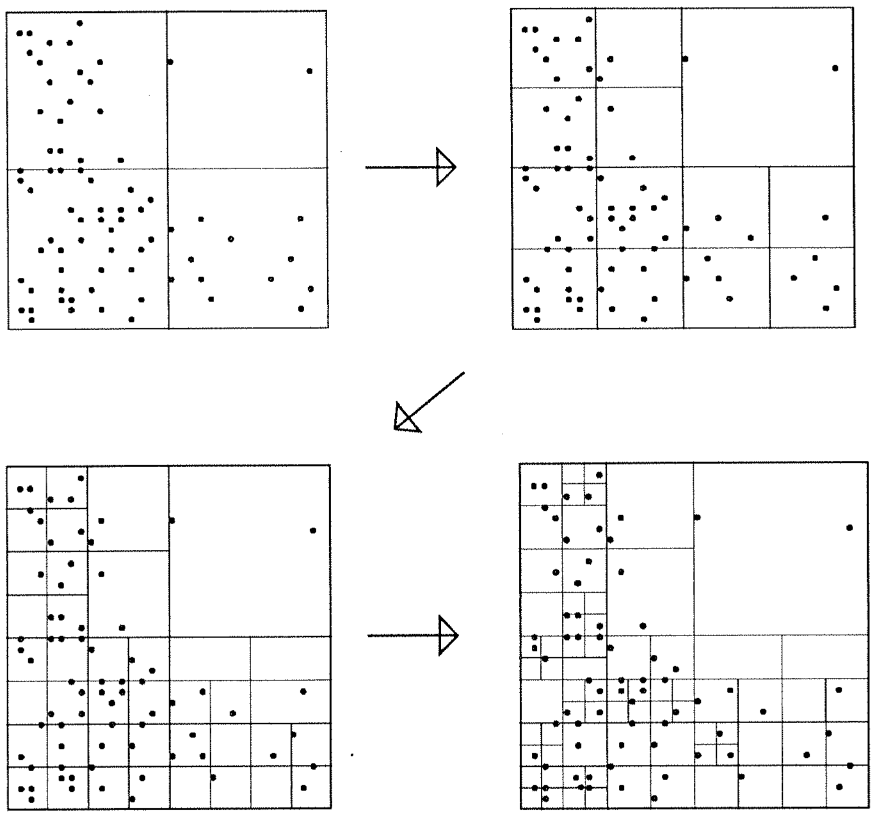

FIG. 3.1 An example showing four stages of subdividing the computational domain. In the final stage each box contains less than three particles.

Due to the simplicity of the geometry of the computational domain the addressing of the elements of the data structure is facilitated significantly. As the construction proceeds pointers are assigned to the boxes so that there is direct addressing of the first and last particle index in them as well as there is direct access to their children and parents. This facilitates the computation of the expansion coefficients of the children from the expansions of the parents for the *BB* algorithm and the expansions of the parents from those of the children for the *PB* algorithm.

3.2.1 Parameters of the Data Structure

The data structure is used to determine when the expansions are to be used and when pairwise interactions have to be calculated. Usually this data structure is referred to as the '(family) tree ' of the particles. It helps in communicating to the computer the geometric distribution of the particles in the computational domain. The particles reside at the finest level of the structure. Clusters of particles form the interior nodes of the tree and hierarchical relations are established. The data structure adds to the otherwise minimal memory requirements of the vortex method. This extra memory however is the tradeoff for the speed of the fast algorithms. One may add several features to this data structure trying to relate to the computer architecture as much information as possible for the particle configuration. However these memory requirements have to maintain a certain degree of uniformity for all the levels of the tree. So if one wants to use large numbers of particles (hence several levels of subdivisions) these memory requirements should be kept to a reasonable level.

The formation and descent of the tree add to the cost of the algorithms and a non-refined implementation may result in the degradation of the whole algorithm. A data structure may be Lagrangian or Eulerian. The former adapts automatically to the locations of the particles and can be the same for several steps. The latter has to be reconstructed at every step as the particles change positions in the domain. There are several ways that nearby particles could be clustered together and some of the decisions to be made are :

The center of expansions. This can be either the geometric center of a certain region in space or the center of mass of the distribution of particles or some other location chosen so that the data structure is conveniently addressed and the expansions converge rapidly. Choosing the center of 'mass' implies that for a uniform particle distribution and a certain number of particles per cluster the minimum radius for convergence is expected. On the other hand the geometric center (as in the present work) facilitates tremendously the addressing of the boxes in the data structure.

The cluster size. Accuracy requirements impose that the cluster size should be as small as possible while efficiency considerations dictate that clusters should contain enough particles so that the use of the expansions is beneficial. There are different approaches to that as others require the finest clusters to contain fewer than

L_{min} (e.g. for the Barnes-Hut algorithm $L_{min} = 2$) particles where others impose the finest level of subdivision beyond which no further subdivisions take place. The former procedure seems to offer more accurate expansions whereas the latter economizes in memory and compactness of the data structure especially when more than one particle resides at the finer boxes and large numbers of terms are to be kept in the expansions. In the present algorithms we follow a hybrid strategy as we keep at least $\mathrm{L_{min}}$ particles per box until we reach a predetermined finest level of boxes. The number $\mathrm{L_{min}}$ may be chosen by the user depending on the particle population and configuration so as to achieve an optimal computational cost.

A related issue is how one subdivides a cluster or box with more than L_{min} particles. Practical implementations of the method resort basically to two techniques. One may split the parent cluster into two children resulting in a so called binary tree. The direction of this dissection depends on the distribution of the particles, attempting to optimally adapt the data structure to the locations of the particles in the computational domain. In practice this has the benefit of requiring fewer terms in the expansions to obtain a certain accuracy (for a relatively uniform particle distribution). Alternatively, as in the present scheme, one may simply divide the box into four boxes. This would require a larger number of expansions to be calculated and stored to obtain the required accuracy, as the radius for convergence is not minimized. However these 'extra' terms have a minimal effect to the overall cost of the scheme as they appear in fully vectorized parts of the algorithm. Moreover the regularity of such a procedure facilitates the logic of the algorithm and the construction and addressing of the data structure.

Addressing the clusters. A key factor in the computer implementation of the method, addressing should be such that it does not inhibit the vectorization or the parallel implementation of the method. Traversing and building the hierarchical tree is highly dependent on this procedure. A simpler technique would be to store information (such as geometric location, size, family ties etc.) for the tree nodes in the memory. Such a procedure may severely limit the number of particles that can be computed (Pépin,1990) if excess information is stored. Besides it would degrade the performance for machines (such as the CRAY-2) where memory access is computationaly intensive. However both restrictions do not seem to consist major drawbacks in the present implementation on the CRAY YMP, provided a reasonable number of subdivisions (less than 10) is allowed.

Another key issue is the addressing of the particles. As particles are usually associated with a certain box it is efficient to sort the particle locations in the memory so that *particles that belong to the same box occupy adjacent locations in the memory devoted to the particle arrays.* Such memory allocation enhances the vectorization tremendously as very often we loop over particles of the same box (e.g., to construct the expansions at the finest level, or to compute interactions) and the loops have an optimum stride of one.

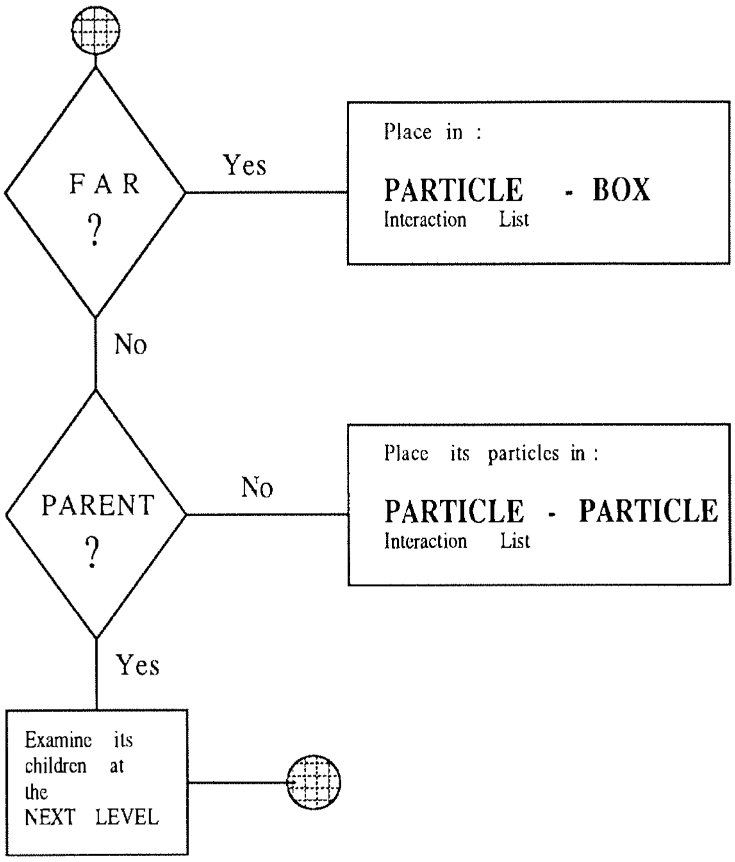

FIG. 3.2 Flow Chart for the tree traversal of the *PB* algorithm for *all particles of a childless box.*

As one can deduce, the possible combinations of the above features may result in a number of different implementations. Depending on the implementation a different degree of vectorization may then be achieved.

3.3 Description of the Algorithms

In both algorithms, described herein, we may distinguish three stages:

- Building the data structure (tree)
- Establishing the interaction lists (by non-recursively descending the tree)
- Velocity evaluations for all particles in the domain.

The building of the data structure is common for both algorithms but they differ in the tree descent and the velocity evaluation. Basic requirements for an efficiently vectorized code are the simplicity of the algorithm, the existence of long vectorizable loops (i.e., simple loops, odd stride e.t.c) and the reduction of memory referencing. Care has been exercised at all stages so that the maximum degree of vectorization is achieved and efficient calculations result. In the present implementation the building of the data structure consumes about $5-7\%$ of the time whereas the descent consumes another $5-10\%$ so that the largest amount is spent in computing the velocities.

3.3.1 The Particle-Box Algorithm

The hierarchical structure of the algorithm has a logical complexity that implies a recursive procedure. The algorithm may be easily described by the following code :

```
subroutine interact (n,C)
    IF (particle n is well separated from cell C)
        velocity = sum of expansions of cell C on n
    ELSE
        CALL interact (n,children of C)
    ENDIF
return
```

Note the recursiveness of this subroutine in the ELSE block. A straightforward approach to the non-recursive programming of this subroutine, in an attempt to vectorize it, would be to unroll it. Such a procedure would introduce a depth-first search of the tree. However this is not very efficient because for most applications the depth of the tree is not long enough to enable optimized vector operations. As reported by Makino (1990), Barnes introduced interaction lists associated with each particle and then vectorized by looping over the particles. Every particle had its own interaction list but increased memory referencing and the additional complexity of the algorithm resulted in degradation of the performance. In our algorithm we employ the following alternate procedure:

Step 1 : Building the data structure (tree)

Step 1a : For each of the squares at each level that are not further subdivided we compute the p-terms of the multipole expansions. These expansions are used to describe the influence of the particles at locations that are well separated from their cluster. The *cost* of this step is $\mathcal{O}(Np)$.

Step 1b : The expansions of all parent boxes are constructed by shifting the expansion coefficients of their children. The tree is traversed upwards in this stage. Rather than constructing the expansions of all the members of a family (that is traverse each branch until the root is reached) we *construct the expansions of all parent boxes at each level simultaneously.* This enables long loops over the parent boxes at each level. Care is taken so that the procedure is fully vectorized by taking advantage of the regularity of the data structure and the addressing of the boxes in the memory. Moreover the regularity of the data structure allows us to precompute many coefficients that are necessary for the expansions. The cost of this step is $\mathcal{O}(Np^2)$, as each shifting requires p^2 operations and there are at most $(4N-1)/3$ boxes in the computational domain.

Step 2 : Establishing of interaction lists

In the present algorithm a **breadth-first** search is performed at each level to establish the interaction lists of each particle (cell). This search is facilitated by the regularity of the data structure and the identification arrays of the cells in the tree. At each level interaction lists are established for the particles (cells) by looping across the cells of a certain level.

Algorithmically this operation is represented as follows:

```
subroutine setlist (n)
Do 1 l = 1, levelmax
    CALL FARCLOSE(Z(n),ds(l),IZ(l),kxm,Lstxm,kfr,Lstfr,kcls,Lstcls)
    CALL CLOSECHECK(kcls,Lstcls,kpp,Lstpp,kxm,Lstxm)
    CALL GATHER(kfr,Lstfr,kpp,Lstpp,ZP,GP,ZB,EB)
1 Continue
return
```

For each particle **n** the hierarchy of cells is traversed breadth first. The check for faraway or close boxes is performed within the subroutine FARCLOSE that is fully vectorized. Initially the addresses of `kxm` cells that belong to the coarsest level are stored in array `Lstxm`. By checking the locations of the boxes at the grid of level `l` and the respective location of the particle on that grid the cells are identified as either faraway (stored in `Lstfr`) or nearby (stored in `Lstcls`). Boxes that are far interact with the particle and are placed in a P-B interaction list `Lstpp` along with their expansions `EB` and locations `ZB` in subroutine GATHER. Boxes that are close however are further examined in the fully vectorized subroutine CLOSECHECK. Those that are childless have their particles stored in an array `Lstpp` so that they interact directly. Those that are parents have their children stored in array `Lstxm` so that they are fed back in FARCLOSE at the finer level. Note that at the finest level all boxes are considered childless and subroutine CLOSECHECK need not be called. This way the interaction lists for the particles are formed successively and a fully vectorized force calculation calculates the velocities of the particles at the following stage.

Note now that this depth first search for interaction lists is further facilitated by the following observation. Every particle belongs to a childless box. It is easy then to observe that *all particles in the same box share the same interaction list comprised of members of the tree that belong to coarser levels.* This way the tree is traversed upwards for all particles in a childless box together and downwards separately for each particle. It is evident that this procedure is more efficient for uniformly clustered configurations of particles as there would be more particles that belong to childless boxes at the finest level.

The *cost* of this step scales as $\mathcal{O}((N/L_{\min})\log N)$ as there are $\log N$ levels of boxes and the tree is traversed ($N/L_{\min}$) times.

Step 3 : Computation of the interactions.

Once the interaction lists have been established the velocities of the particles are computed by looping over the elements of the lists. For particles that have the same boxes in their interaction list this is performed simultaneously so that memory referencing is minimized. Moreover by systematically traversing the tree the particle-particle interactions are made symmetric so that the cost of this computation is halfed. Care has been exercised to compute the velocities with the minimum possible number of operations (Section 3.4).

The *cost* of this step is $\mathcal{O}(Np)$.

3.3.2 The Box-Box Algorithm.

This scheme is similar to the *PB* scheme except that here every node of the tree assumes the role of a particle. In other words interactions are not limited to particle-particle and particle-box but interactions between boxes are considered as well. Those interactions are in the form of shifting the expansion coefficients of one box into another and the interaction lists are established with respect to the locations of every node of the tree.

The scheme distinguishes five categories of interacting elements of the tree with respect to a cell denoted by **c**.

- **List 1 :** All childless boxes neighboring **c**.
- **List 2 :** Children of colleagues of b's parents that are well separated from **c**. All such boxes belong to the same level with **c**.
- **List 3 :** Descendants (not only children) of **c's** colleagues (boxes of the same size as **c**), whose parents are adjacent to **c** but are not adjacent to **c** themselves. All such boxes belong to finer levels.
- **List 4 :** All boxes such that box **c** belongs to *their* List 3. All such boxes are

childless and belong to coarser levels.

- **List 5 :** All boxes well separated from **c's** parents. Boxes in this category do not interact directly with the cell **c**.

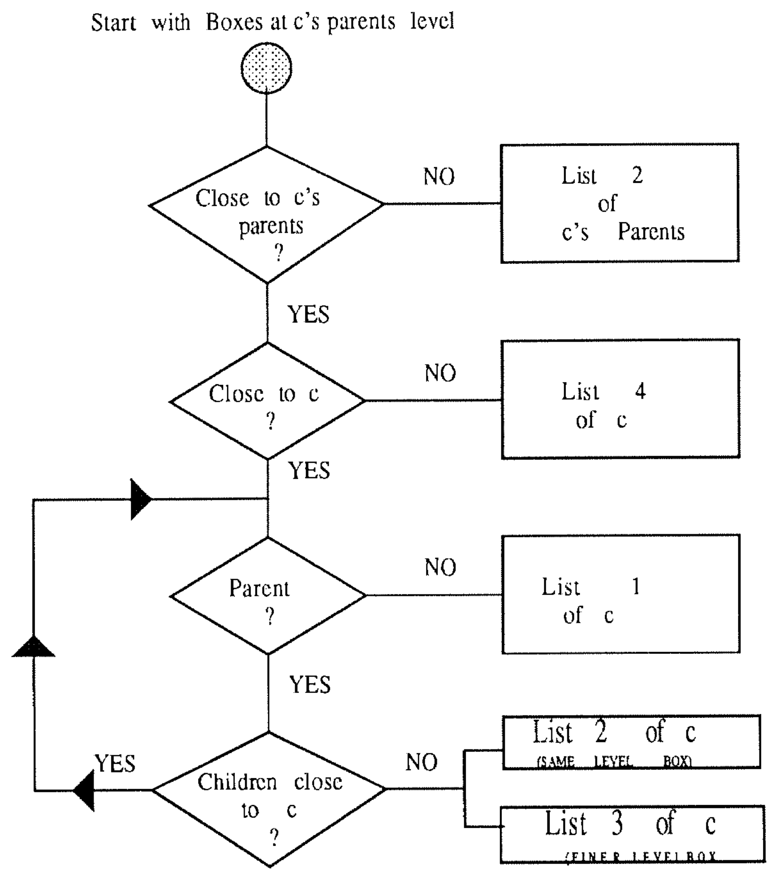

FIG. 3.22 Flow Chart for the Tree Traversal of the *BB* scheme for a *cell* **c**

If the cell **c** is childless it may have interacting pairs that belong to all four lists. However if it is a parent it is associated with boxes that belong to lists 2 and 4 as described above. These observations are directly applied in our algorithm and we may distinguish again the following 4 steps.

Step 1 : Building the data structure.

This procedure is the same as for the *PB* scheme. This fact enables us actually to compare directly the two algorithms and asses their efficiency.

Step 2 : Construction of Interaction Lists.

To establish the interaction lists we proceed again level by level starting at the coarsest level. For each level we distinguish childless and parent boxes. In establishing lists 1 and 3 we need only loop over childless boxes whereas to establish lists 2 and 4 we loop over all cells that are active in a certain level.

Step 2a : Here we establish lists 1 and 3. We start at the level of the parents of box **c** and we proceed level by level examining again breadth first, until we reach the finest level of the structure (the particles). The elements of lists 1 are basically the particles and account for the particle-particle interactions. Care is exercised so that this computation is symmetric and we need to traverse the tree downwards only. The elements of List 3 are the boxes and are accountable for the particle-box interactions in this scheme.

More specifically an algorithmic description of our algorithm is given by :

```
subroutine Lists1&3(lpr,kchlds)
    Do 1 k = 1,kchlds
        Find the colleagues of k's parents (Lclg)
        Place the children of (Lclg) boxes in (Lstxm)
        Do 2 l = lpr,lmax
            CALL FARCLOSE(ZC,ds(l),Kxm,Lstxm,k3,Lst3,kcls,Lstcls)
            CALL CLOSECHECK(kcls,Lstcls,k1,Lst1,kxm,Lstxm)
        2 Continue
    1 Continue
return
```

The outer loop here is over all childless boxes. The colleagues of their parents are identified (either by being retrieved directly from an array or by using FARCLOSE) and subsequently their children are placed in `Lstxm`. Subsequently a call to FARCLOSE distinguishes between faraway and nearby boxes at different levels. Boxes placed in `Lst1` are to interact directly whereas those placed in `Lst3` are to interact as particle-box.

Step 2b : Here we establish interaction lists 2 and 4 for all boxes in the hierarchy. We start at the coarsest possible level and proceed downward until reaching the level of box **c** to establish the interaction lists. To do so for a certain box we start by examining boxes that are not well separated from its parents (otherwise they would have been dealt with at the coarser level). Subsequently the children of those boxes are examined to establish interaction lists. Algorithmically this operation is represented by:

```
subroutine Lists2&4(lpr,kbox)
    Do 1 k = 1,kbox
        Do 2 l = 10,lpr
            Boxes close to parents (?)(use FARCLOSE)
            Close to parents - Childless(?)(use CLOSECHECK)
            Close to parents and Childless - Close to box (?)(Use FARCLOSE)
        2 Continue
    1 Continue
return
```

Step 3 : Computations of the interactions

In this scheme we consider three kinds of interactions: the box-box, particle-box and particle-particle interactions. The latter two categories were discussed in the previous section. For the box-box interactions once the respective interaction lists have been established (with members of lists 2 and 4) we need to transfer those expansions down to the ones of the children and add them to the existing ones. This procedure is vectorized by looping over the number of boxes at each level. The use of pointers to access the children of each box enhances this vectorization. Note that an arbitrarily high number of expansions can be calculated efficiently by unrolling the loop over the number of expansions into the previously mentioned loop.

3.4 Practical Formulas for Velocity Calculations

Once the interaction lists have been established the velocities on the particle locations need to be calculated using formulas 3.2(A-F). Because this is the most time consuming part of the algorithm an effort is made to reduce as much as possible the computational cost. A possible increase in the computational speed may be obtained by some ASSEMBLY programming for the velocity evaluations. Following are some of the formulas employed to reduce this computational cost.

▷ *Particle-Particle Interactions :* In order to achieve higher convergence rates in the vortex methods more complicated functions than the delta functions are usually necessary for the description of the vorticity field. This will reduce the efficiency of the algorithm but to minimize the extra cost one could make use of look up tables. In these look up tables in order to get the accuracy required we should not only store the values of the function but its derivatives as well. Using the MATH77 scientific libraries of the CRAY would help reduce this computational cost.

▷ *Particle-Box Interactions :* Formula 3.2B may be calculated using recursive relations. By computing separately real and imaginary parts for the velocities we have that for a particle located at $z = x + i\,y$ the contribution from a box located at $X_M = X_M + i\,Y_M$ having expansions $\alpha_k = \lambda_k + i\,\mu_k$ may be computed with the following algorithm.

$$r_0 = \frac{x - X_M}{(x - X_M)^2 + (y - Y_M)^2}\,, \qquad f_0 = \frac{y - Y_M}{(x - X_M)^2 + (y - Y_M)^2}\,,$$

$$\begin{aligned} r_k &= r_{k-1} \cdot r_0 - f_{k-1} \cdot f_0 \\ f_k &= r_{k-1} \cdot f_0 + f_{k-1} \cdot r_0 \end{aligned}$$

and the velocities (u,v) would be given as :

$$\begin{aligned} u &= -\sum_{k=0}^{k=P} \lambda_k \cdot r_k - \mu_k \cdot f_k \\ v &= \sum_{k=0}^{k=P} \lambda_k \cdot f_k + \mu_k \cdot r_k \end{aligned}$$

▷ *Parent Cell Expansions :* The expansions of parent cells are computed from the expansions of their children. This procedure does not introduce any errors and it helps in economizing computer time compared to a direct calculation using the particle locations. The amount of necessary calculations is reduced by exploring the regularity of the location of the children boxes with respect to the parents. Thus in formula 3.2D we have

$$Z = Z_M^{\text{children}} - Z_M^{\text{parent}} = \frac{d}{2}\begin{cases} -1 + i, & \text{for Box 1;} \\ -1 - i, & \text{for Box 2;} \\ 1 - i, & \text{for Box 3;} \\ 1 + i, & \text{for Box 4.} \end{cases}$$

where d is the size of the child cell. Hence we may calculate explicitly the powers $\binom{l}{k} Z^{l-k}$ for all values of l,k thus speeding up the computations inside the loop for the boxes.

▷ *Cell-Cell Interactions:* A procedure similar to the one for the particle-cell interactions is followed here. If $\delta_l = \eta_l + i\theta_l$ are the coefficients of a box located at $Z_G = X_G + iY_G$ which are contributed by a box located at $Z_M = X_M + iY_M$ then these coefficients are computed with the following scheme :

$$r_0 = \frac{X_G - X_M}{(X_G - X_M)^2 + (Y_G - Y_M)^2}, \qquad f_0 = \frac{Y_G - Y_M}{(X_G - X_M)^2 + (Y_G - Y_M)^2},$$

$$\begin{aligned} r_k &= r_{k-1} \cdot r_0 - f_{k-1} \cdot f_0 \\ f_k &= r_{k-1} \cdot f_0 + f_{k-1} \cdot r_0 \end{aligned}$$

and finally:

$$\begin{aligned} R_l &= -\sum_{k=0}^{k=P} \lambda_k \cdot r_k - \mu_k \cdot f_k \\ F_l &= \sum_{k=0}^{k=P} \lambda_k \cdot f_k + \mu_k \cdot r_k \\ \eta_l &= r_l \cdot R_l - f_l \cdot F_l \\ f_k &= r_l \cdot F_l + f_l \cdot R_l \end{aligned}$$

▷ *Children Expansions :* Cell-Cell interactions are performed at the coarsest possible level. Then the expansions of the parent boxes are transferred down to the expansions of their children and are added to the existing ones until the finest level at any branch of the tree is reached. Those expansion coefficients are computed using formula 3.2G in a similar way as for the expansions of parent cells discussed above.

▷ *Cell - Particle Interactions :* Once the expansions of all childless boxes have been obtained they are used to compute the velocities on the particles. Formula 3.2E is used and a procedure similar to the one discussed above is followed. Vectorization is achieved by looping over all particles in a certain childless box. This may be inefficient if only a small number of particles is contained in this box.

3.5 Performance of the Algorithms

Comparisons were made for the two fast algorithms and the $\mathcal{O}(N^2)$ in terms of efficiency and computational speed. We require that the fast algorithms produce the same results as the N^2 scheme with minimum accuracy of six significant figures in the velocity field of a random uniform distribution of particles in a square. We allow up to eight levels in the hierarchy of the boxes and use ten terms in the multipole expansions at all levels.

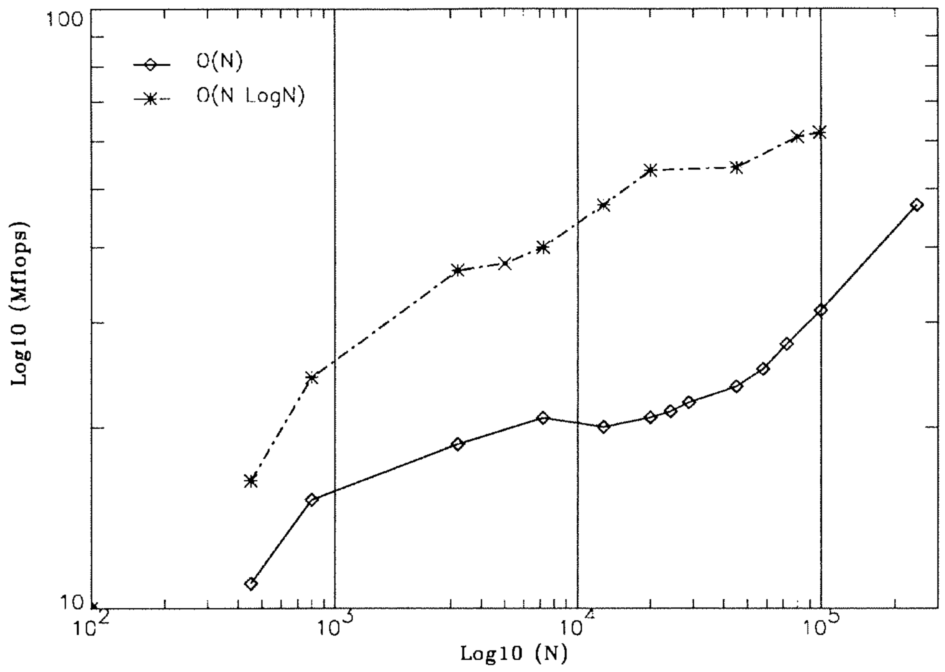

FIG. 3.3 Performance in Mflops of the *PB* and *BB* algorithms on the CRAY/XMP-18

In Fig. 3.3 we show the efficiency in the vectorization achieved for the fast algorithms on the CRAY/XMP-18. The *PB* algorithm is much more efficient than its *BB* competitor in regards with vectorization. This is a result that was expected because of the simplicity of the PP scheme versus the algorithmic complexity of the BB scheme.

This is not the whole story however, since in applications we are mainly interested in the overall speed of the algorithm and not necessarily in its Mflps efficiency. Fig. 3.4 shows the CPU time required on a single processor of an XMP-18 in each of the three methods for an evaluation of all N velocities for N elements with a minimun accuracy

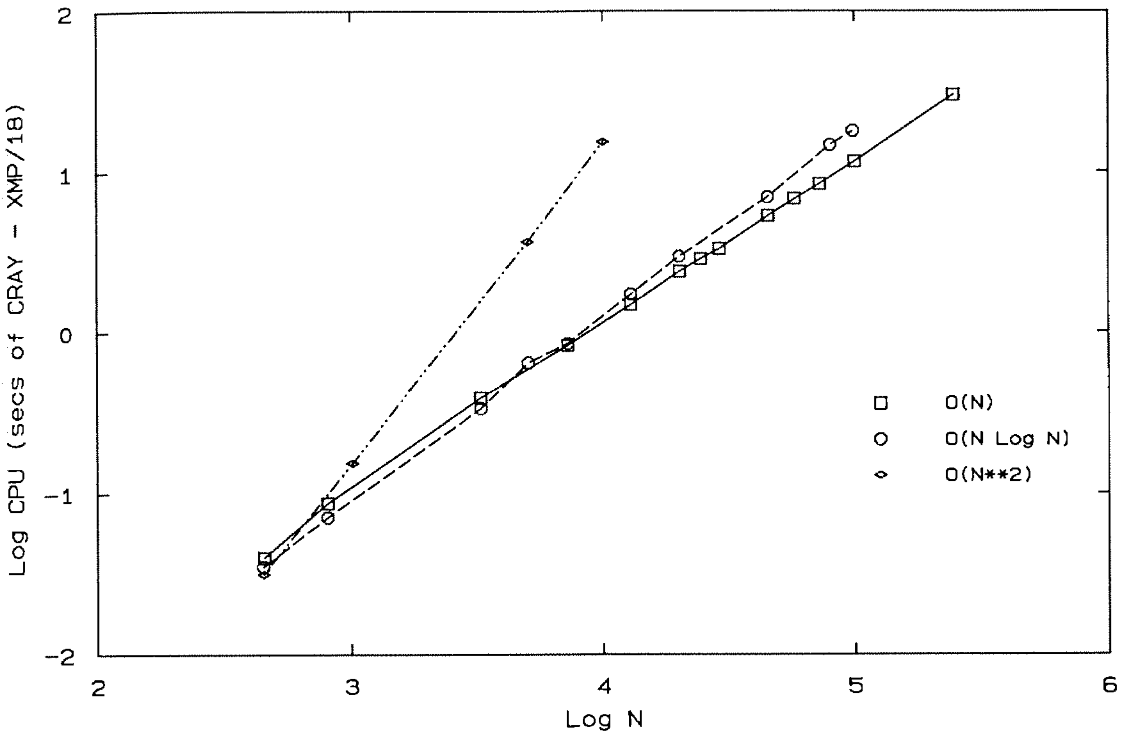

FIG. 3.4 Computational Cost of the PB, BB and direct summation algorithms

of 10^{-6}. Note that fast methods become advantageous for computations involving only $N \approx 500$ particles with the BB method breaking even with the PB method for $N \approx 4400$. Note that the CPU time required on a YMP processor is about 2/3 of the time required on an XMP. A timestep requiring one evaluation of all velocities for a million particles requires about one minute on single processor of a CRAY YMP while the N^2 algorithm would require roughly 24 hours.