Contour Dynamics Simulations with a Parallel Hierarchical-Element Method

R.M. Schoemaker[1,2], P.C.A. de Haas[2], H.J.H. Clercx[1], and R.M.M. Mattheij[2]

[1] Department of Applied Physics
[2] Department of Mathematics and Computing Science
Eindhoven University of Technology, The Netherlands
R.M.Schoemaker@tue.nl

Abstract. Many two-dimensional incompressible inviscid vortex flows can be simulated with high efficiency by means of the contour dynamics method. Several applications require the use of a hierarchical-element method (HEM), a modified version of the classical contour dynamics scheme by applying a fast multipole method, in order to accelerate the computations substantially. In this paper it is shown that the acceleration of contour dynamics simulations by means of the HEM can be increased further by parallelising the HEM algorithm. Speed-up, load balance and scalability are parallel performance features which are studied for several test examples. Furthermore, typical simulations are shown, including an application of vortex dynamics near the pole of a rotating sphere. The HEM has been parallelised using OpenMP and tested with up to 16 processors on an Origin 3800 cc-NUMA computer.

1 Introduction

Large-scale vortices are coherent structures that can be found in the oceans, the atmosphere of our planet as well as in the atmosphere of other planets. Examples of terrestrial nature are high and low-pressure areas, eddies in the ocean and the large polar vortex. Other planetary examples are the Great Red Spot on Jupiter, the Great Dark Spot on Neptune and huge rotating dust structures on Mars. The thickness of the layer of fluid these structures evolve in (on Earth: 1 – 10 km), is small compared to the horizontal size of the coherent structure itself (on Earth: 100 – 1000 km). This geometrical confinement, together with the planetary rotation and the density stratification in the fluid layer, implies quasi two-dimensionality of the flow. The motion of such large-scale structures is slow compared to the rotation speed of the planetary body implying that these structures are nearly non-divergent. Their dynamics can in good approximation be described by the two-dimensional incompressible inviscid variant of the Navier-Stokes equations, viz. the 2D Euler equations.

A suitable and elegant numerical technique for simulating two-dimensional (2D) vortex flows is the contour dynamics method [2] [6]. The collection of 2D vortices is approximated by nested patches of uniform vorticity. Only the evolution of the contour edges needs to be computed for determining the complete

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2329, pp. 1010–1019, 2002.

dynamics of the vortices and this makes the method efficient because no grid is needed. However, when highly complicated flow patterns emerge during a simulation, the conventional numerical scheme becomes inefficient in its time-complexity. Another class of problems concerns the evolution of vortices in the presence of non-uniform background vorticity. The non-uniform background vorticity is usually a local approximation of the latitudinal variation of the Coriolis force. For example, when simulating the dynamics of vortices, located near the poles of a rotating sphere (the local approximation is denoted as the γ-plane), the initial vorticity distribution is rather intricate due to the necessity to discretise, together with the vorticity patches, the background vorticity field[1]. An acceleration of the method is in this case already appropriate from the start of the simulation. The hierarchical-element method for contour dynamics (HEM) [4] solves for the limited applicability of the conventional scheme.

The algorithmic structure of the HEM has certain features that makes parallelisation a good means for speeding-up contour dynamics simulations to an even greater extent. The HEM has been parallelised using OpenMP, which is a new industry standard for parallel programming on shared-memory architectures. The parallel HEM has been tested for speed-up, scalability and load balancing for several test cases and typical vortex configurations including one with a non-uniform background vorticity field. It is shown in this paper that the parallel HEM is a decent tool for studying flows with non-uniform background vorticity.

2 The HEM for Contour Dynamics

The spatial discretisation used in a contour dynamics method consists of three parts: The discretisation of the continuous vorticity profile into a piecewise-uniform vorticity distribution, interpolation of the bounding contours of the regions of uniform vorticity, and the redistribution of the nodes on the contours during the simulation. For the present introduction to contour dynamics and the HEM it is sufficient to focus on the first part of the spatial discretisation.

The dynamics of a 2D incompressible, inviscid fluid flow is described by the Euler equation and the equation of mass conservation. The latter implies a divergence-free velocity field or $\nabla \cdot \mathbf{u} = 0$, with $\mathbf{u}(\mathbf{r}, t)$ the velocity vector. The Euler equation, which expresses the balance of linear momentum, is then written as

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p \ , \tag{1}$$

with p the pressure and ρ the density. The vorticity is defined as $\boldsymbol{\omega}(\mathbf{r}, t) = \nabla \times \mathbf{u}$. For a 2D flow, $\mathbf{u} = (u, v)^T$, $\mathbf{r} = (x, y)^T$ and $\boldsymbol{\omega} = \omega\, \mathbf{e}_z$. The non-divergence condition implies the definition of a stream function $\psi(\mathbf{r}, t)$ through $u = \frac{\partial \psi}{\partial y}$ and $v = -\frac{\partial \psi}{\partial x}$. By taking the curl of (1) we obtain an expression for conservation of

[1] More precisely, the potential vorticity is the conserved quantity and should thus be discretised [5].

vorticity of a fluid particle in two dimensions, viz.

$$\frac{D\omega}{Dt} = \frac{\partial\omega}{\partial t} + \frac{\partial\psi}{\partial y}\frac{\partial\omega}{\partial x} - \frac{\partial\psi}{\partial x}\frac{\partial\omega}{\partial y} = 0 \ , \tag{2}$$

whereas the vorticity ω and the stream function ψ are related through the Poisson equation

$$\nabla^2\psi = -\omega \ . \tag{3}$$

Using the Green's function of the Laplace operator for an infinite 2D domain, $G(\mathbf{r};\mathbf{r}') = \frac{1}{2\pi}\ln|\mathbf{r}-\mathbf{r}'|$ (with $|\mathbf{r}| = \sqrt{x^2+y^2}$), the stream function can be found explicitly as

$$\psi(\mathbf{r},t) = -\frac{1}{2\pi}\iint_{\mathbb{R}^2} \omega(\mathbf{r}',t)\,\ln|\mathbf{r}-\mathbf{r}'|\,dx'dy' \ . \tag{4}$$

As is depicted in Figure 1, an initially continuous vorticity distribution $\omega(\mathbf{r},0)$ is approximated by a piecewise-uniform distribution $\hat{\omega}(\mathbf{r},0) = \sum_{m=0}^{M}\omega_m$. It consists in this case of a constant background vorticity ω_0 and a number of nested patches $\mathcal{P}_m$ with vorticity values ω_m (with $m \geq 1$).

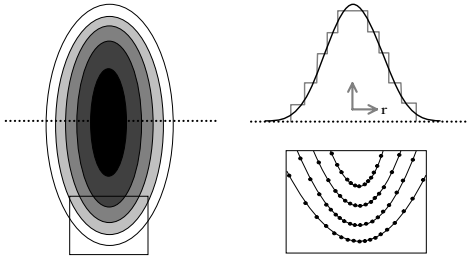

Figure 1 – *Five nested patches with uniform vorticity. Right-top: Vorticity jumps. Right-bottom: Node redistribution.*

We assume for the moment that no background vorticity is present, i.e. $\omega_0 = 0$. Due to the conservation of vorticity the piecewise-uniform distribution of vorticity remains piecewise-uniform in the course of time. The nested patches $\mathcal{P}_m(t)$ deform during the flow evolution although its area is conserved, and the bounding contours $\mathcal{C}_m(t)$ of the patches $\mathcal{P}_m(t)$ will not cut neighbouring boundary contours. Equation (4) can be reformulated as

$$\psi(\mathbf{r},t) = -\frac{1}{2\pi}\sum_{m=1}^{M}\omega_m \iint_{\mathcal{P}_m(t)} \ln|\mathbf{r}-\mathbf{r}'|\,dx'dy' \ . \tag{5}$$

The velocity field $\mathbf{u}(\mathbf{r},t)$ is obtained by taking the derivatives of $\psi(\mathbf{r},t)$ with respect to x and y and subsequently applying Stokes' theorem for a scalar field.

The following expression can be derived [2] [4] [6]:

$$\mathbf{u}(\mathbf{r},t) = -\frac{1}{2\pi}\sum_{m=1}^{M}\omega_m \oint_{\mathcal{C}_m(t)} \ln|\mathbf{r}-\mathbf{r}'|\, d\mathbf{r}' \ , \tag{6}$$

where $d\mathbf{r}'$ denotes an infinitesimal vector tangential to the boundary. From (6) it follows that the evolution of patches of uniform vorticity is fully determined by the evolution of their bounding contours.

For contour dynamics the source distribution consists of patches of uniform vorticity. For the implementation of an acceleration scheme based on Poisson integrals, Vosbeek *et al.* [4] developed the Hierarchical-Element Method in order to reduce the $\mathcal{O}(N^2)$ operation count typical of the conventional contour dynamics approach. The HEM is an adaptation to the method of Anderson [1] and has a time-complexity of $\mathcal{O}(N)$. We present here an overview of the general structure of the HEM which is relevant for the parallelisation strategy.

In the HEM all the patches of uniform vorticity are assumed to reside in a square numerical domain, while the dynamics still involves the infinite plane. This square domain is then subdivided into a set of hierarchical levels with $2^l \times 2^l$ boxes at levels $l = 1, \ldots, l_f$ with l_f a finest level.

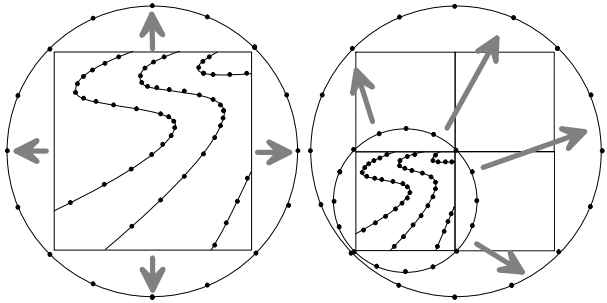

Figure 2 – *Left: Construction of an outer ring around one of the boxes at the finest level by means of direct evaluation. Right: Construction of a ring at a consecutive coarser level. Four smaller rings contribute to one coarser ring. Note that only part of the HEM domain is shown.*

Keeping in mind that our numerical approach is based on the use of rings as computational elements (and using Poisson integrals) we introduce at the finest level l_f a ring with K nodes around each box (Figure 2 left). By means of direct evaluation, the velocity is determined at each node on the ring. These rings are called the outer rings. Four outer rings at the finest level yield the input for the construction of one outer ring at the subsequent coarser level, the parent level, through a Poisson integral. In a similar way outer rings are constructed up to the level $l = 2$ (see Figure 2 right). At level $l = 2$, another kind of ring is formed via the constructed outer rings. This ring, a so-called inner ring, is to be used for the construction of inner rings at finer levels again all the way down to level

$l = l_f - 1$ by means of a similar Poisson integral. This is illustrated in Figure 3 (for $l_f = 4$), at levels $l = 3, \ldots, l_f - 1$ an inner ring is constructed from a parent inner ring at level $l - 1$ and from outer rings at the same level l. Level $l = 1$ (four boxes) is irrelevant in these hierarchical approximations.

The light grey area at each level in Figure 3 is a so-called well-separated area. It is the area with just the optimal distance for a certain box size [4]. The eight dark grey boxes at the finest level are the immediate neighbours of the box containing the evaluation points. Here, direct evaluation of velocities are needed which require $\mathcal{O}(mn)$ operations with m the number of nodes in the eight neighbouring boxes and n the number of nodes in the evaluation box. Outer rings and inner rings are both needed in the hierarchical tree of levels to calculate all the contributions from the subdivided vorticity at the finest level. The construction of the outer rings is necessary to provide information for all the approximating rings. The inner rings contain all information from the outer regions and pass this information on to the next finer level inner ring.

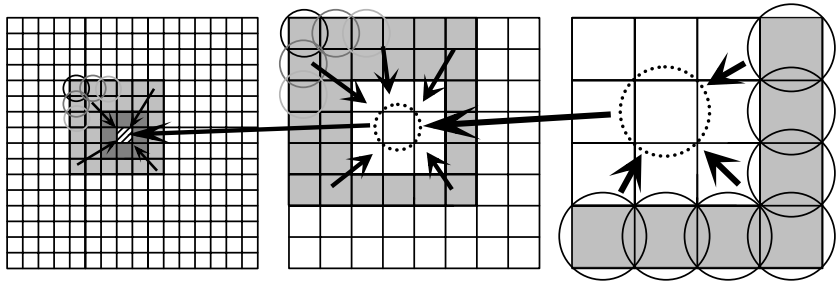

Figure 3 – *The HEM in action for $l_f = 4$. The little box with slanted lines contains evaluation points. Solid rings are outer rings and dotted rings are inner rings.*

If a small number of nodes makes up the vorticity field, it is inefficient to use a very fine meshed finest level with many boxes. In this case, the HEM is even more inefficient than the conventional contour dynamics scheme. On the other hand, however, too many nodes in a box is not efficient either because of the expensive direct interaction computations in each small domain of nine boxes at the finest level. The HEM accounts for these two restrictions by determining the hierarchical tree depth in an adaptive manner during a simulation. Optimal intervals for N are being chosen in such a way that K keeps the same order as the number of nodes per box. This leads to an $\mathcal{O}(N)$ method.

3 Parallelisation Strategy

Parallelisation of the already existing HEM method is achieved by using OpenMP. It is however important to know *a priori* if the numerical scheme can be parallelised in a convenient way. Therefore, the global structure of the HEM should be considered first. Particularly, the parallelisation of the algorithms for the velocity computations and the node redistribution should be taken into account. The numerical structure of the HEM consists of a hierarchy of levels with boxes. At the

finest level these boxes contain pieces of contours, whereas coarser boxes contain the approximating outer and inner rings. Velocity computations are carried out at each level in the tree and parallelisation along all hierarchical boxes seems appropriate. Although most of the computations are at the finest level, the ring computations at each consecutive coarser level—requiring a modest amount of CPU-time—should be included for parallelisation as well. Obviously, parallelisation is most effective when implemented per level. Each level is then decomposed into several subdomains which can have one or several boxes, whereas multiple processors can be assigned to these subdomains. The part of the algorithm dealing with the redistribution of the nodes can not be skipped in the parallelisation process, because it still contributes significantly, depending on the evolving contours. Parallelisation of this part is however completely different, because it has no box-wise HEM structure. For this part, contour-wise parallelisation is adapted. This short paper however will only focus on the much more important box-wise parallelisation of the velocity computations, which is carried out independently from the node redistribution.

The assignment of processors can be scheduled in various ways. Two scheduling policies in OpenMP used for the parallel HEM are static scheduling and dynamic scheduling. When each subdomain is treated as a static subdomain, a processor assigned to this subdomain stays put until all processors have finished their own local job. After that, they proceed in unison to their next assigned subdomain. Static scheduling can be very inefficient when the computational load is non-uniform. However, when an algorithm has a predefined uniform load, processors can be assigned to steady portions of the algorithm and minimize the communication overhead showing an efficient static scheduling policy. Dynamic scheduling, on the other hand, implies that a processor can proceed with the next available subdomain when it has finished its own local job, even when other processors are still busy with their first computation.

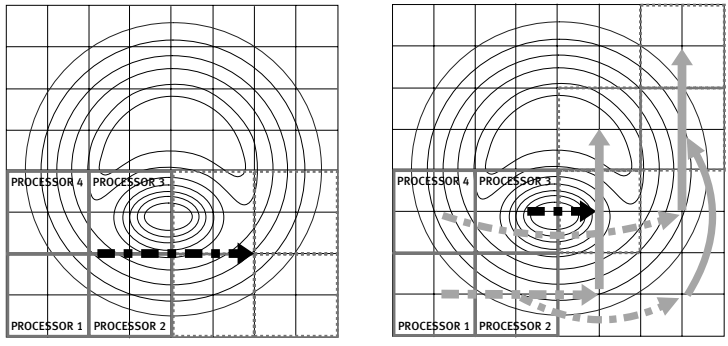

Figure 4 – *Left: Example of static scheduling for the HEM in OpenMP. Right: Dynamic scheduling for the same domain.*

Dynamic scheduling suffers more from communication overhead due to the dynamic displacements of processors. For an unevenly distributed computational load, as is the case in many contour dynamics simulations, processors taking

care of less time-consuming jobs have to wait for the slower jobs and will run idle when static scheduling is applied. This so-called load imbalance is illustrated in Figure 4 for a computation with a non-uniform distribution of nodes in the domain. The left schematic shows a static scheduling which is applied to a finest level $l_f = 3$ with four processors to assign. Choosing an appropriate subdomain size—a so-called chunk size in OpenMP—can be convenient for solving a specific load imbalance problem. The chunk size in Figure 4 is four, and apparently the load is quite out of balance. Processors 1, 2, and 4 have to wait for number 3 before all can proceed in unison to the next subdomain (black arrow). The schematic on the right shows how to solve for the load imbalance when using dynamic scheduling. When processor 1 has finished its computations it will start immediately computing the first available chunk (keeping the ordering in mind). When processor 3—the one with the highest load—has finished (black arrow), processors 1, 2, and 4 have already advanced to the right-top quadrant (gray arrows). The load imbalance for this configuration has been minimized. Parallelisation of the HEM will benefit the most when using a dynamic scheduling policy with the smallest possible chunk size, i.e. as small as one box, contrary to the above illustrative example with four boxes for a chunk. The smaller the chunk size, the more balanced the load is after finishing a certain level.

In order to study the overall parallel performance of the HEM, two important features of parallel programming have been analysed, viz. processor scalability and problem scalability. Processor scalability is given through the parameter S, the speed-up of the parallelisation, and is defined as the ratio between the work done by one processor and the total execution time of a parallel programme with P processors: $S = T_1/T_P$. Problem scalability has been studied for different values of l_f and N. The number of nodes N changes continuously during a simulation and because of this, l_f changes accordingly. When l_f is large, more boxes have to be computed and more communication is necessary between processors. In a shared-memory environment this can be a limiting factor for the parallel performance because communication timing is slow compared to the crucial computations. The more computations a single processor can do in its chunk before communicating its results, the better the overall performance. This feature is called the computation-to-communication ratio.

Originally, the HEM has been introduced as a serial method for accelerating computationally expensive contour dynamics simulations. The use of OpenMP for incremental parallelisation of the method results in a fraction of the method that has been left for serial computation. This fraction depends on a heuristic pre-processing step and lies between 0.5% and 10% of the total amount of computations [3]. For the numerical experiments in the next section this has been taken into account. The serial part of the HEM however acts as a weakest link in the complete computation and slows down the method significantly. This is clearly demonstrated by Amdahl's Law, which states that for an ideal parallel machine (neglecting communication overhead) the speed-up $S_{Amdahl} = \frac{P}{Pf+(1-f)}$, where f represents the serial fraction and P the number of processors.

4 Numerical Experiments and Discussion

Three different numerical experiments have been performed. The goal of the first experiment is to investigate the scalability of the parallelised HEM, without adding additional nodes during the computation or applying any node redistribution. In the second experiment the role of static and dynamic scheduling on the speed-up S is discussed. In this numerical experiment both the number of nodes increases during the simulation (and also l_f increases) and node redistribution is included. In a third numerical experiment the parallel efficiency of the HEM for simulations of vortex evolutions in a non-uniform background vorticity field has been studied. The effort to obtain a parallelised version of the HEM is particularly aimed at solving this latter kind of problems.

Example 1 — A first example is the simulation of concentric patches of uniform vorticity. These patches will stay circular and constant in size. The redistribution of the nodes is therefore unnecessary. During a simulation N is kept constant and the contours rotate in a clockwise direction—due to an overall negative uniform vorticity. The load is distributed symmetrically. To gain insight in the processor scalability, l_f is also kept constant during each simulation. This implies that for a certain N the optimal l_f is not changed accordingly. The following values for P, N, and l_f have been chosen. That is, the range of processors: $P = 1, 2, 4, 8, 16$, the problem size: $N = 1000, 2000, 4000, 8000, 16000$, and the choice of finest level: $l_f = 3, 4$.

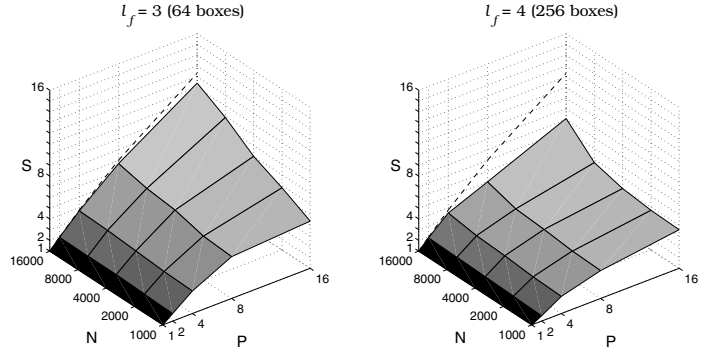

Figure 5 – $S = f(P, N)$ *for* $l_f = 3$ *and* $l_f = 4$. *The speed-up for* $N = 16{,}000$ *has a dashed line indicating Amdahl's limit for a 98% parallelised algorithm* ($f = 0.02$).

Each combination of P, N, and l_f represents a separate simulation. The results for the scalability S as function of P and N are given in Figure 5 as 3D graphs for different values of l_f. According to Figure 5 it is clear that for $N = 16{,}000$, performance is almost ideal for $l_f = 3$ in the complete processor range $P = 1, \ldots, 16$. Furthermore, an increasing computation-to-communication ratio can be observed for increasing N in both graphs. The amount of effective computations per communication step decreases for smaller boxes, resulting in a

smaller S for increasing l_f. This example clearly indicates the parallel efficiency as function of P, N and l_f. For regular simulations however, a constant l_f is certainly not an option. The HEM would lose its $\mathcal{O}(N)$ behaviour.

Example 2 — This example will highlight the difference between static scheduling and dynamic scheduling. The simulation itself requires $\mathcal{O}(10^3)$ nodes and is a relatively 'cheap' computation compared to the kind of simulations the HEM is designed for. The simulation concerns the formation of a tripole from an azimuthally perturbed isolated monopolar vortex. During the simulation the redistribution of the nodes is essential due to the formation of high-curvature segments in the contours. Node redistribution has been parallelised also. Additionally, we should keep in mind that N will increase, and as a result the tree-depth (i.e. l_f) in the HEM will automatically adapt, maintaining an $\mathcal{O}(N)$ operation count. Figure 6a shows the speed-up $S(P)$ for static and dynamic scheduling.

The static scheduling policy is clearly spoiling the parallel performance. Dynamic scheduling delivers a speed-up curve like the one in Figure 5 for $l_f = 3$. This is conform the level updates for $500 < N < 5{,}000$. The finest level remains relatively coarse ($l_f \leq 3$), and some parallel efficiency is lost due to the fact that no optimal benefit of load-balancing can be obtained. As a result a few processors will run idle.

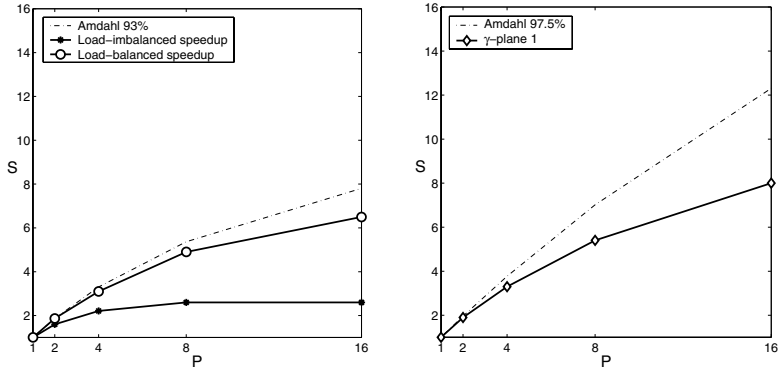

Figure 6 – (a) *Speed-up curves of Example 2. The serial fraction is* $f = 0.07$. (b) *Speed-up curve for the* γ*-plane simulation of Example 3,* $f = 0.025$.

Example 3 — The last example shows the parallel efficiency of simulations the HEM had been made for originally. It concerns the evolution of an isolated monopole—like the one in Example 1—embedded in a non-uniform background vorticity field. These so-called γ-plane simulations generate complex flow patterns and require at least $\mathcal{O}(10^4)$ nodes. The simulation has 21 contours for the background field and the monopole, and a node range of $10{,}000 < N < 30{,}000$. The test starts with only a tenth of this number but arrive in the mentioned range quite early in the simulation. The simulation needs considerably more nodes than the numerical simulations for examples 1 and 2. Redistribution takes

care of the node supply, particularly in the monopolar region, indicating that the load is highly out of balance in the overall domain. This implies the use of dynamic scheduling with the smallest possible chunk size.

Figure 6b shows that the speed-up for the γ-plane is on first site better than the 'cheaper' tripole test, due to a smaller serial fraction. However, the test needs more nodes and will adjust to a higher l_f, thus creating smaller boxes with on the average still the same amount of nodes in a box. The computation-to-communication ratio will unfortunately decrease because of this, although a domain with a greater amount of boxes can benefit more from the load-balancing effect of dynamic scheduling. Apparently the net effect is a lower efficiency.

5 Conclusive remarks

Despite the (inherently involved) shared-addressing complication of the HEM, the parallel HEM scales (very) good for small numbers of processors ($P \leq 8$) for every test-case discussed in this paper. For regular HEM simulations we have however non-uniform node distributions and box-size adaptations, implying a higher computation-to-communication ratio for certain boxes. The adaptations are necessary for keeping the favourable $\mathcal{O}(N)$ operation count of the HEM. For $P > 8$, all tests show that an increasing l_f demands an increasing amount of communication between (expensive) computations, whereas the dynamic scheduling solves for the non-uniform load better when l_f is actually high. A more efficient load-balancing is feasible, i.e. a locally defined tree-depth based on the requirement to keep the average number of nodes per finest level box approximately constant. This approach is not implemented yet and should be pursued in future investigations. Nonetheless, the current parallel HEM is an important improvement to run γ-plane simulations in a much shorter time for small numbers of processors.

References

1. Anderson, C.R.: *An implementation of the fast multipole method without multipoles.* SIAM J. Sci. Statist. Comput. **13** (1992) 923–947
2. Dritschel, D.: *Contour dynamics and contour surgery: Numerical algorithms for extended, high-resolution modelling of vortex dynamics in two-dimensional, inviscid, incompressible flows.* Comput. Phys. Rep. **10** (1989) 77–146
3. Schoemaker, R.M., de Haas, P.C.A., Clercx, H.J.H., Mattheij, R.M.M.: *A parallel hierarchical-element method for contour dynamics simulations.* to appear
4. Vosbeek, P.W.C., Clercx, H.J.H., Mattheij, R.M.M.: *Acceleration of contour dynamics simulations with a hierarchical-element method.* J. Comput. Phys. **161** (2000) 287–311
5. Vosbeek, P.W.C., Clercx, H.J.H., van Heijst, G.J.F., Mattheij, R.M.M.: *Contour dynamics with non-uniform background vorticity.* Int. J. Comput. Fluid Dyn. **15** (2001) 227-249
6. Zabusky, N.J.: *Contour dynamics for the Euler equations in two dimensions.* J. Comput. Phys. **30** (1979) 96–106