A domain decomposition parallelization of the Fast Marching Method

By M. Herrmann

1. Motivation and objectives

Evolving interfaces play an important role in a multitude of different areas, ranging from fluid mechanics, combustion, and grid generation to material sciences, semiconductor manufacturing, seismic analysis, and control problems [see Sethian (1999*b*) for a detailed overview]. Traditionally, interfaces have been treated in a Lagrangian framework tracking their evolution by, for example, marker particles [see among others Brackbill *et al.* (1988)]. In recent years, however, describing the topology and evolution of interfaces by Eulerian partial differential equations (PDE) has become ever more popular since this approach offers certain theoretical and computational advantages over the Lagrangian formulation (Sethian 1999*b*). Depending on the type of problem, two different solution strategies for the Eulerian approach exist. In the case of an initial value problem, level set methods (Osher & Sethian 1988), or alternatively Volume-of-Fluid methods (Noh & Woodward 1976), can be employed to solve the evolving interface. In the case of a boundary value problem, the Fast Marching Method (Sethian 1996*a*) has emerged as the efficient solution method.

In this paper, we focus on two boundary value problems that typically arise in the numerical implementation of level set methods, namely reinitialization and redistribution. In level set methods, an iso-surface of the level set scalar G,

$$G(\boldsymbol{x},t) = G_0 = \text{const}\,, \tag{1.1}$$

is used to define the location of an arbitrary shaped interface Γ. The transport equation for the scalar G can then be derived from simple kinematic considerations as

$$\frac{\partial G}{\partial t} + \boldsymbol{u}\cdot\nabla G = 0\,. \tag{1.2}$$

Since this so-called level set equation (1.2) is valid only at the location of the interface itself, the choice of G outside the interface, i.e. $G \neq G_0$, is in principle arbitrary. However, for numerical reasons, G is generally chosen to be a distance function,

$$|\nabla G|\Big|_{G\neq G_0} = 1\,. \tag{1.3}$$

Enforcing this condition is usually called reinitialization.

Furthermore, some quantities S may be defined only at the location of the interface. In order to extend these quantities to the whole computational domain, they are set constant in the interface normal direction,

$$\nabla S\cdot\nabla G = 0\,. \tag{1.4}$$

This procedure is called redistribution, since it redistributes the values of S from the interface into the surrounding domain.

Both, Eqs. (1.3) and (1.4) constitute boundary value problems, since G and S are defined on Γ only. Several different numerical methods exist to solve these equations. Among these are brute force approaches based on calculating the minimum of the geometric distance between each grid node in the computational domain and every point on the interface (Merriman *et al.* 1994), and PDE-based approaches like the iteration scheme by Sussman *et al.* (1994). While the former are computationally very expensive, the latter inadvertently alter both the location of interface and the value of S on the interface due to their iterative nature. In the case of the reinitialization equation (1.3), supplementary fixes addressing this problem relatively successfully have been proposed (Russo & Smereka 2000; Peng *et al.* 1999; Sussman *et al.* 1998; Sussman & Fatemi 1999; Enright *et al.* 2002).

An alternative approach to solving Eqs. (1.3) and (1.4) is the so-called Fast Marching Method (FMM). It was originally proposed by Tsitsiklis (1994, 1995) and applied to the level set formulation by Sethian (1996*a*) and Helmsen *et al.* (1996). The FMM is a non-iterative procedure that explicitly makes use of the way information in Eqs. (1.3) and (1.4) propagates. The FMM is thus theoretically optimal in its operation count. Still, typical problem sizes of state of the art numerical simulations in general require a domain decomposition approach for parallel computing. However, the FMM is highly sequential and hence not straightforward to parallelize in a domain decomposition sense. At least to the knowledge of the author, no domain decomposition parallelization of the Fast Marching Method has been published yet.

This paper is structured as follows: first, the standard, sequential FMM is reviewed. Then, different parallelization strategies are discussed and a domain decomposition parallelization is proposed. Thereafter, some preliminary results concerning the speedup of the proposed method are presented and discussed. Finally, conclusions are drawn and an outlook to future work is given.

2. The sequential Fast Marching Method

In this section, a short overview of the standard Fast Marching Method is given. For further details, the interested reader is referred to Sethian (1996*a*), Adalsteinsson & Sethian (1999), and Sethian (1999*a*).

The idea of the FMM for level sets is to first solve Eq. (1.3) and use its solution to then solve Eq. (1.4) (Adalsteinsson & Sethian 1999). Hence, we will at first focus on the solution of Eq. (1.3).

To solve Eq. (1.3) correctly, the gradient operator has to be approximated by upwind, entropy-satisfying finite differences (Sethian 1999*a*). The approximation most often used is due to Godunov (Rouy & Tourin 1992):

$$\begin{aligned} |\nabla G| \approx \; & \Big[\max(D^{-x}_{ijk}G, -D^{+x}_{ijk}G, 0)^2 + \\ & \max(D^{-y}_{ijk}G, -D^{+y}_{ijk}G, 0)^2 + \\ & \max(D^{-z}_{ijk}G, -D^{+z}_{ijk}G, 0)^2\Big]^{1/2} \;, \end{aligned} \tag{2.1}$$

where $D^{\pm}_{ijk}$ are difference notations. For example, the first order approximation is

$$D^{-x}_{ijk}G = \frac{G_{ijk} - G_{i-1jk}}{\Delta x} \;, \qquad D^{+x}_{ijk}G = \frac{G_{i+1jk} - G_{ijk}}{\Delta x} \;. \tag{2.2}$$

(a) Calculate all node values that are directly adjacent to the interface and tag them as *accepted*. Tag all nodes adjacent to these *accepted* nodes as *close* and all others as *far*.

(b) Calculate G of all *close* nodes by Eq. (2.3), treating G in any adjacent *close* or *far* node as ∞. Set the loop index $n = 1$.

(c) Mark as *accepted* the *close* node ijk with the smallest G value, denoted by $G^n = G_{ijk}$.

(d) Mark all *far* nodes adjacent to G_{ijk} as *close*.

(e) Recalculate the G values of all *close* nodes adjacent to G_{ijk} by Eq. (2.3), treating G in any adjacent *close* or *far* node as ∞.

(f) Set $n = n + 1$ and return to step (c) until all nodes are *accepted*.

TABLE 1. The sequential FMM algorithm

Thus, the discretized version of Eq. (1.3) solved in the FMM reads as

$$\begin{aligned}&\Big[\max(D^{-x}_{ijk}G, -D^{+x}_{ijk}G, 0)^2 + \\ &\max(D^{-y}_{ijk}G, -D^{+y}_{ijk}G, 0)^2 + \\ &\max(D^{-z}_{ijk}G, -D^{+z}_{ijk}G, 0)^2\Big]^{1/2} = 1\,.\end{aligned} \tag{2.3}$$

Provided that the G values of all nodes neighboring ijk are given, Eq. (2.3) constitutes a quadratic equation yielding G_{ijk} itself. The simple, albeit inefficient way to solve Eq. (2.3) throughout the whole computational domain is to iteratively update each node in the domain by Eq. (2.3) until a stationary solution is reached (Rouy & Tourin 1992).

However, this approach neglects to take advantage of the fact that, due to the upwind structure of Eq. (2.3), information propagates only from smaller to larger values of G. This yields attribute 1 of the FMM:

ATTRIBUTE 1. *A node value G_{ijk} is determined only by those neighboring nodes of smaller value. It can thus globally depend at most on those nodes in the domain that are of smaller value.*

Attribute 1 implies that a smallest node is fixed and cannot change its value. Hence, given appropriate boundary conditions for G_{ijk} at or adjacent to the interface $G = G_0$, updates of G_{ijk} according to Eq. (2.3) can be confined to a narrow band around the globally smallest values that sweeps outward to ever larger values of G_{ijk}. For details see Sethian (1996*a*), Sethian (1999*a*), and Adalsteinsson & Sethian (1999).

In summary, this leads to the sequential FMM algorithm given in Table 1. The algorithm is executed once for all nodes $G^0_{ijk} < G_0$ and once for all nodes $G^0_{ijk} \geq G_0$, where the superscript 0 denotes the initial values of G at node ijk. Furthermore, the following attribute of the Fast Marching Method can be discerned:

ATTRIBUTE 2. *The sequential loop steps (c)-(f) sort all accepted nodes that are not initially accepted in step (a) in ascending order, i.e. $G^{n+1} \geq G^n$.*

Using a heap sort algorithm with back pointers (Sethian 1996*b*) to locate the smallest value G in step (c) makes the sequential FMM algorithm highly efficient with a theoretical operation count of $O(N \log N)$.

(A_m)	Perform steps (a) and (b) of the sequential FMM algorithm.
(B_m)	Send all nodes with $G^0_{ijk} < G_0$ to process #0, all nodes with $G^0_{ijk} \geq G_0$ to process #1.
(C_m)	Perform sequential FMM algorithm steps (c)-(f).
(E_m)	Receive results for nodes $G^0_{ijk} < G_0$ from process #0 and results for nodes $G^0_{ijk} \geq G_0$ from process #1.

TABLE 2. The parallel FMM algorithm $\mathcal{P}_1$

3. Parallelizing the Fast Marching Method

In this section, different strategies to parallelize the FMM are discussed. First, a simple parallelization strategy not based on domain decomposition is given, followed by the discussion of several domain decomposition parallelization approaches.

3.1. G_0*-based parallelization*

The trivial way to parallelize the FMM algorithm described in table 1 is to execute the complete sequential algorithm for all nodes $G^0_{ijk} < G_0$ and for all nodes $G^0_{ijk} \geq G_0$ in parallel, since neither region influences the other. The resulting algorithm $\mathcal{P}_1$ is summarized in Table 2.

This parallelization strategy has two obvious drawbacks. First, the parallelization is limited to two parallel processes, since only two independent regions exist. Secondly, depending on the problem size, memory resource problems arise, because both processes need to work on the whole computational domain.

3.2. *Domain decomposition parallelization*

Domain decomposition parallelization of the sequential FMM algorithm poses two problems. First, the globally smallest *close* value has to be located in step (c). Although this procedure is non-local by definition, it still is easy to parallelize, since each domain can compute its locally smallest value independently, and then simply use these to find a global minimum. Second, a new globally smallest value G^n can be found in step (c) only after steps (d)-(e) of the previous loop step $n-1$ have been executed.

Nevertheless, leaving at first efficiency considerations aside, domain decomposition is straightforward. In order to simplify the notation in the following, we only discuss the domain decomposition into two neighboring domains $\mathcal{D}_m$ and $\mathcal{D}_{m+1}$. All arguments, however, apply analogous to the three-dimensional domain decomposition into an arbitrary number of domains. Figure 1 depicts some naming conventions that are employed in the following. Each domain $\mathcal{D}_m$ is extended by r ghost nodes in the domain boundary normal direction, where r is the spatial order of the difference approximation, Eq. (2.2). Here, only the first order approximation is employed. Hence, each domain is extended as depicted in Fig. 1.

Assuming that each process $m = 0 \ldots N_p$ works only on part $\mathcal{D}_m$ of the whole computational domain, Table 3 summarizes the parallel FMM algorithm $\mathcal{P}_2$. Note that $\mathcal{P}_2$ has to be executed twice on each process, once for all nodes $G^0_{ijk} < G_0$ and once for all nodes $G^0_{ijk} \geq G_0$. In essence, disregarding step (A), algorithm $\mathcal{P}_2$ is a domain decomposed sequential algorithm, because globally only one single node in a single domain is updated per loop step, while all other domains are waiting idle. However, algorithm $\mathcal{P}_2$ introduces a clearly defined inter-domain communication boundary. Recalling that the update of

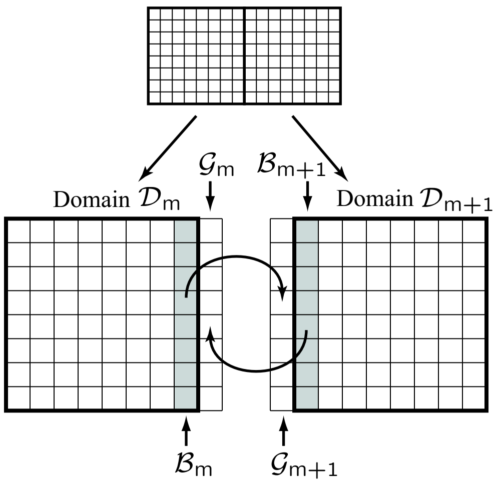

FIGURE 1. Domain naming conventions.

(A) Perform steps (a) and (b) of the sequential FMM algorithm.

(B) Locate the locally smallest *close* G value; denote it as $G_{\mathcal{D}_m}$.

(C) Find the globally smallest *close* G value, $G_G = \min(G_{\mathcal{D}_{0\ldots N_p}})$, and mark it as *accepted*. If $G_G \in \mathcal{D}_m$, go to next step, else go to step (F).

(D) Perform sequential FMM algorithm steps (c)-(e) for G_G.

(E) If any of the *close* nodes, updated in step (e) of the sequential FMM, belong to $\mathcal{B}_m$, communicate them to $\mathcal{G}_{m+1}$ of domain $\mathcal{D}_{m+1}$.

(F) Set $n = n + 1$. Return to step (B) until all local nodes are *accepted*.

TABLE 3. The parallel FMM algorithm $\mathcal{P}_2$

G_{ijk}, Eq. (2.3), using first order approximations, Eq. (2.2), does involve at most the directly adjacent nodes in the $\pm i$, $\pm j$, and $\pm k$ direction, any local node in domain $\mathcal{D}_m$ can be updated correctly, provided that all ghost nodes $\mathcal{G}_m = \mathcal{B}_{m+1}$ are known. Hence, only changes of $\mathcal{B}_{m+1}$ need to be communicated to $\mathcal{G}_m$, as is done in step (E) above.

Taking these arguments into account, one can in fact avoid the strict sequential nature of algorithm $\mathcal{P}_2$. Looking for example at domain $\mathcal{D}_m$, as long as no change in $\mathcal{G}_m$ occurs, the locally smallest *close* value $G_{\mathcal{D}_m}$ can be moved into a local *accepted* group and updates of the nodes adjacent to $G_{\mathcal{D}_m}$ can be performed according to the sequential FMM algorithm steps (d) and (e).

However, special care must be taken whenever a node belonging to $\mathcal{G}_m$, for example G_{ijk}, changes. If G_{ijk} changes to *accepted* status, then, recalling attribute 1, all locally *accepted* nodes belonging to $\mathcal{D}_m$ that are smaller than or equal to G_{ijk} cannot be influenced by this change. Conversely, all locally *accepted* nodes belonging to $\mathcal{D}_m$ larger than G_{ijk} might be wrong, since they could depend on G_{ijk}. Hence, to allow for a consistent algorithm, each domain has to be able to rollback to its state at the beginning of loop step p, where p is such that $G^{p-1} \leq G_{ijk} \leq G^p$. The same argument applies, if G_{ijk} changes from *accepted* to *close* status through a rollback operation in domain $\mathcal{D}_{m+1}$.

(A) Perform steps (a) and (b) of the sequential FMM algorithm.

(B) Check, if any node belonging to $\mathcal{G}_m$ changed status with new value G_B. If so, rollback to state S^p, where p is given by $G^{p-1} \leq G_B \leq G^p$. Mark G_B as *close* and insert it into list $\mathcal{L}_\mathcal{B}$. Set $n = p$.

(C) Locate the locally smallest *close* G value (including list $\mathcal{L}_\mathcal{B}$); denote it as $G_{\mathcal{D}_m}$.

(D) Perform steps (c)-(e) of the sequential FMM algorithm for $G_{\mathcal{D}_m}$.

(E) If node $G_{\mathcal{D}_m}$ or any of its adjacent nodes updated in step (D) belongs to $\mathcal{B}_m$, communicate them to $\mathcal{G}_{m+1}$ of domain $\mathcal{D}_{m+1}$.

(F) Store the current state as S^n.

(G) Set n=n+1. Return to step (B) until all local nodes are *accepted*.

(H) Wait until all other domains reach step (H) or a node belonging to $\mathcal{G}_m$ changes status. In the latter case, go to step (B).

Table 4. The parallel FMM algorithm $\mathcal{P}_3$

These considerations lead to the domain decomposed parallel algorithm $\mathcal{P}_3$ summarized in Table 4.

The drawback of algorithm $\mathcal{P}_3$ is two-fold. First, the complete state of the local domain has to be stored at every loop step in order to allow for a possible rollback to this state. Second, any change in status of a node belonging to $\mathcal{G}_m$ leads to a rollback operation in domain $\mathcal{D}_m$. To overcome these shortcomings, we will make use of the following additional attribute of the FMM:

Attribute 3. *Let G^p_{ijk} be the solution to Eq. (2.3) at loop step p. If any single one of the adjacent nodes $i'j'k'$ becomes smaller, i.e. $G^{p'}_{i'j'k'} \leq G^p_{i'j'k'}$ with $p' > p$, then a subsequent update of node ijk by Eq. (2.3) yields $G^{p'}_{ijk} \leq G^p_{ijk}$.*

The proof of attribute 3 is straightforward. Attribute 3 implies that any node which has been locally *accepted* at loop step p and is then rolled back to status *close*, can retain its G^p_{ijk} value, because any subsequent update will either decrease its value or leave it unchanged. Its change back to *accepted* status at a later loop step is thereby uninfluenced. Following the same line of argument, all neighboring *close* nodes can also retain their $G^p_{i'j'k'}$ values. Thus, step (B) of $\mathcal{P}_3$ needs to rollback only the status of the nodes from *accepted* back to *close*, but does not need to rollback their node values. Furthermore, attribute 3 implies that only the change to *accepted* status of a node in $\mathcal{G}_m$ needs to initiate a rollback.

Incorporating attribute 3, the final domain decomposed parallel algorithm $\mathcal{P}_4$ is given in Table 5. Obviously, the efficiency of algorithm $\mathcal{P}_4$ depends on the required amount of inter-domain communication, i.e. how often nodes belonging to $\mathcal{B}_m$ change to *accepted* status, and how many rollback operations are required in step (B). If the solution of Eqs. (1.3) and (1.4) is required only up to a certain distance T away from the $G = G_0$ interface, then, naturally, only those nodes within this band, $|G^0_{ijk} - G_0| \leq T$, need to be considered in the FMM algorithm. Depending on the geometry of the $G = G_0$ interface, this so-called narrow band approach (Peng *et al.* 1999) may drastically reduce the necessary inter-domain communication, as illustrated in Fig. 2. In fact, as long as there are no nodes ijk belonging to $\mathcal{B}_m$ with $|G^0_{ijk} - G_0| \leq T$, no inter-domain communication is

(A) Perform steps (a) and (b) of the sequential FMM algorithm.

(B) Check, if any node belonging to $\mathcal{G}_m$ changed status to *accepted* and decreased its value from a previously *accepted* value to the new value G_B. If so, rollback by marking all nodes $G^{p\cdots n}$ as *close*, where p is given by $G^{p-1} \leq G_B \leq G^p$. Mark G_B as *close* and insert it into list $\mathcal{L_B}$. Set $n = p$.

(C) Locate the locally smallest *close* G value (including list $\mathcal{L_B}$); denote it as $G_{\mathcal{D}_m}$.

(D) Perform steps (c)-(e) of the sequential FMM algorithm for $G_{\mathcal{D}_m}$.

(E) If node $G_{\mathcal{D}_m}$ belongs to $\mathcal{B}_m$, communicate it to $\mathcal{G}_{m+1}$ of domain $\mathcal{D}_{m+1}$.

(F) Set n=n+1. Return to step (B) until all local nodes are *accepted*.

(G) Wait until all other domains reach step (G) or a node belonging to $\mathcal{G}_m$ changes to *accepted* status. In the latter case go to step (B).

TABLE 5. The parallel FMM algorithm $\mathcal{P}_4$

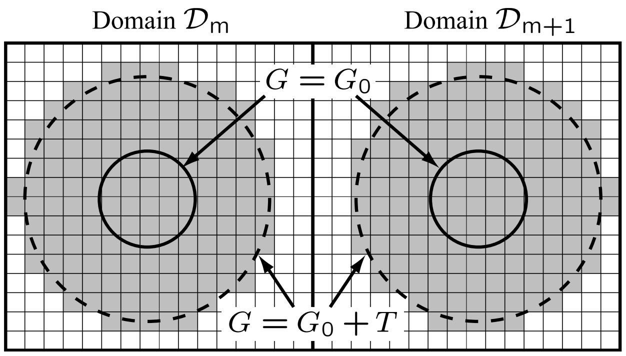

FIGURE 2. Narrow band approach.

required at all and the problem completely decouples. Although in most applications this is not the case, the narrow band approach can still reduce the number of inter-domain communications substantially.

4. Redistribution

The preceding sections dealt exclusively with the solution of the reinitialization equation (1.3) as the prerequisite for solving the redistribution equation (1.4). The basic idea in solving Eq. (1.4) is to again confine its solution to a small band around the globally smallest G_{ijk} values that marches outward to ever larger values of G_{ijk}.

To this end, first, initial values of S have to be calculated at all nodes directly adjacent to the $G = G_0$ interface by either first order approximations (Adalsteinsson & Sethian 1999) or higher order schemes (Chopp 2001). Then, Eq. (1.4) is approximated by

$$\begin{aligned}
&A^{-x}\left(D^{-x}_{ijk}G^n \cdot D^{-x}_{ijk}S\right) + A^{+x}\left(D^{+x}_{ijk}G^n \cdot D^{+x}_{ijk}S\right) + \\
&A^{-y}\left(D^{-y}_{ijk}G^n \cdot D^{-y}_{ijk}S\right) + A^{+y}\left(D^{+y}_{ijk}G^n \cdot D^{+y}_{ijk}S\right) + \\
&A^{-z}\left(D^{-z}_{ijk}G^n \cdot D^{-z}_{ijk}S\right) + A^{+z}\left(D^{+z}_{ijk}G^n \cdot D^{+z}_{ijk}S\right) = 0\,. \qquad (4.1)
\end{aligned}$$

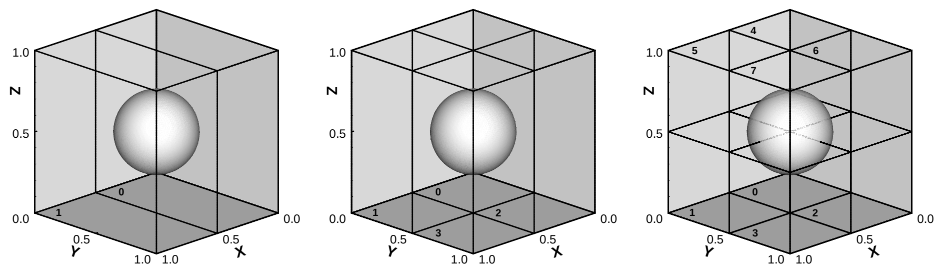

FIGURE 3. Optimal domain decomposition into 2, 4, and 8 domains.

Here, the switch A^{-x} is given by

$$A^{-x} = \begin{cases} 1 & : \quad \max\left(D_{ijk}^{-x}G^n, -D_{ijk}^{+x}G^n, 0\right) = D_{ijk}^{-x}G^n \\ 0 & : \quad \text{otherwise} \,, \end{cases} \tag{4.2}$$

with all other A defined accordingly. The switch A ensures that only those nodes are used to evaluate S_{ijk} in Eq. (4.1) that also contributed to the update of G_{ijk}^n, Eq. (2.3). In every FMM loop step, Eq. (4.1) is only solved for the node that changes status to *accepted* in step (c) of the sequential FMM. If this node belongs to $\mathcal{B}_m$, its new value S^n has to be communicated to the corresponding ghost node in domain $\mathcal{D}_{m+1}$ in step (E) of the parallel FMM algorithm $\mathcal{P}_4$.

5. Results

To evaluate the performance of the proposed parallel FMM algorithm $\mathcal{P}_4$ to solve Eqs. (1.3) and (1.4), the results of four different sets of computations are presented in the following. All four sets are three-dimensional and based upon the same interface geometry, composed of a sphere with radius $R_0 = 0.25$ and center located at $\boldsymbol{x}_c = (0.5, 0.5, 0.5)$. The level set scalar field representing this sphere is given by

$$G(\boldsymbol{x}) = R_0 - |\boldsymbol{x} - \boldsymbol{x}_c| \,. \tag{5.1}$$

The distribution of $\boldsymbol{S}$ on the interface is set to

$$\boldsymbol{S}(\boldsymbol{x}) = \cos\left(\arctan\left(\frac{y-y_c}{z-z_c}\right)\right)\sin\left(\arctan\left(\frac{x-x_c}{\sqrt{(y-y_c)^2+(z-z_c)^2}}\right)\right) \,. \tag{5.2}$$

All computations are performed on a global domain of size [0,1]x[0,1]x[0,1] discretized by 192x192x192 equidistant cartesian grid nodes.

5.1. *Optimal domain decomposition*

As mentioned in section 1, information in Eqs. (1.3) and (1.4) propagates in the interface normal direction. Hence, a domain decomposition such that all domain boundaries are parallel to the interface normal vectors is optimal in the sense that no information does cross these boundaries. Figure 3 shows such a decomposition into 2, 4, and 8 domains. This domain decomposition is also optimal with regard to load balancing. Since all domains contain the exact same number of nodes $G_{ijk} < G_0$ as well as $G_{ijk} \geq G_0$, the load imbalance factor is $F_l = 0$, see Eq. (5.5) defined later.

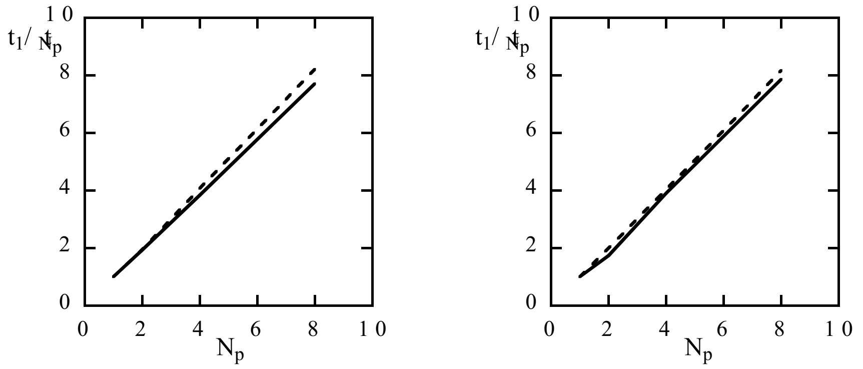

FIGURE 4. Speedup due to parallel FMM employing the narrow band approach (left) and whole domain update (right) with the optimal decomposition into 1, 2, 4, and 8 domains. Dashed line denotes theoretically possible speedup, straight line represents attained speedup.

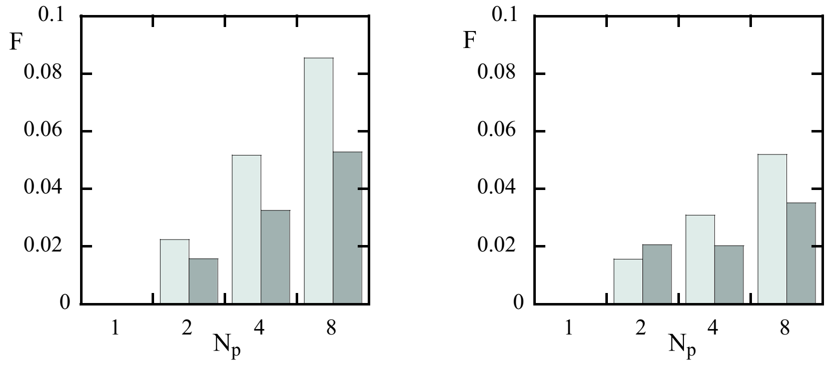

FIGURE 5. Overhead due to parallel FMM employing the narrow band approach (left) and whole domain update (right) with the optimal decomposition into 1, 2, 4, and 8 domains. Shown are communication factor F_c (▫) and rollback factor F_r (▪).

Two different sets of computations were performed. The first set employs the narrow band approach with the width of the band set to $T = 8\Delta x$. The second set performs the FMM throughout the whole computational domain.

Figure 4 shows a comparison of the speedup for both cases to the theoretically possible speedup. The theoretically possible speedup was determined by calculating only domain number 0, see Fig. 3, using Neumann boundary conditions. Note that, since the operation count of the sequential FMM is $O(N \log N)$, slightly hyperlinear speedup is theoretically possible. However, the actual efficiency attained in the parallel computation is approximately 0.96 for the narrow band approach and roughly 0.98 for the whole domain update.

Figure 5 illustrates the overhead due to the parallelization. Depicted are the communication factor F_c, defined as

$$F_c = \frac{\text{total number of communication operations}}{\text{total number of nodes}}\,, \tag{5.3}$$

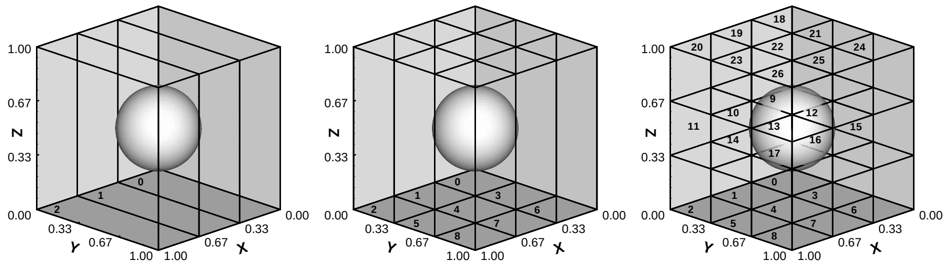

FIGURE 6. Non-optimal domain decomposition into 3, 9, and 27 domains.

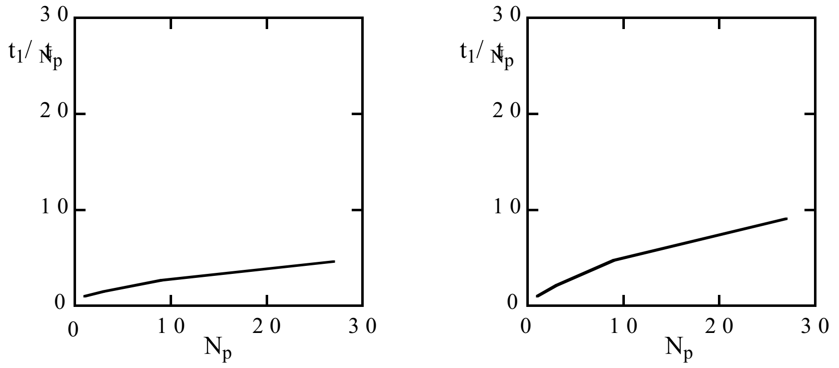

FIGURE 7. Speedup due to parallel FMM employing the narrow band approach (left) and whole domain update (right) with the non-optimal decomposition into 1, 3, 9, and 27 domains.

and the rollback factor F_r, defined as

$$F_r = \frac{\text{total number of rollback operations}}{\text{total number of nodes}} . \tag{5.4}$$

As expected, with optimal domain decomposition, the amount of communication and rollback operations is very small, indicating that the overhead due to parallelization is minimal. This result is consistent with the observed high efficiency.

5.2. *Non-optimal domain decomposition*

When applying the parallel FMM to actual problems, an optimal domain decomposition, as presented in the previous section, is not always viable. In this non-optimal case, three major factors can impact the performance of the parallel algorithm: load imbalancing, communication overhead, and rollback overhead. To illustrate their effect, two sets of computations are performed using a decomposition into 1, 3, 9, and 27 domains as depicted in Fig. 6. The first set again uses the narrow band approach with the width of the band set to $T = 8\Delta x$, whereas the second set calculates the FMM throughout the whole computational domain.

Figure 7 shows the speedup attained for both the narrow band approach on the left and the whole domain update on the right. As can be seen, speedup, and thus efficiency, is significantly lower than the optimal domain decomposition case, see Fig. 4. The efficiency decreases to 0.17 for the narrow band approach and 0.34 for the whole domain update.

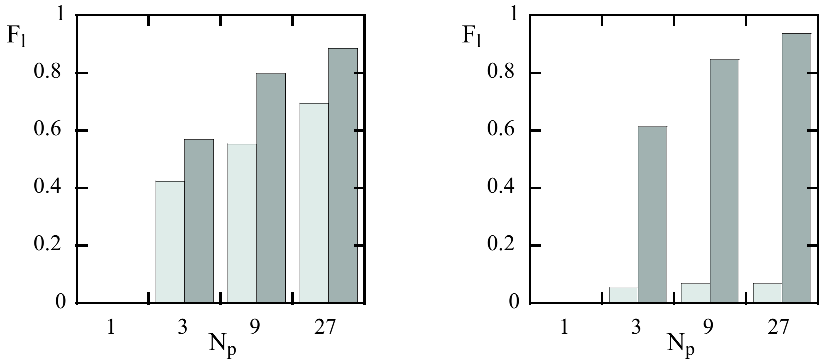

FIGURE 8. Load imbalance factor F_l due to parallel FMM employing the narrow band approach (left) and whole domain update (right) with the non-optimal decomposition into 1, 3, 9, and 27 domains. Shown is F_l for $G_{ijk} < G_0$ (■) and F_l for $G_{ijk} \geq G_0$ (■).

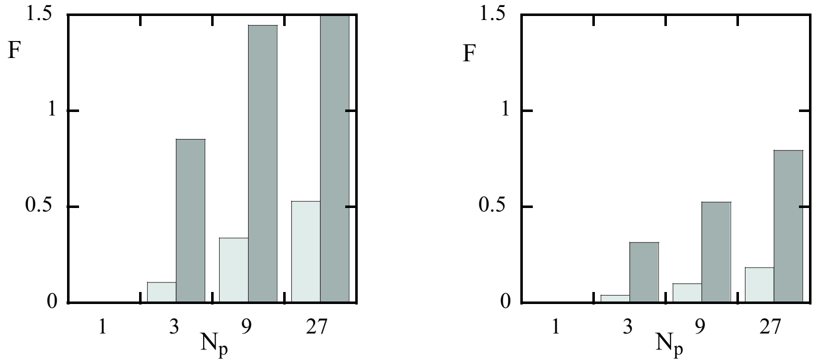

FIGURE 9. Overhead due to parallel FMM employing the narrow band approach (left) and whole domain update (right) with the non-optimal decomposition into 1, 3, 9, and 27 domains. Shown are communication factor F_c (■) and rollback factor F_r (■).

To further analyze this behavior, the load imbalance factor

$$F_l = 1 - \frac{\text{maximum number of nodes per domain}}{\text{average number of nodes per domain}} \tag{5.5}$$

for each of the four different domain decompositions is depicted in Fig. 8. Since the parallel FMM algorithm is called twice, once for all nodes $G_{ijk} < G_0$ and once for all nodes $G_{ijk} \geq G_0$, F_l is calculated accordingly. Obviously, the load imbalance factors for both sets are relatively large. This explains the significantly reduced efficiency of the non-optimal domain decomposition as compared to the optimal domain decomposition. Otherwise, the load imbalance factors for $G_{ijk} \geq G_0$ are roughly the same for both sets. However, the load imbalance factor for $G_{ijk} < G_0$ is significantly larger in the narrow band approach than in the whole domain update, leading to the lower speedup and efficiency in the narrow band approach as compared to the whole domain update.

Figure 9 exhibits the communication factor F_c and the rollback factor F_r. Compared to the optimal domain decomposition case (Fig. 5) both factors are one order of magnitude larger. Furthermore, F_c and F_r are significantly larger in the narrow band approach as opposed to the whole domain update. This is due to the aforementioned higher load imbalance in the narrow band approach.

6. Conclusions and future work

In this paper, the first domain decomposition parallelization of the Fast Marching Method for level sets has been presented. Parallel speedup has been demonstrated in both the optimal and non-optimal domain decomposition case. The parallel performance of the proposed method is strongly dependent on load balancing separately the number of nodes on each side of the interface. A load imbalance of nodes on either side of the domain leads to an increase in communication and rollback operations. Furthermore, the amount of inter-domain communication can be reduced by aligning the inter-domain boundaries with the interface normal vectors. In the case of optimal load balancing and aligned inter-domain boundaries, the proposed parallel FMM algorithm is highly efficient, reaching efficiency factors of up to 0.98.

Future work will focus on the extension of the proposed parallel algorithm to higher order accuracy. Also, to further enhance parallel performance, the coupling of the domain decomposition parallelization to the G_0-based parallelization will be investigated.

Acknowledgments

The support of the German Research Foundation (DFG) is gratefully acknowledged. The author also would like to thank Frank Ham and Yuan-Nan Young for many helpful discussions.

REFERENCES

Adalsteinsson, D. & Sethian, J. A. 1999 The fast construction of extension velocities in level set methods. *J. Comput. Phys.* **148**, 2–22.

Brackbill, J. U., Kothe, D. B. & Ruppel, H. M. 1988 FLIP: A low dissipation, particle-in-cell method for fluid flow. *Comput. Phys. Commun.* **48**, 25–38.

Chopp, D. L. 2001 Some improvements of the fast marching method. *SIAM J. Sci. Comput.* **23**, 230–244.

Enright, D., Fedkiw, R., Ferziger, J. & Mitchell, I. 2002 A hybrid particle level set method for improved interface capturing. *J. Comp. Phys.* **183**, 83–116.

Helmsen, J., Puckett, E., Colella, P. & Dorr, M. 1996 Two new methods for simulating photolithography development in 3D. *Proc. SPIE* **2726**, 253–261.

Merriman, B., Bence, J. & Osher, S. 1994 Motion of multiple junctions: A level set approach. *J. Comput. Phys.* **112**, 334.

Noh, W. F. & Woodward, P. 1976 SLIC (Simple Line Interface Calculation). In *Lecture Notes in Physics Vol. 59, Proceedings of the Fifth International Conference on Numerical Methods in Fluid Dynamics* (ed. A. I. V. D. Vooren & P. J. Zandenbergen), pp. 330–340. Berlin: Springer.

Osher, S. & Sethian, J. A. 1988 Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.* **79**, 12–49.

Peng, D., Merriman, B., Osher, S., Zhao, H. & Kang, M. 1999 A PDE-based fast local level set method. *J. Comput. Phys.* **155**, 410–438.

Rouy, E. & Tourin, A. 1992 A viscosity solution approach to shape-from-shading. *SIAM J. Num. Anal.* **29**, 867–884.

Russo, G. & Smereka, P. 2000 A remark on computing distance functions. *J. Comput. Phys.* **163**, 51–67.

Sethian, J. A. 1996*a* A fast marching level set method for monotonically advancing fronts. *Proc. Natl. Acad. Sci. USA* **93**, 1591–1595.

Sethian, J. A. 1996*b* *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Material Science*. Cambridge, UK: Cambridge University Press.

Sethian, J. A. 1999*a* Fast marching methods. *SIAM Review* **41** (2), 199–235.

Sethian, J. A. 1999*b* *Level Set Methods and Fast Marching Methods*, 2nd edn. Cambridge, UK: Cambridge University Press.

Sussman, M. & Fatemi, E. 1999 An efficient, interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow. *SIAM J. Sci. Comput.* **20** (4), 1165–1191.

Sussman, M., Fatemi, E., Smereka, P. & Osher, S. 1998 An improved level set method for incompressible two-phase flows. *Comp. Fluids* **27** (5-6), 663–680.

Sussman, M., Smereka, P. & Osher, S. 1994 A level set method for computing solutions to incompressible two-phase flow. *J. Comput. Phys.* **119**, 146.

Tsitsiklis, J. 1994 Efficient algorithms for globally optimal trajectories. In *Proceedings of the 33rd Conference on Decision and Control*, pp. 1368–1373. Lake Buena Vista.

Tsitsiklis, J. 1995 Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control* **40**, 1528–1538.