Advanced parallel computing in material forming with CIMLib

Youssef Mesri — Hugues Digonnet — Thierry Coupez

MINES ParisTech, CEMEF - Centre de Mise en Forme des Matériaux
CNRS UMR 7635, B.P. 207
1 Rue Claude Daunesse 06904 Sophia Antipolis cedex
{youssef.mesri, hugues.digonnet, thierry.coupez}@ensmp.fr

ABSTRACT. *This paper presents a fully parallel multi-component Library called CIMLib. CIMLib contains a set of components that allow to build efficiently numerical simulation of a various processes mainly in material forming. We describe in this paper the main components of the library: parallel mesh partitioning, parallel remeshing, the Finite Element modeling and the parallel storage and visualization. Two large numerical simulations are presented: the first one focuses on a multi-bodies contact problem, including friction, for complex 3D forming processes. The mesh is evolving during the simulation from 52K nodes to 7M nodes and 64 cores are used to handle this application. The second simulation concerns the multiphase problems involved in the manufacturing processes of full parts. The simulation is done using 88 processors and the mesh is refined during the simulation the final mesh has over 25M nodes.*

RÉSUMÉ. *Dans cet article on présente certaines des méthodes numériques utilisées pour construire une bibliothèque parallèle appelée CIMLib. CIMLib est constituée d'un ensemble de composants logiciels permettant de réaliser la simulation numérique de procédés. Nous décrivons les composants principaux de la bibliothèque : génération de maillage et remaillage parallèle, solveur éléments finis, stockage des données et visualisation parallèles. Ensuite deux grandes simulations numériques sont présentées : la première représente un problème de contact multicorps récurrent dans la mise en forme des matériaux. La deuxième représente un problème diphasique modélisant le processus de fabrication des pièces pleines. Dans les deux cas, le maillage est raffiné successivement pendant la simulation.*

KEYWORDS: *parallel computation, dynamic load balancing, finite element, material forming, mesh generation, large deformation.*

MOTS-CLÉS : *calcul parallèle, équilibrage dynamique de charge, éléments finis, mise en forme des matériaux, génération de maillage, grandes déformations.*

DOI:10.3166/EJCM.18.669-694 © 2009 Lavoisier, Paris

1. Introduction

During the last decade, we developed a component based library CIMLib including high performance computing (HPC) tools for simulation of complex $3D$ multi-material/fluid and free-surface flow applications (Coupez *et al.*, 1997; Mocellin *et al.*, 2001; Pichelin *et al.*, 1997; Basset *et al.*, 2005). These tools include geometry modeler and mesh generator (Coupez, 2000; Coupez *et al.*, 2000), highly scalable and optimized parallel finite element solver (Coupez *et al.*, 1997; Basset *et al.*, 2005), and using client-server technology for remote visualization (Squillacote, 2005). These HPC tools have provided us a unique capability to simulate applications using unstructured meshes with 200 million elements. Such large simulations generate hundreds of Gigabytes of data. These HPC tools are highly optimized for both memory and parallel scalability especially for such large applications.

One objective of this paper is to highlight the importance of new technologies in design of the massively parallel computing platforms. The model suggested here is the CIMLib library which gathers parallel software components allowing to build powerful applications in material forming.

The organization of this library is based strongly on fundamental choices of the numerical methods implemented that it is necessary to recall. In particular the mixed finite element formulations (stable and stabilized) allow the use of unstructured meshes in 2D as in 3D in fluid as in solid. It is then possible to combine tools for mesh generation and mesh adaptation. The associated linear systems are sufficiently well conditioned to be solved by iterative methods. Anyhow, parallelism will be transparent only if meshing, remeshing, repartitioning are parallel and transparent. These are the key points of an efficient implementation of the library components. It is the principal contribution of the authors in this paper.

A second objective of this paper is to show that a library as CIMLib can associated design pattern. This allows the mutualisation of the developments with the maintenance of good performances.

In this paper, we describe the main components of our library CIMLib. CIMLib is a Multi-components library designed in such a way that the user defines the simulation process as a set of needed components to be executed by a root driver. It is based on a programming interface containing several components that solve specialized problems (like FE equations, meshing the computational domain, contact detection. . .) with a main program that is able to construct and organize the different components to build processes to be simulated like forging, polymer injection, mixing. . .

To illustrate the efficiency of CIMLib, we present two large scale numerical simulations. These consist in:

i- Hot forging process: one involving large deformation within a Lagrangian framework and the other one for flow calculation within an Eulerian formulation of metal material using remeshing stage during the simulation. Here, we use a Lagrangian technique to update the computational domain, the mesh then gets distorted and without remeshing can rapidly degenerate. Remeshing techniques are mandatory for such applications. A parallel meshing/remeshing (Coupez *et al.*, 2000) is used to adapt the mesh during simulation. A $3D$ stabilized mixed finite element solver (as component of CIMLib) is used to solve the nonlinear incompressible stokes equations.

ii- Free-surface flows in complex $3D$ applications. In our approach, the governing equations are the Navier-Stokes equations written for multiple incompressible fluids. We solve these equations over a fixed mesh. A Level Set function is associated with a multi-phase description (Basset *et al.*, 2005). This function is convected throughout the computational domain by scaling a transport equation. Here also, a mixed first order finite element method based on $P1+/P1$ bubble stabilization is used to simulate free-surface flow applications. Moreover, an anisotropic adaptive remeshing can be used (Mesri *et al.*, 2008) to increase the accuracy of the interface capturing.

This paper is organized as follows: Section 2 details the governing partial differential equations that are used in the modeling. Sections 2.1 and 2.2 are devoted to the main applications of this paper, the forging and free surface flow modeling. Section 3 is devoted to the parallel computation framework, which consists in the parallel implementation of all software components. In Section 4 we depict numerical results which exhibit the advantages of the combined parallel component tools to handle accurately complex computational mechanics problems.

2. Modeling framework

CIMLib allows to run a large range of incompressible viscous fluid applications, from viscoelastic to linear elasticity and viscoplasticity. In order to cover and ensure a certain compatibility with the software components, we use a generalized fluid-solid modeling.
The following generalized Stokes equations are used to compute the solid materials and incompressible flows in a cavity Ω; v represents the velocity field and p the pressure:

$$\left\{ \begin{array}{l} -\nabla \cdot (2\eta(|\varepsilon(v)|)\varepsilon(v) + \tau) + \nabla p = 0 \\ \nabla \cdot v = 0 \end{array} \right. \qquad [1]$$

where η and ρ denote respectively the viscosity and the density, and ε is the symmetric gradient operator:

$$\epsilon(.) = \frac{1}{2}(\nabla^T . + \nabla .)$$

and τ is an extra stress tensor field that occurs for much more complex constitutive equations (visco-elasticity, elasto-plasticity...). In the following we will focus on a Stokes equations where τ is neglected.

The following functional spaces are considered for the variational formulation of the Stokes equations:

$$V = (H^1(\Omega))^d, V^0 = (H^1_0(\Omega))^d, \text{ and } P = L^2(\Omega) \qquad [2]$$

where d is the space dimension, $L^2(\Omega) = \{q,\ \int_\Omega q^2\ d\Omega < \infty\}$ and $H^1(\Omega) = \{v \in L^2(\Omega), \nabla v \in (L^2(\Omega))^d\}$.

The variational problem consists in finding $(v, p) \in (V, P)$ such that:

$$\begin{cases} \int_\Omega 2\eta(|\varepsilon(v)|)\varepsilon(v) : \varepsilon(w) - \int_\Omega p\nabla.w = 0 \\ \int_\Omega q.\nabla \cdot v = 0 \end{cases} \qquad [3]$$

for any test functions $(w, q) \in (V, P)$.

Now, we build finite-dimensional spaces V_h and P_h such that $V_h \rightarrow V$ and $P_h \rightarrow P$ when $h \rightarrow 0$. The domain Ω is then decomposed in tetrahedra T: $\Omega = \cup_T T$ so that we can apply a mixed finite element method using the so-called MINI element (P1+/P1) (cf. (Arnold *et al.*, 1983)) with a linear continuous interpolation $P1$ for both pressure velocity and the pressure, and a bubble enrichment of the velocity. This bubble function, which is necessary to satisfay the LBB stability condition (cf. (Fortin *et al.*, 1991)), can be seen as an enrichment of the element by four piecewise linear functions. The solution (v, p) is then approximated by the solution $(v_h, p_h) \in (V_h, P_h)$ for the following problem

$$\begin{cases} \int_\Omega 2\eta(|\varepsilon(\bar{v}_h)|)\varepsilon(\bar{v}_h) : \varepsilon(w_h) + \\ \int_\Omega 2\eta(|\varepsilon(\bar{v}_h)|)\varepsilon(b_h) : \varepsilon(w_h) \ - \ \int_\Omega p_h\nabla.w_h \ = \ 0 \\ \int_\Omega q_h.\nabla \cdot \bar{v}_h \ + \ \int_\Omega q_h.\nabla \cdot b_h \ = \ 0 \end{cases} \qquad [4]$$

where $(w_h, q_h) \in (V_h, P_h)$ and $v_h = \bar{v}_h + b_h$ is decomposed in *resolvable* scales $(\bar{v}_h)$ and *unresolvable* scales (b_h) that are represented as bubbles.

We notice that the MINI element is stable in the sense of the Brezzi Babuska conditions. It is close to the stabilized mixed method, the stabilization operator being obtained by a static condensation of a bubble term inside each element. A very important advantage of this approach is the possibility to use an iterative solver for linear system solution. It is based on a GMRES method (Generalized Minimal Residual method) and it shows to be extremely efficient for solving generalized incompressible Stokes flows. Note that today, every linear system are solved using a parallel iterative method. The library used here is the PetSc toolkit that offers a large number of capabilities.

Bubbles can play a direct role in stabilizing the Stokes equations by means of a multiscale approach (Hughes, 1995). Thus, the *resolvable* scales $(\bar{v}_h)$ are distinguished from the *unresolvable* scales (b_h) that are, in fact, represented as bubbles.

We present in the following sections, two different problems to be modeled with the equations [1]. We give also, with respect to each problem, the boundary conditions that is needed to close the PDE system [1].

2.1. *Forging modeling*

2.1.1. *Continuous problem*

In this section, we adopt the flow formulation which is quite natural in viscoplasticity and covers a large range of applications. This formulation can be extended to elastoplastic calculations as it has been shown in reference (Gay *et al.*, 1994). However, under hot forging conditions, the elastic deformations can be neglected, thus assuming the material to be homogeneous, isotropic and incompressible. The resulting nonlinear partial differential equations are described in [1].

The velocity v and the pressure p are computed upon a time-dependent domain $\Omega = \Omega(t)$ of boundary $\partial\Omega$. The viscoplastic behavior is modelled by a viscosity which is a function of the strain rate tensor $\epsilon(v)$. The following power law, known as the Norton-Hoff law, is considered here: $\eta(|\epsilon(v)|) = K(\sqrt{3}|\epsilon(v)|)^{m-1}$ where K is the consistency of the material and m is the strain rate sensitivity coefficient that ranges between 0 and 1. Boundary conditions [5] on the contact surface with the forming tools are added, assuming that the friction is governed by Norton law

$$\begin{cases} (v - v_{tool}).n - \frac{\delta}{\Delta t} \leq 0 \\ \tau = -fK||\Delta v_t||^{p-1}\Delta v_t \end{cases} \qquad [5]$$

where f is the coefficient of the friction law, v_{tool} is the tool velocity, n is the outward normal, δ is the distance to the tool surface, Δt is the time step and Δv_t is the relative tangential velocity. Contact condition is an equality constraint. We remark that inequality can be converted into equality by:

$$(v - v_{tool}).n - \frac{\delta}{\Delta t} \leq 0 <=> [(v - v_{tool}).n - \frac{\delta}{\Delta t}]^{+} = 0 \qquad [6]$$

where $[x]^{+}$ is the positive part of the quantity x. This equality constraint is then imposed by penalty formulation. Therefore, a mixed velocity/pressure formulation is used for the variational form of the equations [7], where v is a kinematically admissible velocity field that satisfies the contact conditions [5] for any test functions (w, q) in $V \times P$.

$$\left\{\begin{array}{l} \int_\Omega 2\eta(|\epsilon(v)|)\epsilon(v) : \epsilon(w) d\Omega - \int_\Omega p\nabla.w d\Omega + \\ r\int_{\partial\Omega} \lambda w.n d\partial\Omega + \int_{\partial\Omega} f||\Delta v_t||^{p-1}\Delta v_t w \partial\Omega = 0 \\ \int_\Omega q\nabla.v d\Omega = 0 \\ \lambda = [(v - v_{tool}).n - \frac{\delta}{\Delta t}]^+ \end{array}\right. \quad [7]$$

where r denotes the homogeneous penalty coefficient $(r >> 1)$ used to impose the contact constraint. An Euler explicit scheme is used for time integration and the spatial discretization is based on tetrahedral elements with a P1+/P1 interpolation. The contact condition [5] is handled by a penalty method. The resulting non linear equations are solved by a Newton-Raphson algorithm and the linear system by a Preconditioned Conjugate Residual algorithm. The contact between deformable bodies, i.e. the workpiece and the forming tools, is treated using a master-slave algorithm as presented in (Bruchon *et al.*, 2009; Pichelin *et al.*, 2001).

2.1.2. *Contact treatment*

At each time step, a search for nodes that are potentially going to penetrate the opposite surface is performed. A penalty contribution, based on the penetration distance, is added to the functional for these nodes. The contact terms arising from contact between different bodies are computed with a coupled approach based on a master-slave algorithm which deals with contact, friction and thermal conduction terms. The contact algorithm will ensure that slave nodes will not penetrate into master faces. The choice of the slave and master surfaces is based on two main rules: - when the mechanical behaviors of the two opposite bodies are different, the master surface is the more rigid; - when the mechanical behaviors of the two opposite bodies are the same, it is better to have a finer mesh on the slave surface in order to minimize penetration. The non-penetration condition will be written between a slave node and a master triangular face (introduction of fictitious elements) in the current configuration. At the beginning of each time increment, a search algorithm is performed to build the node-face contact pairs which determine for all slave boundary nodes the closest master face. More details can be found in (Pichelin *et al.*, 2001).

2.2. ***Multi-phase modeling***

2.2.1. *Interface capture*

This paragraph shows a way to couple the Stokes Equations [1] with the scalar advection equation in order to simulate multi-phase flows. The issue here is to capture effectively the interfaces between phases. To this end, a regular Level-Set function α is used to approximate the interface Γ between different phases (cf. (Sethian, 1986;

Osher *et al.*, 2001)). we initialize α as a signed distance function to the interface $\alpha(x) = dist(x, \Gamma)$:

$$\begin{cases} \alpha > 0 \text{ in the first phase} \\ \alpha < 0 \text{ in the second phase} \\ \alpha = 0 \text{ in the interface} \end{cases} \qquad [8]$$

Then, when this function is transported during the simulation, the isosurface zero of α is kept to be the fluid interface. Moreover, thanks to the fact that α is signed, the different fluids in the flow can be located by using an appropriate mixture law presented in the next section. For the sake of simplicity, this paper only consider a Newtonian $(\eta(|\varepsilon(v)|) = \eta = Cte)$ bi-fluid flows, but the presented methods can easily be extended to a higher number of different fluids present in the flow.

2.2.2. *A Mixture law for multi-phase modeling*

Since we have several phases and only one flow solver, we need to turn the different viscosities and densities into homogeneous parameters to use during resolution of the Stokes system. Thus, based on a mixture law, we make η and ρ depend on the characteristics of each fluid: for a simulation involving a first fluid with viscosity η_1 and density ρ_1 , and a second fluid with viscosity η_2 and density ρ_2 , we define the parameters η and ρ in the Stokes equations as the following:

$$\eta = \eta_1 H(\alpha) + \eta_2(1 - H(\alpha))$$
$$\rho = \rho_1 H(\alpha) + \rho_2(1 - H(\alpha))$$
$$H(\alpha) = \begin{cases} 1 \text{ if } \alpha > 0 \\ 0 \text{ if } \alpha < 0 \end{cases} \qquad [9]$$

where H is the Heaviside function. With such definition, we have $\eta = \eta_1$ and $\rho = \rho_1$ in the first phase; $\eta = \eta_2$ and $\rho = \rho_2$ in the second phase; and intermediate values given inside a mixture zone.

We will see that this buffer zone where the two fluids coexist at the same time can be as small as the element size. By this way, the diffusivity goes no larger than the mesh size all along the simulation. $H(\alpha)$ can be straight or not, abrupt or gradual, etc...; and that determines the nature and the size of the buffer zone. Here, a smooth transition on a certain thickness is considered and is given by:

$$H_e(\alpha) = \begin{cases} 1 \text{ if } \alpha > e \\ \frac{1}{2} + \frac{\alpha}{2e} \text{ if } \alpha \in [-e, e] \\ 0 \text{ if } \alpha < -e \end{cases} \qquad [10]$$

where e is an element size. Such a regularization is used to control the thickness of the mixture zone. Actually, it allows to determine the amount of all fluids present inside elements crossed by the interface; while other elements are fully filled by only one fluid. A big benefit would be that the zone where several fluids coexist is totally controlled, and cannot grow larger than the element size e.

2.2.3. *Transport equation*

Usually, the motion of the interface is driven by a velocity field v related to the differential properties of the manifold. This leads to a pure advection scalar equation that is well suited to compute moving liquid when the complex interface shape varies quickly at very large amplitude:

$$\frac{d\alpha}{dt} = \frac{\partial \alpha}{\partial t} + v.\nabla\alpha = 0 \quad \in \Omega \tag{11}$$

The variational problem consists in finding $\alpha \in H^1(\Omega)$ with $\varphi \in H_0^1(\Omega)$ such that:

$$\int_\Omega \frac{\partial \alpha}{\partial t}.\varphi + \int_\Omega \nabla\alpha.v.\varphi = 0 \tag{12}$$

With the same discretization as for the Stokes equations, α is approached by the solution α_h of this problem

$$\int_\Omega \frac{\partial \alpha_h}{\partial t}.\varphi_h + \int_\Omega \nabla\alpha_h.v_h.\varphi_h = 0 \tag{13}$$

While finite element methods (and the classical Galerkin formulation naturally associated with them) are well suited for solving incompressible fluid flows, they are somehow inappropriate for purely convective problems. The reason is that the centered differencing property of the standard Galerkin method causes spurious oscillations in the solution when the problem is not dominated by diffusion, but by advection. First, we observe that a standard Galerkin method gives much better results (and less oscillations in the solution) when the function α to be transported is initialized as a Level-Set function; i.e. a distance function to the fluid interface. Second, we can apply stabilization by keeping the bubble and multiscale philosophy (Hughes, 1995), as for the Stokes equations. This way, the method like the Residual-Free Bubbles (Brezzi *et al.*, 1994; Brezzi *et al.*, 1998) provides us the wanted stabilization for the advection problem:

$$\int_\Omega \frac{\partial \alpha_h}{\partial t}.\varphi_h + \int_\Omega \nabla\alpha_h.v_h.\varphi_h + \int_\Omega \Big(\frac{\partial \alpha_h}{\partial t} + \nabla\alpha_h.v_h\Big).\tau_a.\Big(\frac{\partial \varphi_h}{\partial t} + \nabla\varphi_h.v_h\Big) = 0 \tag{14}$$

where $\tau_a = 1/3\frac{h_T^v}{|v_T|}$, $|v_T|$ is the average velocity norm in the element T, and h_T^v is the length of the longest segment parallel to v_T and contained in T.

2.2.4. *Coupled formulation*

The two models we seek to couple are strongly dependent, so the coupling is done in this order: we solve the (v, p) system, when we get v we then compute α. α is used to update ρ and η. Consequently, the resulting coupled system contains five unknowns per node of the mesh cavity (in three dimension). The coupled systems are defined in the whole domain Ω, the test functions of the corresponding weak formulations are required to vanish at the boundary of the domain, but not at the interface of the fluids.

3. Parallel computation framework

Our demonstration application library, CIMLib (Digonnet *et al.*, 2003), is an unstructured mesh, parallel finite element code developed by the CIM group of the Ecole des Mines Center for material forming (CEMEF). CIMLib uses the single program multiple data (SPMD) message-passing programming model. Since, the mesh partitioning has a dominant effect on parallel scalability, CIMLib contains a mesh partitioning/repartitioning algorithm called MeshMigration, that allows to balance well the number of mesh entities (vertices or elements) per processor (see (Digonnet, 2001)). MeshMigration is extended to heterogeneous architectures by taking into account the parallel machine characteristics (CPU and network) (see (Mesri *et al.*, 2005)).

The local matrices and second members resulting from the implicit finite element formulations are assembled and solved by using PETSc library. We used an ILU preconditioning and a Generalized Minimal Residual (GMRES) resolution method.

CIMLib is a Multi-components library designed in such a way that the user defines the simulation process as a set of needed components to be executed by a root driver. It is based on a programming interface containing several components that solve specialized problems (like FE equations, meshing the computational domain, contact detection . . .) with a main program that is able to construct and organize the different components to build the process we want to simulate. Thus, we can define CIMLib as:

- a set of components, each of which allows solving one particular problem like remeshing, boundary conditions. . .

- an engine that organizes the components and therefore allows to simulate a particular process like forging, polymer injection . . .

Some functional components that can be interpreted like the classical instructions present in every language. By instruction, we mean "block", "if then else", "while" . . . Others focus on numerical simulations: mesh management, solving FE problems with large linear/non linear systems. All these components are provided to help the user in building/modeling the process he wants to simulate and not dedicated to a specific process. We take particular care to keep a very good upgradeability: it is easy to add a new component, but also by implementing components that can be easily derived to solve a large number of problems.

It is important to enable the component organization to exchange information. For example the velocity, computed by solving the Navier-Stokes equation, needs to be used as a parameter for the transport equation. To make possible such exchange, we introduce the notion of "Field". By Field we consider common results like velocity, temperature, stress which are represented over the mesh, but also some scalar values like the global volume of the piece. The Field interface is then derived into several interpolation Fields: from the P0 interpolation, a constant value over the whole domain, to the discontinuous P0 interpolation and continuous P1 interpolation. Field is the main object used to interact between components. One field computed in one component can be used by another one as a parameter. The large majority of the components present into CIMLib takes as input some parameter Fields and has also some Fields as output. These Fields are the main object to exchange information between components, even if there are other objects. For example, a "Geometer" can be used in a particular component to compute the distance to a particular form.

3.1. *Mesh partitioning component*

3.1.1. *Workload model*

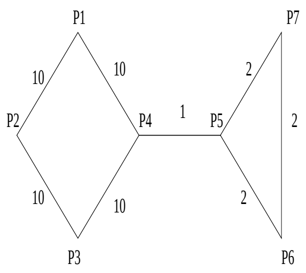
In this work based on the previous work of ((Basermann *et al.*, 2000)), we consider the combined (dual and nodal) graph for modeling the Finite Elements applications. We represent the application as a weighted undirected graph $W = (V(W), E(W))$, which we will call the *workload graph* . Each vertex v has a computational weight $\omega(v)$, which reflects the amount of the computation to be done at v. An edge between vertices u and v, denoted $\{u, v\}$, has a computational weight $\omega(\{u, v\})$, which reflects the data dependency between them.

3.1.2. *Heterogeneous architecture model*

Partitioning applications onto heterogeneous architecture such as a Grid environment requires a special model architecture that reflects both heterogeneous resource characteristics and also non-homogeneous communication network between these different resources.

The machine architecture can be represented as a weighted undirected graph $A = (V(A), E(A))$, which we will call the architecture graph. It consists in a set of vertices $V(A) = \{p_1, p_2, ..., p_n\}$, denoting processors, and a set of edges $E(A) = \{\{p_i, p_j\}|p_i, p_j \in P\}$, representing communication links between processors. Each processor p has a processing weight s_p, modeling its processing power per unit of computation. Each link has a link weight v_{pq}, that denotes the communication bandwidth per unit of communication between processors p and q.

In this work, we assume that the machine architecture can be represented by a complete graph: given any two processors p and q, there always exist a path connecting them even if they are not directly connected by a physical link. In this case, the weight of the edge $\{p, q\}$ will be evaluated as the minimum of the link weights on the shortest path connecting p and q. The communication matrix in the Figure 1 gives for instance the communication weights matrix corresponding to the adjacent architecture graph.



$$C = \begin{pmatrix} \times & 10 & 10 & 10 & 1 & 1 & 1 \\ 10 & \times & 10 & 10 & 1 & 1 & 1 \\ 10 & 10 & \times & 10 & 1 & 1 & 1 \\ 10 & 10 & 10 & \times & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & \times & 2 & 2 \\ 1 & 1 & 1 & 1 & 2 & \times & 2 \\ 1 & 1 & 1 & 1 & 2 & 2 & \times \end{pmatrix}$$

Architecture graph Communication Matrix

Figure 1. *Heterogeneous graph architecture and the corresponding matrix of communication*

3.1.3. *Mesh partitioning model*

We consider a workload graph $W(V(W), E(W))$ which represents the application, and a architecture graph $A(V(A), E(A))$ which represents the parallel machine. The machine architecture is heterogeneous both for network and processors. So we consider the characteristics of architecture graph to define the partitions. A mapping of a workload graph onto a architecture graph can be formally described by:

$$m \,:\, V(W) \longrightarrow V(A) \qquad [15]$$

where $m(v) = p$, if the vertex v of W is assigned to processor p of A.
In order to evaluate the quality of a mapping, we define two cost models: one for estimating the computational cost and the other one for the communication cost evaluation.

3.1.3.1. Computational cost

For each mapping of the workload graph onto the architecture graph we can estimate the computational cost as follows: If a vertex v is assigned to a processor p, the computational cost is given by $t_p^v = \omega(v)/s_p$, that is the ratio of the computational weight of v per the processing weight of p. Computational cost estimates the time units required by p to process v.

3.1.3.2. Communication cost

The communication cost is introduced when we have a data communication transfer between two different nodes in the target graph. Suppose that $\{u, v\} \in E(W)$ and $u \in V(W)$ is assigned to processor p and $v \in V(W)$ is assigned to processor q. The data is transferred from the local memory of p to the local memory of q via message passing. In this case, the communication cost is given by $c_{pq}^{u,v} = \omega(\{u, v\})/v_{pq}$, that is the ratio of the communication weight of edge $\{u, v\}$ per the link weight between p and q. The communication cost represents the time units required for data transfer between the vertices u and v.

3.1.3.3. Cost function

Let $m \ : \ V(W) \longrightarrow V(A)$ be a mapping of $W(V)$ onto $V(A)$, the weight of subgraph assigned to a processor p in $V(A)$, is the sum of the weights of the vertices in the subgraph: $C(p, m) = \sum_{v \in V(W), m(v)=p} \omega(v)$. For all p in $V(A)$, the computational time is given by $t_p = \frac{C(p,m)}{s_p}$, where $C(p, m)$ is defined above and s_p is the processing weight.

Let $\{p.q\} \in E(A)$, we define the communication cost associated to the processors p and q as:

$$\mathcal{C}(\{p, q\}, m) = \sum_{\substack{m(u)=p \\ m(v)=q \\ \{u,v\} \in E(W)}} c_{pq}^{u,v}$$

The total communication time associated to processor p is defined by:

$\mathcal{C}(p, m) = \sum_{q \in V(A) \neq p} \mathcal{C}(\{p, q\}, m)$. To evaluate the quality of the mapping, we define a cost function as follows: $F(W, A, m) := T + C$ where $T = {}^t\big(t_1, \ldots, t_{card(V(A))}\big)$ and $C = {}^t\big(\mathcal{C}(1, m), \ldots, \mathcal{C}(card(V(A)), m)\big)$. The definition of the graph partitioning problem is to find a partition (mapping m) which minimizes the cost function $F(W, A, m)$. Clearly, the problem is extensible to the classical graph partitioning and task assignment problem, and it is well known that this problem is NP-complete. In the next section, we describe the iterative algorithm chosen to minimize this cost function and find the efficient partitioning.

3.1.4. *Parallel partitioning/repartitioning algorithm*

MeshMigration is a parallel graph/mesh repartitioning scheme developed for heterogeneous architectures such as the Grid. We employ a local method of parallel repartitioning developed during the DRAMA project (Basermann *et al.*, 2000).

The principal steps of this strategy are the following:

- Form disjoint pairs of processors that will present an important gain for the cost function (see section 3.1.4.2).

- Optimize the mapping on each formed pair: This optimization is performed by transferring vertices (elements and nodes) from one processor to another by using the notion of strip migration (see the last part of the section 3.1.4.2). The definition of strips to be migrated is given in the section 3.1.4.1.

- The two previous steps are iterated as long as we are able to globally optimize the partition.

3.1.4.1. Strip migration

Let (p,q) be two processors and let $\mathcal{I}_{p.q} = \{\{u,v\} \in E(W)/m(u) = p \ \text{ and } \ m(v) = q\}$ be the interface between the processors p and q. For any w such that $m(w) = p$ (resp. q) the topological distance (denoted $dist$) of w from the interface is defined as the shortest path between w and a vertex of the interface in the nodal mesh graph (if w is a node of the mesh) or in the dual mesh graph (if w is an element). We then define, a strip as the set of nodes and elements that have the same topological distance from the interface. The optimization of the partition is then performed by transferring strips from processor p to processor q in order to increase topological distance as long as this transfer improves the cost function.

3.1.4.2. Formation of processor pairs

The goal of this algorithm is to perform a parallel and automatic coupling of processors. It provides the maximum number of pairs of processors which consent to optimize the partitions.

If we consider a pair of processors (p,q), the cost function of initial partition between p and q is given by: $F_0|_{pq} = max(t_p, t_q) + \mathcal{C}(\{p,q\}, m)$ where $\mathcal{C}(\{p,q\}, m) = \mathcal{C}(\{q,p\}, m)$. To improve the initial partition, we evaluate the cost function strip per strip as long as we find the best strip associated to the minimum of the cost function on p, denoted F^p_{min} and on q, denoted F^q_{min}. On a processor p, the cost function is evaluated at every strip s as follows:

$$F^p(s) = max(t'_p, t'_q) + \mathcal{C}'(\{p,q\}, m) \qquad [16]$$

where

$$t'_p = t_p - \frac{C'(p,m)}{s_p}, \qquad t'_q = t_q + \frac{C'(p,m)}{s_q}$$

$$C'(p,m) = \sum_{\substack{v \in V(W) \\ dist(v) \leq dist(s)}} w(v)$$

$$\mathcal{C}'(p,m) = \sum_{\substack{dist(s) \leq dist(u) < dist(s+1) \\ dist(s-1) < dist(v) \leq dist(s)}} C^{u,v}_{p,q}$$

Then, we define a Friendship function between the processors p and q which is given by the maximal potential gain:

$$\textit{Friendship}(p,q) = max\Big(F_0|_{pq} - F^p_{\text{min}}, F_0|_{pq} - F^q_{\text{min}}\Big) \qquad [17]$$

The first pair of processors formed is given by:
$\textit{Friendship}(p,q) = max(\textit{Friendship}(i,j))$, for all i and j in $V(A)$.

The migration between p and q is determined as follows:
if $(F_0|_{pq} - F^p_{\text{min}}) > (F_0|_{pq} - F^q_{\text{min}})$ (resp. $(F_0|_{pq} - F^p_{\text{min}}) < (F_0|_{pq} - F^q_{\text{min}})$), the elements and nodes having a distance lower than the distance of the best strip associated to F^p_{min} (resp. F^q_{min}) are migrated from p to q (resp. from q to p).

3.2. *Parallel mesh generation*

In this section we review a parallel mesh generation and adaption procedure which is based on a topological mesh generator. In the next section (3.2.1) we describe briefly the sequential mesh generator MTC and further (section 3.2.2) we describe the strategy adopted to parallelize this mesh generator.

3.2.1. *Serial remeshing*

MTC is a mesh generator developed by Thierry Coupez at Ecole des Mines de Paris, Center for Material Forming, Sophia Antipolis. It is based on the idea to improve iteratively, an initial unsatisfactory mesh by local improvements.

MTC mesh generator re-meshes the initial mesh iteratively by a local mesh optimization technique. The mesh optimization technique consists in local re-meshing of cavities formed by small clusters of elements in order to increase the “quality” of the elements of the cluster.

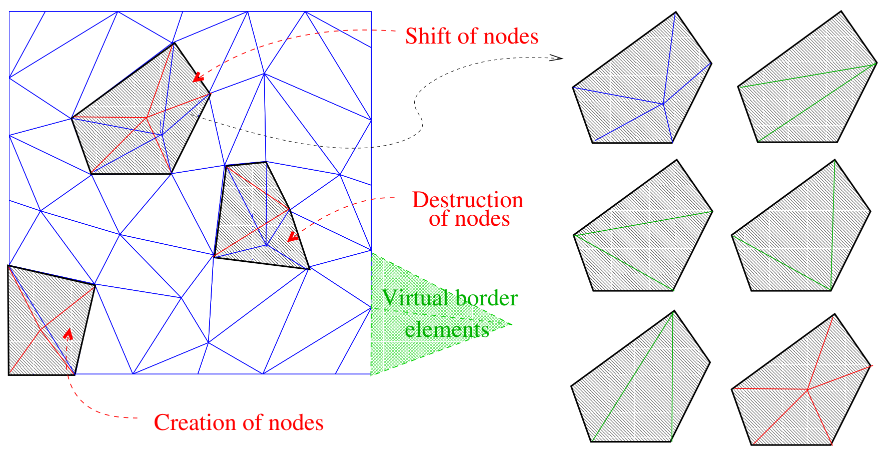


Figure 2. *Local mesh optimization process in MTC*

In the re-meshing process, two principles are enforced :

– *Minimal volume*, which assures the conformity of the mesh, with no overlaps of elements: let $\mathcal{T}_i(Cavity)$ denote the i-th set of elements T filling the local cavity. Following the minimum volume principle we choose as an optimal (possibly not unique) re-triangulation of the cavity the one satisfying

$$\sum_{T \in \mathcal{T}_i(Cavity)} |(Volume)(T))| \rightarrow \min, \qquad [18]$$

where the minimization is done over a small set $i = \{1, \ldots, I\}$ of possible triangulations $\mathcal{T}_i(Cavity)$ of elements (Fig. 2 right) connecting the nodes on the border of the cavity, or other nodes like the cavity barycenter, with all boundary faces.

– *The geometrical quality* $Q(T)$, which is evaluated for each element. If the minimum of [18] is not unique, this criterion picks among all admissible cavity re-triangulations the one improving the geometrical quality of the mesh by improving the quality of the worst element of the triangulation.

While the former criterion assures the conformity of the mesh, if the initial mesh was conforming, the latter handles improvements of element shape, size, connectivity, etc., depending on the quality function $Q(T)$. Usually, the quality function $Q(T)$ is a function of the geometry of the element T and the prescribed background metric, which give together a measure for the element size and the element form (aspect ratio). For further details see (Coupez, 1994; Coupez, 2000).

3.2.2. *Parallel remeshing*

Parallelism of the mesh generator MTC is performed by partitioning/repartitioning the initial mesh into submeshes, the individual submeshes are refined/derefined or adapted to an error estimator by using the serial MTC. In order to keep a global mesh correctness, the interface faces between the submeshes should be subdivided the same way. To perform this efficiently in the parallel context, we have chosen to keep interfaces unchanged during the local (inside each subdomain) re-meshing. Which avoid any communication between processors during the remeshing step. The blocking interface chose introduce therefore a new constraint in the iterative remeshing process (elements near the interfaces are not remeshed). To obtain a satisfactory final mesh regarding the quality function, we perform a repartitioning step to move the interface inside the domain in order to enable re-meshing in a next phase. We notice that a few iterations (three to five depending of the space dimension) between remeshing and repartitioning stages could be necessary to build the optimal mesh. The time spent in the remeshing and the repartitioning per iteration decreases drastically as there are less and less elements to move and remesh. The partitioning/re-partitioning of a mesh is performed in parallel using MeshMigration tool that is described in the section 3.1. Figure (3) shows this strategy that is applied to a simple $2D$ mesh with 7 submeshes partitioned onto 7 processors. The process depicted in the figure (3) consists in a parallel isotropic refinement of an initial unstructured $2D$ mesh. We depict the mesh partition and the zones to remesh during the iterative remeshing procedure. At the first iteration, all mesh domain has to be remeshed and step by step the zones to remesh become small and small and then the remeshing time decreases drastically.

A preliminary version of this work is presented in (Coupez *et al.*, 2000). It handles large scale simulation by removing any sequential part of the code and has also reduce by around 20% the simulation run time. Parallel performance analysis presented in (Digonnet *et al.*, 2007) show reasonable but not optimal speed-up (six over eight processors) when looking only at the remeshing stage. In this paper we focus on parallel efficiency to overcome some bottleneck algorithmic. To this end, we have worked to fully take advantage of the local procedure optimization used in MTC and DRAMA. A new repartitioning method have been implemented to move efficiently the interfaces. A strict extraction of local zones that need to be remeshed is performed using a permutation technique (see figure (3)). This local extraction method allows to reduce the complexity of the algorithm from $O(N_p)$ to $O(N_z)$ where N_p denotes the size of a processor submesh and N_z denotes the size of local zone in the processor submesh with $N_p >> N_z$.

We illustrate the parallel efficiency of the new development by some test cases (isotropic refinement in $2D$ and $3D$). Table (1) shows the parallel remeshing speed-up of 1 to 32 processors. The speed-up realized is closed to the optimal for $2D$ and $3D$ test-cases. All these implementations are included as a component of the CIMLib library in order to perform large scale simulations in material forming processing. Complete simulations using a large number of processors are also done and analyzed in the following sections.

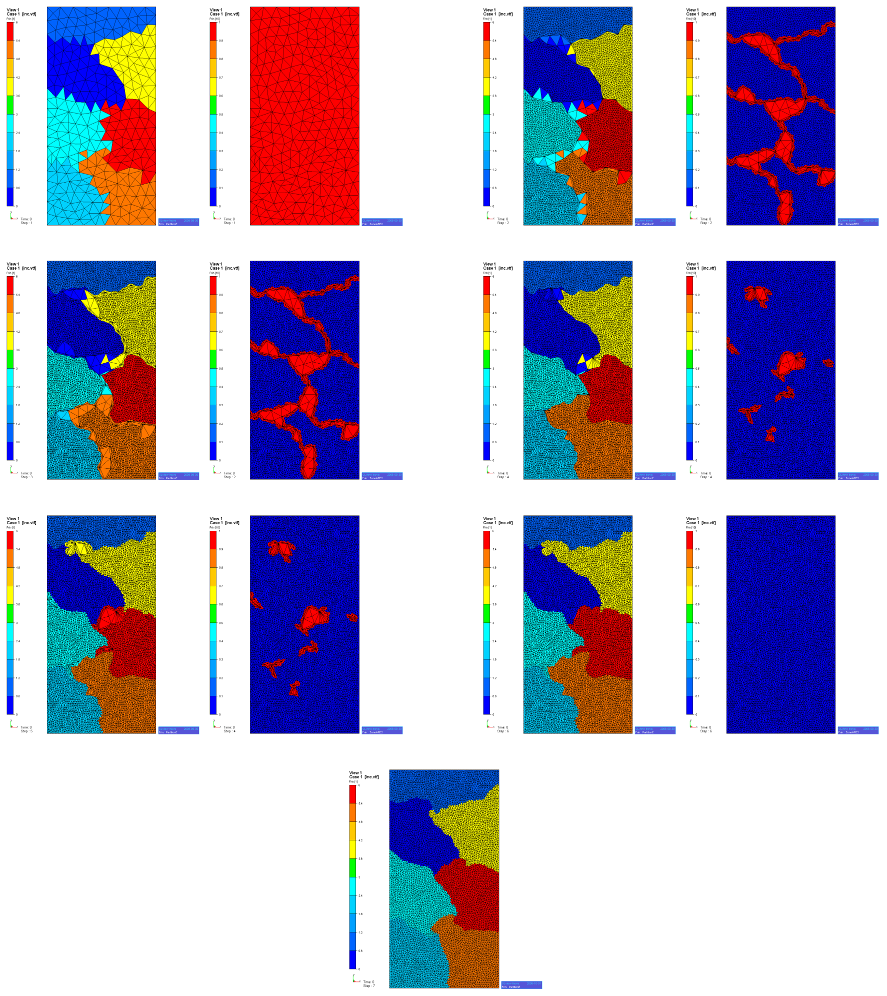


Figure 3. *Illustration of the strategy used to parallelize the mesh generator. From the top to the bottom we show the successive steps of parallel repartitioning and parallel re-meshing by keeping interfaces unchanged*

3.3. *Finite element component*

A finite element component is used to determine the solution fields of PDEs using the finite element method. To solve such equations, we have to compute, store and solve large linear/non linear systems. Non linear systems being solved using Newton-Raphson method lead to the resolution of several linear systems. Here again, we must take care of hiding parallel instructions in the part of the code where we are

Table 1. *Parallel remeshing speed up of 1 to 32 processors obtained onto* $2D$ *and* $3D$ *meshes*

CPUs	Remeshing (s)	Speed up
1	2391.34	1
2	1097.32	2.17
4	510.201	4.68
8	269.179	8.88
16	119.735	19.97
32	68.423	34.94

$2D$ mesh test-case

CPUs	Remeshing (s)	Speed up
1	3079.98	1
2	1443.76	2.13
4	753.538	4.087
8	385.474	7.99
16	196.179	17.48
32	112.045	27.48

$3D$ mesh test-case

planning to make most future developments (Digonnet *et al.*, 2003). For that, we have chosen to use the PETSc library (McInnes *et al.*, 1995) that provides functions to store and solve large systems in parallel. We have made a clear separation between the storage-solving linear system which use PETSc and the way to compute local contributions at the element level. As soon as the Object PETSc can deal with solving symmetrical/non symmetrical systems with a free number of unknowns located at each element or node of the mesh, we developed an interface that computes local contributions of a problem. By using this, common developer only needs to implement the FE model of his physical problem at the element level in a sequential way and can use it in parallel.

3.4. *Parallel visualization component*

The computational meshes that we use contain several millions of points. For instance, the largest mesh we deal with uses up to 25 millions mesh points to derive a good solution. Visualizing the solution data from this type of calculation is particularly challenging because the associated unstructured meshes are typically large in size and irregular in both shape and resolution. Moreover, calculations are done in parallel and solutions are also stored in a distributed form, then it is efficient to visualize the parallel solution directly without assembling it in a serial form. For this purpose, we use the parallel visualization tool ParaView [1], to render in parallel the unstructured-mesh data.

1. http://www.paraview.org

4. Numerical results

4.1. *Performance analysis*

In this section, we present computations performed on the CEMEF's cluster. The presented case was chosen to evaluate the parallel performance of CIMLib. It regards a linear Stokes system. The linear stokes system is derived from [1] by assuming a constant viscosity η and ommitting the time-dependent term and the second member. The mesh used contains 1 million of nodes. The parallel simulation was running onto 1 to 256 processors in order to study the speed-up. The parallel machine (cluster) used here is heterogeneous and is constituted by 512 Opteron cores (dual cores with 2.4 Ghz, 2Gb memory, 1Gb of cache memory and quadri-core with 2.3 Ghz, 2Gb memory and 512Mo of cache memory) linked by an infiniband network.

Table (2) presents the computational time in seconds, as well as the Speed-Up measured during such a simulation. For one CPU, we depict the runtime over two different nodes to highlight the heterogeneity in CPUs of the cluster.The table shows the number of iterative iteration to solve the linear system, the time spent in assembling and resolution, the time spent in computing the symmetric gradient operator $\epsilon(v) = 1/2(\nabla.v + \nabla^t.v)$ and then the speed-up with respect to both sequential reference time (ref1 and ref2).

Regarding the resolution of the FE problem, the time spent is divided by the number of used processors in spite of the heterogeneity of the parallel machine. However, the speed-up is mainly affected by the heterogeneity of resources.

Table 2. *Time profiling for the Stokes equations*

CPUs	NbIter	Assembling (s)	Resolution (s)	$\epsilon(v)$ (s)	Speed-up
1 (ref1)	579	114.5037	2899.3799	18.9484	1
1 (ref2)	579	134.7185	2417.4146	16.4948	1
2	600	58.5123	1095.7660	9.2421	2.2/2.64
4	608	29.8496	509.9427	4.6028	4.74/5.68
8	623	14.7492	311.4674	2.4108	7.76/9.3
16	620	7.6469	176.7225	1.1869	13.76/16.4
32	639	4.2451	138.0940	0.5606	17.5/20.99
64	626	2.0935	63.8948	0.2852	37.83/45.38
256	818	0.5123	18.1372	0.0624	133.33/158.92

4.2. *Forming processes*

The geometry of the problem is composed of three bodies as depicted in figure 4. A constant vertical velocity is imposed on the top of the upper die Ω_2. No displacement is imposed to the lower die Ω_3. At the interface of the bodies, the contact is supposed to be sliding with a viscoplastic friction law defined by $\alpha = 0.3$ and $q = 1$. Other boundaries are free surfaces. We choose a consistency of $K = 179.2MPa$ and a strain rate sensitivity $m = 0.3$ for the intermediate block. The lower and upper dies have a consistency one hundred times higher. These values are representative of hot forging of steel. The intermediate body is the slave body and the upper and lower dies are master bodies. In our calculation, the slave mesh is always finer than the master one. Furthermore, the interface meshes do not coincide. We notice that the mesh is deformed during the process and must be adapted continuously in order to avoid distorsion and to obtain a good description of the contact zone. In this calculation, the remeshing strategy described in 3.2 is used periodically in order to adapt the mesh to the contact evolution. The required average number of Newton-Raphson iterations for convergence is six. The convergence test requires a relative convergence of 10^{-6} on the residual norm. This residual takes into account the contact terms.

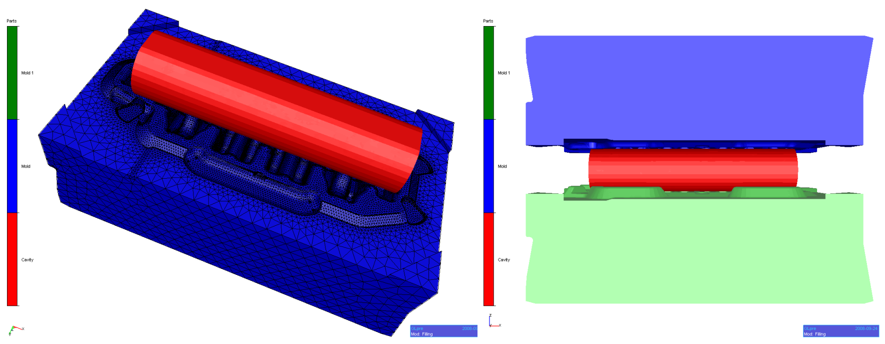


Figure 4. *Contact configuration: upper die, workpiece and lower die*

Computations are performed on 64 cores of the CEMEF's cluster. This cluster is constituted by 512 Opteron cores (dual cores with 2.4 Ghz, 2Gb memory, 1Gb of cache memory and quadri-core with 2.3 Ghz, 2Gb memory and 512Mo of cache memory) linked by an infiniband network.

At each increment we perform, a parallel remeshing stage (including metric computation and field transport from the old mesh to the new mesh), a non-linear and a linear systems computation and resolution. Therefore, we should depict both: the parallel remeshing and the parallel FE solver. An initial workpiece mesh with around 52354 nodes and 33000, 32000 nodes respectively for the upperdie and lowerdie meshes are chosen to perform these computations. The two dies meshes are fixed during computations since the workpiece mesh is evolving and it reaches 5 millions nodes at the 120 increment.

Table 3 presents the computational time in seconds for different increments. We show at every 20th increment, the number of mesh nodes (in million), the number of non linear iterations, the time spent for solving that includes resolution and assembling time, remeshing which represents parallel remeshing and field transport and finally the time spending in writing output files. The number of mesh nodes increases as the deformation of the part progresses. That is due to the successive refinement of the deformed regions of the part and explains the increase in the number of points of the grid from an increment to another in Table 3. Moreover, one can observe that the nonlinear iteration count obviously influence the solution time. A high number of nonlinear iterations increases the solution time. The difference of the nonlinear iteration count of an increment to another reflects the difficulty encountered with the resolution of the contact problem that is strongly nonlinear. Figure 5 at the top-left

Table 3. *Time instrumentation for the forging simulation*

Increments	20th	40th	60th	80th	100th	120th
# of nodes	1.3M	1.89M	2.46M	3.29M	4.09M	5.1M
NL iterations	277	210	161	223	260	210
Solver (s)	1234.16	2276.60	2310.78	7350.31	9120.07	10854.37
Remeshing (s)	1356.72	1752.38	2005.73	5414.71	6617.85	8257.44
Outputs (s)	392.11	665.22	958.32	1208.87	1631.53	2511.60
Sum (s)	2982.99	4694.20	5274.83	13973.89	17369.45	21623.41

shows the evolution of the deformed body from the initial shape to the one obtained at increment 120. In the top-right of figure (5), the velocity magnitude is high in the upper and low in the lower parts. Figure (5) at the bottom-left shows the distance of the body to the tools and at the bottom-right we show the viscosity field.

4.3. *Injection process simulation*

The presented case clearly illustrates the capabilities of parallel computation. Such a test is challenging and needs a full parallelization of all parts of the code. The visualization is also done in parallel by using ParaView tool.

This case is run on the old CEMEF's cluster, which is composed by 24 bi-processors bi-cores AMD Opteron 280 with 8 GB of RAM. The network is Infini-Band. To calibrate the test, we first need to think how to determine the limits of such architecture by taking into account the global RAM. Here, every processor (core) has 2 GB that corresponds solving linear systems with around 1.2 million unknowns (after estimation). This results into 115 millions of unknowns for the cluster. As this cluster is not used by a single person and that some nodes could be out of order, we have chosen a test case that requires solving systems with 100 millions unknowns. The largest system is for solving the Stokes equations, using the mini-element P1+/P1 in velocity and pressure. In a $3D$ case, we have 3 unknowns for each node for the velocity and 1 unknown more for the pressure. At the end, we build a mesh containing around 25 million nodes (figure 6 represents the initial mesh and one portion of the 25 million nodes fine mesh).

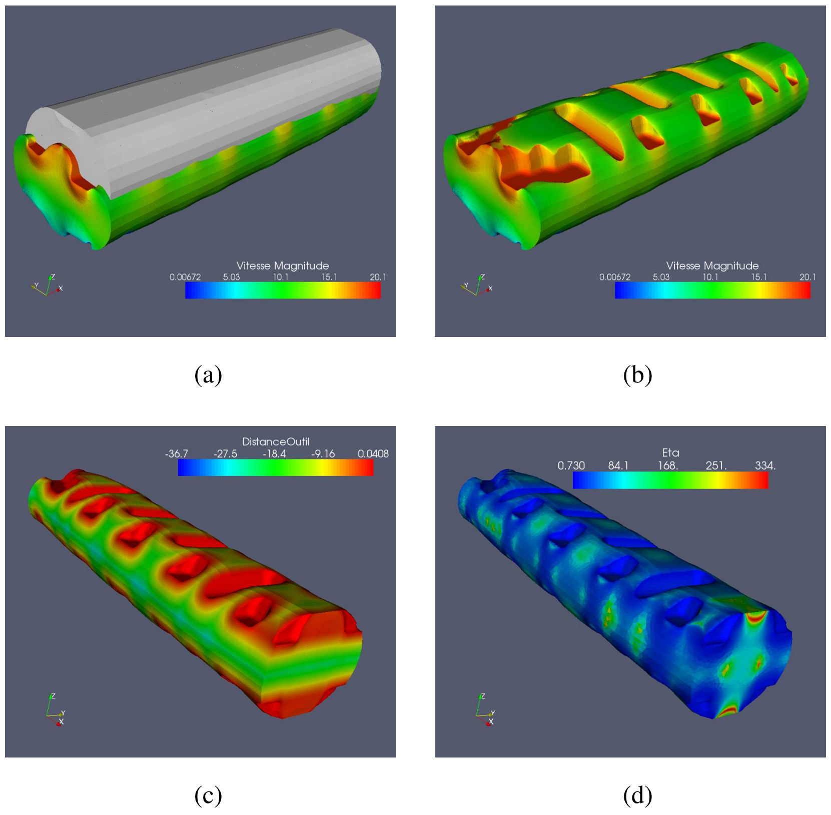
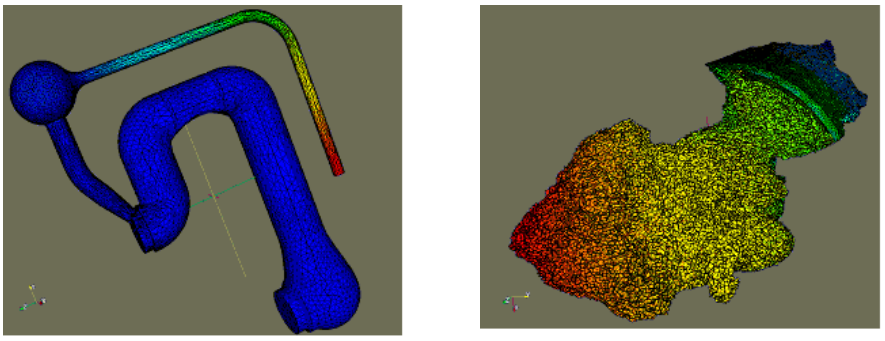


Figure 5. *All figures are captured at increment 120: (a), the initial and the deformed body obtained at increment 120. (b), the velocity magnitude over the deformed body. (c), the signed distance between tools and the body. (d), the viscosity field*

We remark that it is impossible to build such a mesh sequentially, as the highest value of memory we can allocate to one of our sequential computer is 16GB, which is not enough to store the mesh. To build it, we used the mesh component of CIMLib and we stored the mesh under its partitioned form. We started with the coarse mesh and then refined it successively by reducing the metric mesh size and by increasing the number of processors at each iteration.

Computing such a mesh was the first objective and after that, running the test case was not so difficult, as soon as the maximum of memory used is well dimensioned.

The computed simulation is close to the injection process even thought we use a simplified model with only one resolution of the Stokes equations. After that, the transport equation is solved using a Level-Set method, according to the computed velocity. The simulation was carried out by first doing one resolution of the Stokes



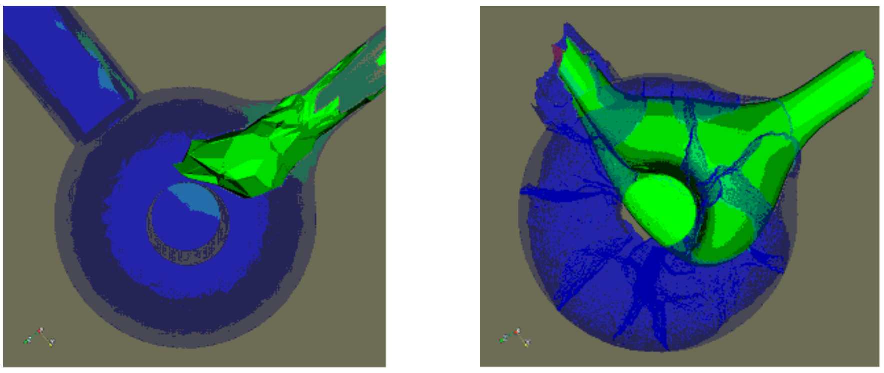
(a) (b)

Figure 6. *(a), the initial coarse mesh of the part with* $11,866$ *nodes and* $37,657$ *elements. (b), the mesh of only one of the* 88 *partitions used to represent the global piece. This subdomain contains* $304,252$ *nodes and* $1,637,436$ *elements and will be treated by one processor (one core)*

equation, and then by continuing with 1250 time increments to inject the polymer inside the cavity. This globally gives: - one linear system with 100 millions of unknowns for the Stokes equation (4 unknowns per node) - $1,250$ linear systems of 25 millions of unknowns for the transport equation (1 unknown per node and one linear system per increment)

By using 88 processors (cores) of the cluster, we have been able to do this computation. The global simulation time was close to 28 hours, with a little more than 2 hours to solve the Stokes problem, 21 hours solving transport equations and the last 5 hours to carry out some other computations, including writing results files. The next figure 6, we present the position of the front after one hundred time increments on the coarse mesh and on the very fine one. The front is illustrated by the zero iso-surface of the Level-Set. Of course, here the difference is very important and an intermediate mesh with some mesh adaptation techniques can be preferable in terms of precision vs computation cost (Mesri *et al.*, 2008). But the calculation made can be used as good reference for comparing with other computations.

The accuracy of the results on the 25 million nodes fine mesh is really impressive compared to the one of the coarse mesh. This is clearly due to the very large computations that we have been able to do. For example, on this small part of the global piece we have more than 5 millions nodes. The precision of this $3D$ calculation can be compared to what we are able to do in sequential but in $2D$. The size of the elements with such a mesh is close to a pixel.



(a) (b)

Figure 7. *(a), the predicted front position on the coarse mesh after 0.1 second, (b) the same front position at the same time using the 25 millions nodes mesh*

5. Conclusion

We presented CIMLib a fully parallel multi-component library. This library is based mainly on a parallel remeshing technology and a Finite Element solver. In this paper, new developments are performed in order to improve the efficiency of the parallel remeshing. The performance analysis shows that our parallel remeshing scales well. The measured speed-ups are close to the optimal. An other performance analysis is also obtained to highlight the scalability of the FE solver and other staff. A demonstration of CIMLib efficiency on large $3D$ simulations is also presented. The first large simulation presented focuses on a multi-bodies contact problem, including friction, for complex $3D$ forming processes. The second one is devoted to the multi-phase problems involved in the manufacturing processes of full parts. The mesh is refined during the both simulations the final mesh has over 25M nodes for the last one.

6. References

Arnold D., Brezzi F., Fortin M., “ A stable finite element for stokes equations”, *Calcolo*, vol. 4, n° 21, p. 337-334, 1983.

Basermann A., Clinckemaille J., Coupez T., Fingberg J., Digonnet H., Ducloux R., Gratien J.-M., Hartmann U., Lonsdale G., Maerten B., Roose D., Walshaw C., “ Dynamic load balancing of finite element applications with the DRAMA library”, *Appl. Math. Modelling*, vol. 25, n° 2, p. 83-98, 2000.

Basset O., Digonnet H., Guillard H., Coupez T., “ Multi-phase flow calculation with interface tracking coupled solution”, *in* E. O. M. Papadrakakis, B. Schrefler (eds), *Int. Conf. on Computational Methods for coupled problems in science and engineering*, 2005.

Brezzi F., Franca L., Russo A., " Further considerations on residual-free bubbles for advective-diffusive equations", *Comput. Meth. Appl. Mech. Engrg*, vol. 166, p. 25-33, 1998.

Brezzi F., Russo A., " Choosing bubbles for advection-diffusion problems", *Math. Meth. Models Meth. App. Sci.*, vol. 4, p. 571-587, 1994.

Bruchon J., Digonnet H., Coupez T., " Using a signed distance function for the simulation of metal forming processes: formulation of the contact condition and mesh adaptation. From a Lagrangian approach to an Eulerian approach.", *International Journal For Numerical Methods In Engineering*, vol. 78, n° 8, p. 980-1008, 2009.

Coupez T., " A mesh improvement method for 3D automatic remeshing", *in* N. W. et al. (ed.), *Numerical Grid Generation in Computational Fluid Dynamics and Related Fields*, Pineridge Press, p. 615-626, 1994.

Coupez T., " Génération de maillage et adaptation de maillage par optimisation locale", *Revue européenne des éléments finis*, vol. 9, n° 4, p. 403-423, 2000.

Coupez T., Digonnet H., Ducloux R., " Parallel meshing and remeshing", *Appl. Math. Modelling*, vol. 25, n° 2, p. 83-98, 2000.

Coupez T., Marie S., " From direct solver to a parallel iterative solver in 3D forming simulation", *International Journal of Supercomputer Applications And High Performance Computing*, vol. 11, n° 4, p. 277-285, 1997.

Digonnet H., Repartitionnement dynamique, mailleur parallèle et leurs applications à la simulation numérique en mise en forme des matériaux, PhD thesis, Ecole Nationale Supérieure des Mines de Paris, 2001.

Digonnet H., Coupez T., " Object-oriented programming for fast and easy development of parallel applications in forming processes", *in* K. Bathe (ed.), *Second MIT conference on computation Fluid and Solid Mechanics*, MIT, Elsevier, p. 1922-1924, 2003.

Digonnet H., Luisa S., Coupez T., " CIMlib: A Fully Parallel Application For Numerical Simulations Based On Components Assembly", Numiform, 2007.

Fortin M., Brezzi F., *Mixed and hybrid finite element method*, Springer, 1991.

Gay C., Montmitonnet P., Coupez T., Chenot J., " Test of an element suitable for fully automatic remeshing in 3d elastoplastic simulation of cold forging", *Journal of Material Processing Technology*, vol. 45, p. 683-688, 1994.

Hughes T., " Multiscale phenomena: Green's functions, the Dirichlet to Neuman formulation, subgrid scale models, bubbles and the origin of stabilized methods", *Comput. Meth. Appl. Mech. Engrg*, vol. 127, p. 387-401, 1995.

McInnes L., Balay S., Gropp W., Smith B., PETSc user manual, Technical Report - Revision 2.1.3 n° ANL-95/11, Argonne National Laboratory, 1995.

Mesri Y., Digonnet H., Guillard H., " Mesh Partitioning for Parallel Computational Fluid Dynamics Applications On a Grid", *Finite Volumes for complex applicationsIV*, Hermes Science, p. 631-642, 2005.

Mesri Y., Zerguine W., Digonnet H., Luisa S., Coupez T., " Dynamic parallel mesh adaption for three dimensional unstructured meshes: Application to interface tracking", *in* R. V. Garimella (ed.), *Proceedings of the 17th International Meshing Roundtable*, Springer, p. 195-212, October, 2008.

Mocellin K., Fourment L., Coupez T., Chenot J.-L., “ Toward large scale F.E. computation of hot forging process using iterative solvers, parallel computation and multigrid algorithms”, *International Journal For Numerical Methods In Engineering*, vol. 52, p. 473-488, 2001.

Osher S., Fedkiw F., “ Level set methods: An overview and some recent results”, *Journal of Computational Physics*, vol. 169, n° 2, p. 463-502, 2001.

Pichelin E., Coupez T., “ Finite element solution of the 3D mold filling problem for viscous incompressible fluid”, *Computer methods in applied mechanics and engineering*, vol. 163, p. 359-371, 1997.

Pichelin E., Mocellin K., Fourment L., Chenot J., “ An application of master-slave algorithm for solving 3D contact problem between deformable bodies in forming processes”, *European Journal of Finite Elements*, vol. 10, n° 8, p. 857-880, 2001.

Sethian J., “ Level sets methods and fast marching methods”, *Cambridge monograph on applied mathematics*, 1986.

Squillacote, *The ParaView Guide*, Kitware, Inc., 2005.

Received: 5 November 2008
Accepted: 27 October 2009