Parallel complex systems simulation

Schoneveld, A.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (http://dare.uva.nl)
Chapter 5

Dynamic load balancing using P-CAM

“I think there is a world market for maybe five computers.”

—Thomas Watson, chairman of IBM, 1943

5.1 Introduction

In this chapter we will discuss the implementation of two realistic simulation problems using the P-CAM framework. Two different approaches are taken to apply the P-CAM framework to the dynamic load balancing problem. The first approach entails a complete integration of the P-CAM framework and the simulation problem, in other words the parallel simulation program is entirely implemented using P-CAM. In the other approach, the parallel simulation program and the dynamic load balancing process are separated: an existing parallel simulation code utilizes a dynamic load balancing algorithm implemented in P-CAM to steer the repartitioning process.

In this first case study the emphasis is on issues like fast task graph generation and irregularity both in the task graph and in the decomposition. Furthermore, the ability of the framework to enable an application to utilize advanced parallel computing techniques like dynamic load balancing is demonstrated. The simulation model describes an aggregation process of an irregularly shaped object in a flowing medium.

In the second case study an existing industrial parallel finite element program is extended with dynamic load balancing capabilities. The objective of this study is to show the benefit of using the P-CAM framework in a large scale simulation.
5.2 Dynamic Load Balancing in an Aggregation Process

We have implemented a Lattice Boltzmann solver to simulate the fluid flow around a growing obstacle. The fluid flow is calculated inside a two dimensional domain containing an irregularly shaped growing object. In this simulation the workload continuously decreases as a consequence of the growing object. In the remainder of this section we will introduce the different aspects of the simulation. First, the Lattice Gas method is briefly described as an introduction to the Lattice Boltzmann method. Subsequently, we will discuss the process of growth or aggregation. Having stated our simulation problem, the generation of a task graph and an initial decomposition from a problem description is described. Finally, various simulation results are presented, focusing on the computational aspects.

5.2.1 A Particle Approach to Fluid Flow

An alternative approach to model physical systems in fluids and gases, for example complex flows, complex fluids, and reactive systems, are lattice gas methods. The basic idea is illustrated in Fig. 5.1. The behavior of the fluid surrounding the obstacles can be modeled as a continuum (Fig. 5.1a) using a set of partial differential equations, as a molecular liquid (Fig. 5.1b) using a Molecular Dynamics method, or alternatively using a discrete velocity (lattice) gas (Fig. 5.1c).

![Figure 5.1: Three different approaches to model the fluid surrounding the obstacles: (a) a set of partial differential equations, (b) molecular dynamics, and (c) a lattice gas (after [87]).](image)

A very simple lattice gas method with which an incompressible fluid in 2D can...
be modeled is the FHP Lattice Gas Automata (LGA) [55]. In this method the fluid is represented by "particles" which move, synchronously, from a lattice site to a neighbouring lattice site; they interact locally according to a given set of rules. In the FHP model, the particles reside on a triangular lattice and are located in the centers, the nodes, of the hexagons in the lattice. Each particle, or lattice site, is connected with six neighbors and can move in a next time step to one of these neighbors, or can be stationary. In the FHP model both time and velocities are discretized. Two successive time steps of the FHP model are shown in Fig. 5.2.

Basically, the time evolution of the model consists of two phases:

1. In the propagation phase, particles move along lattice bonds, from a lattice node to one of its neighbors.

2. In the collision phase, particles residing on the same lattice node shuffle their momenta locally, subject to conservation of mass and momentum.

In Fig. 5.3 a subset of the collision rules, used in the FHP model, is shown. Within LGA models, many types of boundary conditions can be relatively easy specified. For example the rigid no-slip condition can be specified by setting the lattice sites which represent the obstacle, to the state "stationary." The lattice gas method is especially suitable to determine flow patterns, for low velocities, about obstacles with a complex geometry. An example of such an experiment is depicted in Fig. 5.4 where flow about an irregular object is shown. The single particle dynamics of the lattice gas approximation introduces noise into the results. In the computation of flow patterns this discreteness artifact can be partially avoided by taking spatial averages for the velocities. In practice the average velocity \( u \) in a 2D lattice gas model is determined in clusters with a minimal size of \( 64 \times 64 \) nodes [39]. The limited range of velocities also restricts the allowed range of Reynolds numbers, which can be simulated with the lattice gas method.

**Figure 5.2**: From left to right: the initial, propagation and collision phases in the FHP model.

**Macroscopic equations**

The flow of a time dependent incompressible fluid can be described by the two basic macroscopic (Navier Stokes) equations from hydrodynamics:

\[ \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \]

\[ \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \mathbf{u} = -\nabla p + \mu \nabla^2 \mathbf{u} \]

*A fluid with constant density, as for example water under most conditions.*
where \( u \) is the velocity, \( t \) time, \( p \) the pressure, and \( v \) the kinematic viscosity which will be assumed to be constant. For low velocities it can be demonstrated that the FHP model approximates the Navier Stokes Equations (Eqs. 5.1 and 5.2) (see [55]):

\[
\nabla \cdot u = 0
\]

\[
\frac{\partial u}{\partial t} + (u \cdot \nabla) u = -\frac{1}{\rho} \nabla p + v \nabla^2 u
\]  

(5.2)

where \( \rho \) is the average number of particles per cell and \( g(\rho) \) a function of this density \( \rho \). The FHP model is not Galilean invariant \(^1\). The lack of Galilean invariance appears in the equations of motion through the factor \( g(\rho) \). In other words, in the FHP model there exists a velocity dependency of the density [39]. This does not correspond to physical properties of an incompressible fluid which should have constant density everywhere. For low velocities this velocity dependency does not have strong effects. When in Eq. 5.3 the time \( t \), the viscosity \( v \), and the pressure \( p \) are scaled with the factor \( g(\rho) \) \( (t' = g(\rho)t, v' = v/g(\rho)) \), the equation transforms into the Navier Stokes equation [39].

\(^1\) A physical system is Galilean invariant if there is no dependence of the reference frame.
5.2 Dynamic Load Balancing in an Aggregation Process

Figure 5.4: Flow pattern around an irregularly shaped obstacle as a result of an LBGK simulation. The flow is streaming from left to right. The streamlines are depicted as well as the flow velocity (darker colors indicate lower velocities).

Figure 5.5: Face Centered hypercube (FCHC). Along the dotted links between the nodes at most one particle can propagate, while at the solid links up to two particles can propagate.

Lattice gases in 3D

There also exist 3D extensions of the FHP-model [54]. In the 3D extension, the particles reside on a cubic lattice (A Face Centered HyperCube (FCHC), see Fig. 5.5) where each node is connected with 24 other nodes. Along the dotted links between the nodes at most one particle can propagate, while at the solid links up to two particles can propagate. In practice this FCHC method suffers from several problems: to avoid discreteness effects by spatial averaging an enormous lattice is required and the implementation of the collision operator, due to the large amount of possible collisions, is troublesome [152]. Due to the large scale separations it is not necessary to use a model which in-
cludes the detailed mechanics of the molecules surrounding the obstacles. The fluid can be modeled with a lattice gas, the “particles” in the fluid are represented on a much higher magnitude scale than the actual molecules. It can be demonstrated that lattice gas methods still include most features of a fully molecular simulation, while it is an order of magnitude faster in computation. The lattice gas methods fill the gap between MD and PDE methods, the LGA method is sometimes described as the “poor man’s MD” [39]. However, as already discussed, the LGA method has a number of disadvantages. To overcome these disadvantages, a derived method, the Lattice Boltzmann Method (LBM), has been formulated.

The Lattice Boltzmann Method

As noted in the previous sections, the major drawbacks of early lattice-gas methods are statistical noise, lack of Galilean invariance and the exponential complexity of the collision operator. Statistical noise is inherent to the Boolean nature of the computations (each bond is occupied by at most one particle). Part of the problems of the LGA method can be overcome by using an alternative lattice gas method based on the Lattice Boltzmann Equation (LBE) [12, 25, 129, 24]. The idea behind the LBE method is to track a population of particles instead of a single particle (a reasonable modification justified by the Boltzmann molecular chaos assumption from kinetic theory). This mean-value representation of particles eliminates the statistical noise problem. Several modifications of the collision operator have been published [39]. Basically, the collision operators have been expressed in terms of relaxation of the hydrodynamic fields, like density and velocity. The latest major modification to date is the lattice BGK (Bhatnagar-Gross-Krook) model where the collision operator is based on a single time relaxation to the local equilibrium distribution [124]. This model is the simplest one in the hierarchy of lattice Boltzmann methods and has been used in our simulations.

The time evolution of the lattice-BGK model is given by [124]:

\[ f_i(r + c_i t + 1) = f_i(r, t) + \frac{1}{\tau} (f_i^{(0)}(r, t) - f_i(r, t)) , \]  

where \( c_i \) is the i-th link (see Fig. 5.6), \( f_i(r, t) \) is the density of particles moving in the \( c_i \)-direction, \( \tau \) is the BGK relaxation parameter, and \( f_i^{(0)}(r, t) \) is the equilibrium distribution function towards which the particle populations are relaxed. The hydrodynamic fields such as the density (\( \rho \)) and the velocity (\( v \)) are obtained from moments of the discrete velocity distribution \( f_i(r, t) \):

\[
\rho(r, t) = \sum_{i=0}^{8} f_i(r, t) \quad \text{and} \quad v(r, t) = \frac{\sum_{i=0}^{8} f_i(r, t) c_i}{\rho(r, t)}.
\]

The equilibrium distribution function can be chosen in many ways. A common choice is [124]
5.2 Dynamic Load Balancing in an Aggregation Process

Figure 5.6: A LBGK particle with 8 links, storing 8 velocity populations \( f_i \) and 1 rest particle with population \( f_0 \).

\[
f_i^0 = t_i \rho (1 + \frac{1}{c_s^2} (c_i \cdot v) + \frac{1}{2c_s^2} (c_i \cdot v)^2 - \frac{1}{2c_s^2} v^2),
\]

where \( t_i \) is a weight-factor depending on the length of the link vector and \( c_s \) is the speed of sound (in our simulations, \( t_0 = \frac{1}{3}, t_1, t_4, t_6, t_8 = \frac{1}{5} \) and \( t_1, t_3, t_5, t_7 = \frac{1}{36} \)). The Lattice Boltzmann models presented here yield the correct hydrodynamic behavior for an incompressible fluid in the limit of low Mach and Knudsen numbers [124]. The kinematic viscosity of the simulated fluid \( \nu \) and the speed of sound \( c_s \), expressed in lattice units, are \( \nu = \tau \frac{1}{3} \) and \( c_s = \sqrt{\frac{1}{3}} \) [124]. The fluid pressure \( p(r,t) \) is given by the relation

\[
p(r,t) = c_s^2 (\rho(r,t) - \bar{\rho}), \tag{5.5}
\]

where \( \bar{\rho} \) is the mean density of the fluid.

The LBGK method has been implemented in the P-CAM framework in order to support parallel simulation and dynamic load balancing. Each P-CAM cell is assigned an LBGK particle (see Fig. 5.6). The collision operator is only dependent on information local to the cell and for the propagation operator only nearest neighbour interaction is required. Hence, the LBGK method is easily mapped onto the P-CAM cell framework. We have defined three different kind of LBGK cells, fluid nodes, boundary nodes and solid nodes. All fluid nodes and boundary nodes are updated in parallel by applying sequentially the propagation step and LBGK collision operator. Boundary nodes have 1 or more solid nodes as neighbors. Fluid nodes are completely surrounded by fluid nodes. A bounce back boundary condition is used for the boundary nodes, that is, if a link
points towards a solid node, the value of that link is bounced back to its opposite link. Furthermore the fluid is driven by applying a so called body force to each fluid node at each iteration step. The direction and strength of this body force is fixed from the beginning of the simulation.

5.2.2 Aggregation

In flow limited growth simulations, the growth dynamics are usually dependent on properties of the surrounding fluid [81]. Because we are interested in parallelization issues of the simulation and not in the interpretation of the simulation experiments, for simplicity the object grows independently of the fluid flow. A simple aggregation process, called Eden growth (see Section 1.3.3), is used in the simulation experiments. In Eden growth, aggregation occurs at perimeter sites of the seed object with a fixed probability. The dynamics results in a compact shaped cluster. The perimeter sites that belong to the object become new seed sites and as such do not contain any fluid (solid nodes). The growth steps occur in between (a number of) flow steps, inducing a steady reduction of the number of fluid nodes. In addition, the workload reduces locally at those processors containing the newly acquired solid nodes.

5.2.3 Generating task graphs and decompositions

In the case of a fluid flow problem, the problem domain entails a fluid medium possibly filled with obstacles. This problem domain is constructed, using a suitable modeling tool, from the physical specifications of the actual system. Given such a specification, i.e., a geometrical description, a task graph is generated. In our case, this description is a 2 dimensional pixel grid. The task graph is constructed based on the LBGK 8-connected nodal connectivity (see Fig. 5.6). The colors in the pixel grid are used to discriminate between three types of nodes: fluid, solid and seed nodes. The fluid nodes contain fluid, the solid nodes are empty and the seed nodes specify the points where growth can occur. Subsequently, the generated task graph is fed into a decomposition tool, which partitions the task graph in a specified number of parts. For our experiments, we have used the problem domain depicted in Fig. 5.7. From this figure a task graph is generated and decomposed using a domain decomposing technique: ORB in 16 parts.

5.2.4 Experimental setup and results

In this section we study the computational results and the simulation results of a fluid flow through the irregular object introduced in the previous paragraph. In the center of the simulation domain (see Fig. 5.7) a seed is placed. During the simulation the object grows according to the Eden model introduced in Section 5.2.2. The perimeter sites of the solid nodes are the seed sites. The growth probability at the seed sites is fixed to 0.1. Between two growth steps, 20 LBGK
iterations are executed to relax the fluid velocity field. Fluid nodes that have become solid nodes are removed from the simulation nodal list (annihilated). Due to the continuous growth of the object, the number of fluid nodes decreases during the simulation and consequently the work load associated with an individual processor decreases as well. The implicit synchronization of the LBGK algorithm causes long idle times for under loaded processors, wasting valuable processing resources. This problem can be solved either by using a special static initial partitioning or by resolving the imbalance dynamically by applying the dlb methods as discussed in Section 2.3.

We consider three different parallelization strategies: static ORB decomposition, scattered redundant decomposition [38] and dynamic load balancing. For the experiments we fix the number of processors to 16. Static ORB decomposition simply partitions the initial domain (Fig. 5.7) into 16 parts, which remains unaltered during the simulation. As a consequence, the workload on the interior processors will decrease and eventually become zero given by the growing cluster. A static approach to overcome this load imbalance is to apply a so called scattered decomposition. A scattered decomposition can be realized as follows: first the domain is split into a number of parts much larger than the number of processors (e.g., by ORB), followed by a random assignment of the created partitions to the 16 available processors (see Fig. 5.8). Depending on the number of initial partitions, the load imbalance situation can be resolved. One has to consider the following trade off: scattering on the finest grain size (randomly assign
cells to processors) will eliminate the load imbalance entirely, however at the cost of a large (maximum) communication volume. An optimal situation must find a balance between load imbalance reduction and communication volume. Because it is not our objective to solve this optimization problem, we have chosen to use an arbitrary initial ORB decomposition in 64 parts. The third strategy, employs the dynamic load balancing method as introduced in Section 2.3. As already stated, the calculated workflow can be resolved by choosing from several cell selection algorithms (see Chapter 2). In our experiments, we will apply four of them:

- **dlb 1:** the categorical strategy (see Section 2.3.4)
- **dlb 2:** the com strategy (see Section 2.3.3)
- **dlb 3:** the random with edge smoothing strategy (see Sections 2.3.3 and 2.3.5)
- **dlb 4:** the random without edge smoothing strategy (see Section 2.3.3)

These strategies will be compared to the following two static load balancing approaches:

- **no dlb:** only use initial ORB partitioning
- **scatter:** scatter an ORB partitioning on the available processors, using more partitions than processors

**Figure 5.8:** The simulation domain decomposed into 16 partitions by scattering a 64 ORB partitioning.
The advantage of the dynamic load balancing method over the static approaches is the fact that the load balance can be maintained during the entire simulation. But again, we have to consider the trade-off of exact load balance and the time spent in the dynamic load balancing step. We have chosen to use a fixed dynamic load balancing interval of 5 (growth) iterations. This results in a monitoring of the load imbalance $I$, which is defined as:

$$I = \frac{W_{\text{max}} - W_{\text{avg}}}{W_{\text{max}}}$$

(5.6)

where $W_{\text{max}}$ and $W_{\text{avg}}$ are the current maximum load and the achievable average load respectively. $I$ is monitored every 5 iterations and actual re-balancing whenever $I > 0.025$ (arbitrary). As an initial decomposition for the dynamic load balancing strategies we use the 16 part ORB.

All load balancing strategies are transparently implemented using the P-CAM framework. The no $\text{dlb}$ and $\text{scatter}$ strategies simply require different initial decompositions. The $\text{dlb}$ strategies require an additional dynamic load balancing step based on the cell migration functionality of P-CAM. The simulation code itself does not need to be changed.

To compare the different strategies, we run the fluid flow/aggregate growth simulation for 300 (growth) iterations. During the simulation the time spent in the actual simulation (1), the time spent in the load balancing step (if appropriate) (2) and the total time (3) are monitored for each loop iteration. Do note that the
Dynamic load balancing using P-CAM

Figure 5.10: Evolution of the number of fluid cells on each of the 16 processors, for each iteration of the algorithm using dynamic load balancing strategy dlb 2.

| strategy | total time | comp. time | migr. time | |E|
|----------|------------|------------|------------|----------------|
| no dlb   | 2842       | 2842       | 0          | -              |
| scatter  | 2721       | 2171       | 0          | -              |
| dlb 1    | 2580       | 2528       | 51         | 1628410        |
| dlb 2    | 2523       | 2482       | 41         | 1505373        |
| dlb 3    | 2651       | 2585       | 66         | 1577628        |
| dlb 4    | 2661       | 2602       | 58         | 3139393        |

Table 5.1: Total running time, time spent in simulation problem, total migration time for the different parallelization strategies and the cumulative communication volume |E| (total number of shared edges over all iterations).

Timing results include synchronization times. If, for example, the load balancing step involves an irregular workflow, i.e., large differences exist between the amount of workflow migrated to the different processors, this may result in extra synchronization times for fast processors after the dynamic load balancing step. This additional synchronization time will become part of the loop iteration time following the load balancing step.

The loop iteration times (1) results are shown in Fig. 5.9. The gross gain, i.e., not including re-balancing times is lowest at each iteration step for the dlb 2 dynamic load balancing strategy. The scattered strategy is initially the slowest, but after the cluster has reached a substantial size, scattering pays off.

In Tab. 5.1, the cumulative times are depicted. Compared to the static 16 part no dlb strategy, the other strategies have a lower total execution time. The total gain is maximum for the dlb 2 strategy, which boils down to a modest 9% gain.
in execution compared to the 16 partition \textit{no dlb} strategy. For the four different dynamic load balancing strategies we compare their total communication volume over 300 iterations.

Figure 5.11: The fluid velocity field after 300 growth steps. Darker colors indicate lower velocities. The fluid inlet is positioned on the left side of the figure, while the fluid leaves the geometry from the right.

To complement the computational results, the final fluid velocity field is shown in Fig. 5.11. Note the lower fluid velocities near the boundaries of the domain. Finally, to give an indication of the dynamics of dynamic load balancing, a snapshot of a decomposition, after 250 iterations, is given in Fig. 5.12, using the \textit{dlb} 2 strategy.

5.2.5 Discussion

The results show the strength of using P-CAM for applying a variety of parallel computing solutions to the same parallel complex simulation. The genericity of the task graph approach and the assignment of graph nodes or cells to processors becomes clear. In Section 5.2.4, we have experimented with several parallelization approaches without altering the simulation code. The first approach simply used a static ORB partitioning, which, due to the dynamics of the simulation problem, ultimately leads to an imbalanced workload situation. The slowest processor determines the execution time of the simulation. In this simulation the cluster grows from the center, causing the processor partitions located in the middle of the simulation domain (see Fig. 5.7) to become under
loaded and eventually idle. A possible solution to this problem is dividing the simulation in more parts than the available processors, resulting in a so called redundant decomposition enabling a more balanced partitioning of those partitions containing the growing cluster (see Fig. 5.8). The third solution uses the dynamic load balancing of Section 2.3, enabling an exact detection and resolution of the workload imbalance. The experimental results of Fig. 5.9 and Table 5.1 show the effect on the execution time of the three strategies. For this specific simulation problem a scattered decomposition gains 4% in execution time compared to a 16 processor ORB decomposition in spite of the additional communication overhead introduced by the scattered method. For the dynamic load balancing strategies we used three different cell selection heuristics as discussed in Section 2.3. From Table 5.1 we may conclude that the random cell selection mechanism (dlb 3) performs poorer than the other two strategies, which is the result of the rougher processor boundaries. Enabling “edge smoothing” with random cell selection (dlb 4), does improve the results. The edge smoothing heuristic reduces the total communication volume, however at the cost of optimal load balance. This explains the fact that, although the categorical cell selection heuristic has a higher communication volume than random with edge smoothing, the first strategy performs better. The edge smoothing algorithm did not have much effect on the dlb 1 and dlb 2 strategies (data not shown), therefore we have omitted it from the presented results.

Figure 5.12: A snapshot of a decomposition near the end of the simulation. The 16 processor partitions are shaped by the dynamic load balancing process and the cell selection strategy.
The simulation problem of aggregate growth in a fluid flow does not provide a spectacular gain if dynamic load balancing is applied (only 9%). As argued in Section 2.3, this is caused by the locally decreasing workload nature of the problem. A problem with locally increasing workload will have a larger performance gain if dynamic load balancing is used.

### 5.3 Parallel Finite Element Simulation

In this section we will introduce a new method for parallelizing Finite Element (FE) simulations enabling the use of dynamic load balancing. A physical space partitioning is obtained by dividing the bounding cube of an FE mesh into a large number of sub cubes. The cube mesh together with a workload attribute assigned to each cube is used to present an abstract view of the simulation to a dynamic load balancing process. Based on this abstract view the dynamic load balancing process decides on a possible repartitioning of the mesh. The dynamic load balancing process itself is diffusion based, that is cubes are migrated between neighboring partitions. The P-CAM framework is used to implement the dynamic load balancer by assigning cubes to P-CAM cells. In Fig. 5.13 a schematic overview is given of how the P-CAM load balancer cooperates with the parallel finite element solver. The solver that is used in this study is the explicit FE solver MSC/DYTRAN from the MacNeal Schwendler Corporation.

![Diagram of P-CAM dynamic load balancer](image)

**Figure 5.13:** A schematic overview of the P-CAM dynamic load balancer to support the parallel finite element simulator MSC/DYTRAN
5.3.1 Cube based partitioning

Most existing partitioning methods for parallel Finite Element simulations are based on examining the connectivity of the element mesh and dividing the mesh in such a way that the number of cut edges is minimized (see e.g., [141, 135]). These methods assume data flow only between adjacent elements in the mesh. For different reasons this is not applicable to the explicit finite element solvers of MSC/DYTRAN [28]. Specifically, such explicit solvers can operate on several meshes at once. Data can flow between two meshes that are not even related by connectivity due to contact and coupling. MSC/DYTRAN contains two finite elements solvers: Lagrangian (structural) and Eulerian (fluid). Beside these two main solvers, which are entirely separate, an algorithm for structure/structure interaction (contact) and fluid/structure interaction (coupling) is also provided. Two different meshes are used to represent contact and coupling between elements. Both reside in the same physical space. These interactions can not be derived from the connectivity of the mesh and have to be determined by a spatial search. Because of this we have decided to partition the model in physical space to ensure that data flow can be localized to single CPUs or nearest neighbor CPUs. Physical partitioning is realized by superimposing a coarse imaginary 3D mesh of 512 cubes over the model meshes. Each cube can contain entities from various meshes. Each cube has a property called workload, which is the amount of CPU time required to process all the elements in the cube. The workload per cube varies over time as the cube’s content changes in quantity or character. The physical space is divided into 512 ($8^3$) cubes in order to create a redundant decomposition [38], resulting in many more cubes than CPUs. A redundant decomposition is necessary to enable the dynamic load balancing process to shift cubes between processors in order to resolve an imbalance situation.

5.3.2 Finite Element Simulation of an Underwater Explosion

We will use a finite element simulation of an underwater explosion as our study object. Because there are two materials in the Euler domain (gas and water) we use multi-material elements. The explosive is placed in the middle of the bucket and resides at the corner of two symmetry planes about 22 cm below the water surface. The region where the explosive is initially located is meshed very fine to be able to accurately compute the detonation process (see Fig. 5.14). The computational mesh remains fixed in space and in time and the materials can flow through the mesh from element to element. Due to the asymmetrical nature of the reduced model (only a quarter of the original physical model is used in the simulation) and the propagation of a shock wave through the water upon detonation, we can expect an imbalance situation when the model is initially partitioned in equally sized volumes (ORB partitioning). Therefore we need repartitions in the dynamic load balancing process in order to cope with this situation.
The major part of the computational time in the solver is taken up by the trans­
portation of multi-material elements which affects the workload in the cur­
rent simulation process. Triggered by the gas explosion, the transportation of
mass, momentum and energy from element to element causes the work loads
to change during the simulation.

Fig. 5.15 visualizes the simulation process of the pressure distribution in the
model using a "pentolite" explosion with SIE (specific internal energy)= 5.706
MJoule from timestep=0.15 to 0.45 mSec. Upon detonation, the solid explosive
material is instantaneously transformed into high pressurized explosion gases.
By means of the material flow, the pressure explosion will expand radially and
after some time steps will distribute like a "butterfly" away from the charge as
shown in the figure. The simulation has been executed on 8 processors. The
dynamic load balancing process is shown in the same figure. Each processor
partition is represented by a unique grey value.

5.3.3 Results

In this section we will present some of the simulation results of the underwa­
ter explosion and the computational behavior of various load balancing strate­
gies. We will present experiments on two different kinds of parallel machines:
an MPP SP2 cluster and a small scale 2 node 486-Pentium cluster. The exper­
iments on the MPP are used to show the effectiveness of the dynamic load bal­
ancing strategy on a homogeneous dedicated parallel machine. For this case the
re-balancing dynamics are entirely driven by the inhomogeneous load dynamics of the parallel simulation. For the 2 node workstation cluster the initial imbalance is mainly caused by the difference in computational performance between the 486 and the Pentium machine.

**MPP results**

We did an extensive series of experiments on an IBM SP2 cluster using 4, 8 and 16 processors. The communication primitives for DYTRAN have been implemented using the MPI message passing library for Fortran. The dynamic load balancing or partitioning process runs on 1 processor. Prior to repartitioning all cube loads are collected on processor 0 by a global communication step. Each parallel simulation is initially decomposed using an orthogonal recursive bisection (ORB) of the cubes. After the initial decomposition one out of the three available load balancing methods is chosen: no dynamic load balancing (no dlb), dynamic load balancing with *categorical* cell selection and dynamic load balancing with *com* cell selection. Two important parameters for dynamic load balanc-
5.3 Parallel Finite Element Simulation

Table 5.2: Variation of the MAX.IMB parameter for com cell selection and DLB_FREQ=25. The total times spent in the Euler solver and the partitioning process (collecting loads and migration) and the total execution time are given in seconds.

<table>
<thead>
<tr>
<th>P</th>
<th>MAX.IMB (%)</th>
<th>Selection</th>
<th>DLB_FREQ</th>
<th>Euler</th>
<th>Partitioning</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>no db</td>
<td>-</td>
<td>-</td>
<td>3508</td>
<td>-</td>
<td>4931</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>com</td>
<td>25</td>
<td>2784</td>
<td>268</td>
<td>4587</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>com</td>
<td>1</td>
<td>3209</td>
<td>5123</td>
<td>9943</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>categorical</td>
<td>25</td>
<td>2874</td>
<td>194</td>
<td>4570</td>
</tr>
<tr>
<td>16</td>
<td>no db</td>
<td>-</td>
<td>-</td>
<td>2752</td>
<td>-</td>
<td>4240</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>com</td>
<td>25</td>
<td>1543</td>
<td>225</td>
<td>3215</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>com</td>
<td>1</td>
<td>1452</td>
<td>4549</td>
<td>7602</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>categorical</td>
<td>25</td>
<td>1670</td>
<td>258</td>
<td>3516</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>categorical</td>
<td>1</td>
<td>1510</td>
<td>5102</td>
<td>8132</td>
</tr>
</tbody>
</table>

Table 5.3: Comparing load balancing strategies. The total times spent in the Euler solver and the partitioning process (collecting loads and migration) and the total execution time are given in seconds.

<table>
<thead>
<tr>
<th>P</th>
<th>MAX.IMB (%)</th>
<th>Selection</th>
<th>DLB_FREQ</th>
<th>Euler</th>
<th>Partitioning</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>no db</td>
<td>-</td>
<td>-</td>
<td>3508</td>
<td>-</td>
<td>4931</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>com</td>
<td>25</td>
<td>2784</td>
<td>268</td>
<td>4587</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>com</td>
<td>1</td>
<td>3209</td>
<td>5123</td>
<td>9943</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>categorical</td>
<td>25</td>
<td>2874</td>
<td>194</td>
<td>4570</td>
</tr>
<tr>
<td>16</td>
<td>no db</td>
<td>-</td>
<td>-</td>
<td>2752</td>
<td>-</td>
<td>4240</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>com</td>
<td>25</td>
<td>1543</td>
<td>225</td>
<td>3215</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>com</td>
<td>1</td>
<td>1452</td>
<td>4549</td>
<td>7602</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>categorical</td>
<td>25</td>
<td>1670</td>
<td>258</td>
<td>3516</td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>categorical</td>
<td>1</td>
<td>1510</td>
<td>5102</td>
<td>8132</td>
</tr>
</tbody>
</table>

Note that including the communication time in such a synchronized simulation would result in \( I \) being approximately zero.
In order to determine an optimal value of the MAX_JMB parameter we vary the value of the MAX_JMB parameter (5%, 10%, 15% and 20%). 1200 cycles of the Euler solver are executed on different numbers of processors; the time spent in the Euler solver, the partitioning process and the total execution time are monitored. A selected set of results is depicted in table 5.2 using the com cell selection strategy. The optimal value of MAX_JMB is determined by a function of the migration cost and the performance benefit of reducing the load imbalance.

Based on varying the MAX_JMB parameter we have chosen the value MAX_JMB = 7% for the subsequent experiments. The other parameter, the frequency of calling the partitioner, DLB_FREQ is also important for finding a balance between partitioning overhead and the computational gain of reducing the load imbalance. The overhead of calling the partitioner is mainly caused by the global communication, which collects all the cube loads on one processor. The workload for each cube that is supplied to the partitioner is averaged over the last DLB_FREQ iterations to reduce the noise level and to ignore spurious dynamics. The difference in computational performance between calling the partitioner every cycle and calling it only every 25 cycles is shown in table 5.3. As for the previous experiment, times spent in the Euler solver, the partitioning step and the total execution time are monitored. Both the categorical and com cell selection strategy results are displayed using 4, 8 and 16 processors and MAX_JMB = 7%.

Clearly the overhead in calling the partitioner too often (in this case every cycle) is significant. We have empirically established that a DLB_FREQ of 25 is close to optimal for our simulation experiments.

Figure 5.16: The evolution of the imbalance parameter for three load balancing strategies on 8 processors: no dynamic load balancing (No db), dynamic load balancing with com cell selection and with categorical cell selection.
5.3 Parallel Finite Element Simulation

In Fig. 5.16 the evolution of the imbalance, \( I \), on an 8 processor partition is given for each of the three load balancing strategies. The spikes in the plots are due to extra delays by cache misses and page faults. The block wall character of the dlb strategies is caused by the value of DLB_FREQ; the width of the blocks is exactly 25 iterations. The same experiment is shown in Fig. 5.17 for 16 processors.

To quantify the efficiency of the dynamic load balancing process we plot the dlb time \( t_{\text{dlb}} \) versus the nodlb time \( t_{\text{nodlb}} \). In Fig. 5.18, \( t_{\text{dlb}} \) for the Euler kernel and the wall-clock time is depicted for the categorical and com strategies on 4, 8, and 16 processors.

From the experimental results presented in Figs. 5.16, 5.17 and 5.18 it is clearly demonstrated that dynamic load balancing pays off in terms of turn around time. From Figs. 5.16 and 5.17, a clear reduction of the load-imbalance \( I \) in the Euler solver can be observed for different cell selection strategies compared to static decomposition. Also from this figure it can be observed that the com selection strategy seems slightly better in terms of load-imbalance than the categorical selection strategy. This observation is supported by the results of Fig. 5.18 where the efficiency of the dlb strategies is compared to no dlb, for 4, 8 and 16 processors. The wall clock time efficiency of the com strategy is better than categorical for 8 and 16 processors and approximately equal for 4 processors. However, just taking into account the Euler kernel, the com strategy degrades for 16 processors compared to the categorical strategy. For 16 processors the center of mass based strategy is less suited to resolve the workflow than the connection based scheme. This is not caused by some inherent
imperfection of the strategy, but merely by an “unlucky” choice of cells. More specifically, heavily loaded cells are selected that result in an overshoot of the workflow. An explanation why the com strategy still has a lower turnaround time than categorical must be found in the communication pattern resulting from the cell selection. Apparently com selection results in allocations with on average lower communication volumes between processors.

**Heterogeneous workstations results**

The previous section showed how an heterogeneous load distribution resulted in a triggering of the dynamic load balancer. An other interesting situation is a heterogeneous computing environment, either due to external load or due to differences in the performance of the computational resources. We have setup an experiment with a substantial performance mismatch between the CPUs, which renders the heterogeneous load distribution of our finite element simulation almost insignificant compared to the CPU performance imbalance. The smallest possible workstation cluster, a 2 node system is used to simulate the underwater explosion. The cluster consists of a 66MHz 80486 and a 400MHz Pentium II processor. Figure 5.19 shows the evolution of the cube partitioning over the two machines and the pressure distribution. Initially the ORB partitioning divides the simulation model equally between the 486 and the Pentium. After two repartitioning steps (on cycles 11 and 21), 90% of the cubes are allocated on the Pentium. Because the average cube load over the past 10 cycles (DLB_FREQ) is passed to the partitioner, the final reallocation is delayed to cy-
5.3 Parallel Finite Element Simulation

Figure 5.19: The two upper pictures and the picture left under depict the cube partitioning after a re-balancing step on cycles 1, 11 and 21 respectively. The picture on the right at the bottom show the pressure distribution on cycle 41.

cle 21. In this experiment, the categorical cell selection strategy has been applied. The wall-clock iteration times for the 486 and the Pentium partitions are shown in fig. 5.20. After the last reallocation step on cycle 21 the iteration times of both machines approximately coincide.

5.3.4 Discussion

In this section we have described a physical domain partitioning approach for the dynamic load balancing of a parallel finite element simulation. A parallel simulation framework, P-CAM, has been used to implement a separate dynamic load balancing module. The simulation model is decomposed into 512 cubes, where each cube contains a part of the finite element mesh. Associated with each cube is a workload, quantifying the total CPU time to finish a single iteration of the Euler solver. The 3D cube mesh and its corresponding workloads are used by the dynamic load balancing module to decide whether to repartition and how to resolve the load imbalance by reallocating cubes to processors. Using an underwater explosion simulation, the computational performance of
a statically decomposed parallel process is compared with a dynamically load balanced version. We have shown that the approach that has been chosen to deal with dynamic repartitioning, is able to resolve both heterogeneity in the application workload as well as heterogeneity in the computational resources. This is a consequence of the duality between both quantities, i.e., the workload determines the computational performance as well as the other way around.

Summarizing, we have shown that physical domain partitioning is a viable route towards dynamic load balancing of complex finite element simulations. A re-usable parallel framework has been applied to model the computational dynamics of the parallel simulation and to steer the reallocation process of simulation tasks. However, many open issues remain in this work. The granularity of the sub-cubes is a very important question. Large cubes induce smaller overhead in migration and communication but also less room for balancing the workload. Small cubes result in a lot of overhead but great flexibility in dynamic load balancing. The scale of the spatial events in the specific simulation plays an important role in choosing the correct cube size: small spatial effects require a small granularity in the decomposed physical domain. Other unsolved issues are the frequency of calling the partitioning process and the tolerated level of imbalance. Ideally one would like to reside to a kind of adaptive parameter tuning of these quantities.

The development of parallel DYTRAN is still continuing at MSC. In our case study we have left out the Lagrange part of the original simulation model (a rigid boat) considering that the a simulation with Lagrange/Euler coupling is still part of ongoing study.
5.4 Conclusions

In this chapter we have applied the P-CAM framework to two realistic simulation problems, a Cellular Automata based fluid flow model and a industrial parallel finite element simulation. For the first case the entire simulation problem has been implemented using P-CAM, enabling a parallel implementation including advanced parallel computation methods like dynamic load balancing. The dynamic load balancing feature of P-CAM has been applied as a separate migration decision process for a parallel finite element simulation. In this case the P-CAM framework implements a dynamic load balancing decision support process. Alternatively the entire parallel finite element simulation could be implemented using P-CAM, essentially in the same manner as the parallel LBGK simulation has been implemented. However, this requires an entire re-implementation of the original MSC/DYTRAN code.
Conclusions

A statically decomposed parallel process is compared with a dynamically load-balanced version. We have shown that the approach that has been chosen to deal with dynamic repartitioning, is able to resolve both heterogeneity in the application workload but also in heterogeneity in the computational resources. This is a consequence of the duality between both quantities, i.e., the workload determines the computational performance as well as the other way around.

Summarizing, we have shown that physical domain partitioning in a viable route towards dynamic load balancing of complex finite element simulations. A reusable parallel framework has been applied to model the computational dynamics of the parallel simulation and to steer the recreation process of simulation tasks. However, many open issues remain in this work. The granularity of the sub-division is a very important question. Large cubes induce smaller overhead in migration and communication but also less room for balancing the workload. Small cubes result in a lot of overhead but great flexibility in dynamic load balancing. The scale of the spatial extent in the specific simulation plays an important role in choosing the correct cube size. Small spatial extent require a small granularity in the decomposed physical domain. Other unresolved issues are the frequency of visiting the repartitioning process and the tolerated level of imbalance. Ideally one would like to proceed to a kind of adaptive parameter tuning of these quantities.

The development of parallel HYBRAN is still continuing at DLO. In our next study we have set out the knowledge part of the original simulation model in a more detailed consideration that the interaction with Lagrangian Euler coupling is still part of ongoing study.