EUROSIM '95
F. Breitenecker and I. Husinsky (Editors)

Modelling for Parallel Simulation: Possibilities and Pitfalls.

P.M.A. Sloot

Parallel Scientific Computing and Simulation Research Group, Faculty for Mathematics, Computer Science and Physics. University of Amsterdam, Kruislaan 403 1098 SJ Amsterdam, The Netherlands*

One of the key issues in designing new simulation models for parallel execution, or in the migration of existing models to parallel platforms, is the mapping of the application architecture to the parallel system architecture. In this mapping process we can easily loose track of the inherent locality present in the different architectures. In this paper we present an overview of these issues and examine, by means of new results from case-studies, consequences of the design and implementation choices for the various mapping processes. We will show that the potential for High Performance Simulation comes from a 'holistic' approach, taking into account all aspects from the application up to the underlying hardware.

1. BACKGROUND & INTRODUCTION

The need for High Performance Computing (HPC) is still growing. Although originating from the physical sciences, engineering and the study of complex systems, an increasing variety of new applications are emerging, such as HPC in financial modelling, transaction processing, image analyses, etc. It is this strong application pull that has motivated national and international research programs to stimulate research, education, and hardware development especially in the field of parallel computing and parallel systems design. Financial support comes mainly from resources allocated by the national and international governments, for instance in the USA the HPCC bill aims at a 1.10^9 dollar budget for HPCC related research *per year*. In Europe the EC funds the 4th framework program by 300.10^6 ECU over 5 years in addition to many national and intergovernmental funding.

If we take a closer look at the type of applications that are addressed by these initiatives, we find that the majority deals with large scale simulation problems, ranging from weather prediction to vehicle dynamics. It is clear that only massive parallel processing (MPP) can provide the extreme processing power required by these grand engineering and simulation applications. We should not restrict ourselves to MPP but rather consider also the possibilities arising from *heterogeneous* parallel computing. Here we can think of large clusters of high-end workstations of different architecture or combinations of MPP systems with Vector Super computers. A nice example of this last category comes from the results presented by Paul Messina (Caltech) at the HPCN95 Europe conference where he reported on a chemical reactor simulation that took 18 hrs on a Cray C90 (vector super) or 16 hrs on the (parallel) Delta Touchstone. However, when the problem was divided over the Cray for the eigenvalue calculation, and the Delta for the dense matrix calculation the *mixed* simulation only took 4 hrs! This notion of the relevance of simulation in heterogeneous parallel processing is also prominent in the new USA HPC initiative (following the 1982 Lax Report and the 1989 Al Core HPCC initiative) the so-called: Acceleration Strategy for Computational Initiatives (ASCI), a 10 year (140 M$/year) collaboration between the Department of Energy and the Industry. In this initiative simulation studies such as multi-physics, 3D-geometry's, nuclear plant simulation, complex systems

* Electronic Mail: peterslo@fwi.uva.nl, URL: http://www.fwi.uva.nl/fwi/research/vg4/pwrs/

and modelling are mentioned explicitly, together with software and hardware programmes for distributed and massive parallel computing.

If we look at it from the bright side, we find that within a very short time span the HPC initiatives have had an enormous impact on the computer society in general and the simulation society specifically, resulting in many new scientific journals, new HPC research centres (some of which are organised in the European HPCnet E.G. [1]), new educational programmes and new hardware initiatives (in Europe for instance: Parsys' Supernode, Parsytec's PowerStone, Meiko's CS2) that might eventually lead to TeraFlop performance or even PetaFlop performance [2].

There is however a dark side if we take a more realistic view at what can be obtained and what is actually being done. Although a strong emphasis in HPC programmes has been on stimulating industry to participate (for instance through the European 4th framework programme and the Europort projects) hardly any industry really uses this technology in their core products. This is a sign on the wall that HPC is still not accepted by industry and indicates the immature character of the technology.

Some reasons for this relatively slow take up of technology might be:

- It takes al lot of effort to outperform a vectorized code running on a vector-super.

- Most applications that are successfully parallelised run in an embarrassingly parallel mode and can easily be outperformed by a large pool of independently (sequentially) working workstations.

- There are hardly any formal models beyond the classical PRAM and CSP models for parallel computing. The models that do exist won't support the distributed memory MIMD type of machines (PRAM) or hamper the description of the algorithm (CSP).

- There is still little to no understanding of the differences in- and consequences of- parallel versus sequential computing. Much work is to be done on the numerical aspects of parallel algorithms and the implicit differences in parallel and sequential solvers (for instance it is still an open question whether parallel Simulated Annealing algorithms probe the same phase space as sequential SA's [3]). Even worse are these cases where, thanks to the capacity of the parallel systems, too large a problem is studied without quantitative understanding of the numerical consequences [4].

- Computer Science has failed in developing a quantitative understanding of simulation in a distributed environment. New potential models that take up the notion of complex system theory are still in its infancy [5,6,7].

- Even if we assume for a moment that we do have large TeraFlop (or even PetaFlop) machines, than we still lack good computational models that fully exploit the parallelism present in applications and that take care of well load balanced parallel execution.

Perhaps it is time for the scientific -simulation- society to reconsider the various design and implementation stages for applications that require advanced parallel systems.

Therefore throughout this paper we ask ourselves the following questions: What do we want? What do we get? What *can* be obtained? What *should* we have wanted? A way to attack these issues is by going through the computer experiment step by step. In this paper we have a less ambitious approach and take a helicopter view and zoom in on some of the crucial decision moments in the modelling and simulation cycle, by identifying 'hot spots' in the modelling phase and the model execution phase, each with its own possibilities and pitfalls.

The modelling phase:

The first step to simulation is the development of an abstract model of the real-world system under study [7b]. Strange enough there are hardly any formal methods of modelling which supports completeness, correctness and efficiency. The lack of such methods might harm the success of this research field significantly. There are specific journals and internet sites dedicated to the compilation

of (the consequences of using) insufficient, incorrect and inefficient models. Examples are flight simulation experiments where the underlying simulator worked perfectly, but where deficiencies in the CAD/CAM model resulted in catastrophic misinterpretation, or dangerous designs of backseat airbags resulting from erroneous simulation studies etc.

Solvers form the kernel of the modelling phase. Here we refer to (mathematical) methods that translate the real-world systems into a computational specific model. A rough distinction can be made between solvers for *discrete (event) systems* and solvers for *continuous systems*. Solvers for discrete event systems are event-set algorithms, which ensure that the events occur in the proper order and at the proper time. Conventional solvers for continuous systems are finite difference, finite element/volume, and a large class of linear algebra solvers, where the model system is tracked in time. Of special interest here are new results obtained with algorithms mimicking nature's processing power. We have tossed the term 'Natural Solvers' for these type of algorithms, examples are: Genetic Algorithms, Artificial Neural Networks, Simulated Annealing, etc. Their specific relevance stems from the natural way in which they translate the real-world system into an abstract algorithm, moreover they are highly adaptive and often easily scalable.

If we model for parallel execution we must understand where in our abstract model locality in space or time (or space time for that matter) is present and how to exploit this locality in the computational model. Again no quantitative methods exist although interpretation of parallel computing as the mapping of one complex system (the application) onto another complex system (the parallel machine) may prove to be a fruitful alternative to formalise parallel computing [7, 8], this is however beyond the scope of this paper, some recent ideas will be discussed elsewhere [9].

The model execution phase:

This phase essentially contains all the elements necessary for the model *execution* on some -parallel- platform. Here we concentrate on the mapping of the different solvers to the machine architecture. Since the type of problems we are interested in are computationally very demanding, much research effort is going on in the efficient use of modern architectures (parallel computer systems) for simulation.

The computation specific model consists of representing the derived conceptual model into a *language* that can be implemented. One of the things that often goes wrong in these stages is that we tend to confuse code with a abstract model or—even worse—with the real-world system itself. We have 'fallen in love with our model' [10], this is a situation we should be prepared for and try to avoid at all costs. One way to maintain a critical attitude is to carefully design and test sub stages in the modelling and simulation cycle and to add primitives to the simulation that constantly monitor for 'unrealistic' results.

In parallel execution we specifically need to address the mapping of the abstract model onto the underlying hardware. Here we should distinguish between distributed and parallel computing since each paradigm has its own consequences for the conservation of locality present in the model.

Finally after identification of the space time locality in our problem and the correct (i.e., with conservation of locality) mapping onto a parallel platform we need to investigate consequences of architecture peculiarities on model execution. Figure 1 depicts how the various phases in modelling and simulation cycle fit together.

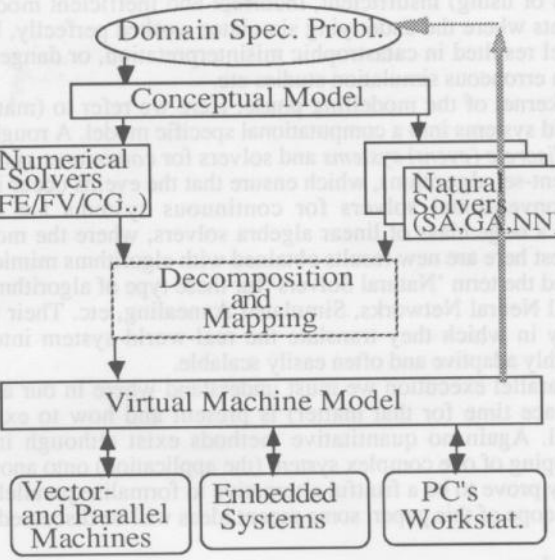

Figure 1: Overview of conceptual steps in modelling and simulation cycle

From the domain specific problem (for instance turbulent flow around a dynamically changing object) we need to abstract a conceptual model (e.g., a theory of liquids) and annotate all the possible instances of locality (e.g. flow behaviour in the vicinity of the object is largely independent of the behaviour at the flow source). Next we cast this conceptual model into a computer specific model, basically this comprises the solver that allows for an algorithmic description of the conceptual model (e.g. define Navier Stokes equations and choose relaxation methods to solve the discretised PDE's). Finally we need to implement the solver for a parallel platform. Here we must consider the interconnectivity and locality of both the computer specific model and the hardware (e.g. can we map the loosely coupled program into a SPMD model to be executed on a distributed memory MIMD architecture?).

It is obvious that we need to make many choices in these mapping processes, a thorough discussion of all these aspects is certainly beyond the scope of this paper. Here we only address some aspects, new insights and new tools that we feel are relevant to this field of HPC and that might guide the reader in designing and implementing his parallel simulation.

In section 2 we will discuss the application architecture and promote an alternative computational model specifically suited for parallel simulation. In section 3 we will identify the differences in distributed versus parallel execution on distributed memory systems, and suggest a possible solution to the implicit load imbalance problem. Finally, in section 4, we briefly mention consequences of slightly different machine architectures and ways to predict the behaviour of code execution in different environments.

2. APPLICATION ARCHITECTURE

With the type of complex simulations we want to address in High Performance Simulation we need to rethink the (mathematical) structure of the applications. Ed Masi from Intel once formulated it in this way: 'With horse driven carts people never worried about aero-dynamics...'. Basically there are two fundamental issues we should take into account, since we are not dealing with horse driven carts anymore:

1] Can we simulate real physics on a computer or are we mimicking?
2] What is the underlying structure of the application and can it be mapped efficiently to a parallel/concurrent architecture.

Aspects related to the first question were beautifully discussed in a paper by Richard Feynman [11]. We can rephrase these questions to: Does nature itself behave as a Universal Computer and can we build systems to execute this universal computer. The relation between physics and universal computing has been addressed extensively by Fredkin [12] and Wolfram [8] but is still an open question. In a forthcoming paper we will discuss in detail some new idea's in this research area [9]. Here we are mainly concerned with the second part of the question namely: 'What kind of models can be mapped efficiently to parallel systems'. In this section we will explore an example where we mimic nature to build a 'natural solver' for parallel complex growth models in dynamically changing environments.

2.1 Natural Solvers

A very promising class of solving techniques can be identified by 'natural solvers', these techniques have in common that they are inspired by processes from nature and preserve domain properties in the mapping from solver to the computational model. Important examples of natural solvers are Genetic Algorithms (inspired by the process of natural selection), Simulated Annealing (inspired by the process of cooling heated material which converges to a state of minimal energy), the Lattice Boltzmann method (a many particle system with a macroscopic behaviour that corresponds to the hydrodynamic equations), and artificial Neural Networks (inspired by the transmission of signals in the brain). In an 'non-natural solver', as for example finite differencing, a number of approximations and abstractions are involved in the simulation of the real physical phenomena, as for example diffusion and flow. In the simulation model this process of approximation and abstraction obscures the explicit information on the physical phenomena and as a consequence violates the 'domain conservation'. Even worse, the possible implicit parallelism of the problem becomes completely indistinct in the abstraction process. Traditionally, methods as finite differencing are widely used to simulate physical phenomena. In parallel computing especially the class of natural solvers is a very promising approach, since the physical characteristics of the original physical phenomenon remain visible in the solving method and the implicit and explicit parallelism of the problem remain conserved.

2.2 Case study: Diffusion Limited Aggregation

Many growth phenomena, for example the growth process of a bacteria colony, viscous fingering, electric discharge patterns and growth forms of electro deposits, can be simulated with one model: the Diffusion Limited Aggregation model [13]. At the heart of all these growth patterns there is one Partial Differential Equation,

$$\nabla^2 c = 0 \tag{1}$$

the Laplace equation, which describes the distribution of the concentration c , pressure, electric potential etc. in the environment of the growth pattern. First we will discuss the numerical solver for such a system, then the natural solver and finally parallelisation aspects for the natural solver.

The numerical solver: Finite differencing

This equation can be solved numerically and a DLA cluster can be constructed using the nutrient distribution over the lattice. The cluster is initialised with a seed and the following boundary conditions are applied: $c = 0$ on the cluster itself and $c = 1$ at the nutrient source, which in itself may be circular, linear etc. The cluster is constructed using the following rules:

1: solve the Laplace equation (Eq.1), using the boundary conditions.
2: new sites are added to the cluster are added to the cluster with probability p (Eq.2).
3: goto 1

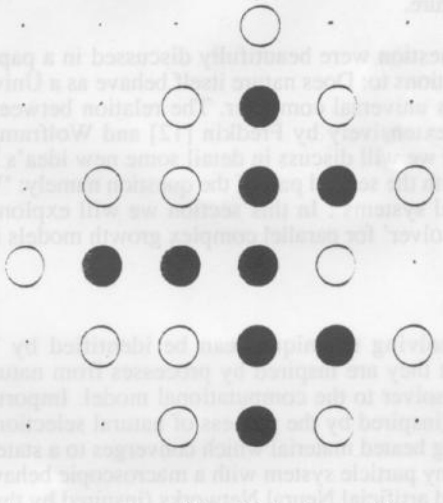

Figure 2: First steps in the construction of the DLA-cluster. Sites which are part of the cluster are visualised as black circles, sites which are possible candidates to be added to the cluster in next iteration steps are indicated with open circles.

The probability p that a perimeter site (the sites indicated with an open circle in Figure 2 with index k will be added to the DLA-cluster (black circles in Figure 2) is determined by

$$p(k \in \circ \rightarrow k \in \bullet) = \frac{c_k}{\sum_{j \in \circ} c_j}, \text{ where } c_k = \text{concentration at position } k\,. \qquad (2)$$

The sum in the denominator represents the sum of all local concentrations of the possible growth candidates (the open circles in Fig. 2). The Laplace equation can be solved, using the boundary conditions mentioned above, by finite differencing and the successive over-relaxation method:

$$c_{i,j}^{n+1} = \frac{\omega}{4}\left(c_{i-1,j}^{n+1} + c_{i,j-1}^{n+1} + c_{i+1,j}^{n} + (1-\omega)c_{i,j}^{n}\right) \qquad (3)$$

In this method the new local nutrient concentration in the lattice $c_{i,j}^{n+1}$, at a site with lattice co-ordinates i,j, is determined in an iterative procedure which converges as soon as the difference between the new and old local nutrient concentration $\left(c_{i,j}^{n+1} - c_{i,j}^{n}\right)$ is below a certain tolerance level.

The ω in Eq. 3. is the over-relaxation parameter, which in general lies within the range $1 \leq \omega \leq 2$. After many construction steps this procedure results in a DLA-cluster as shown in Figure 3.

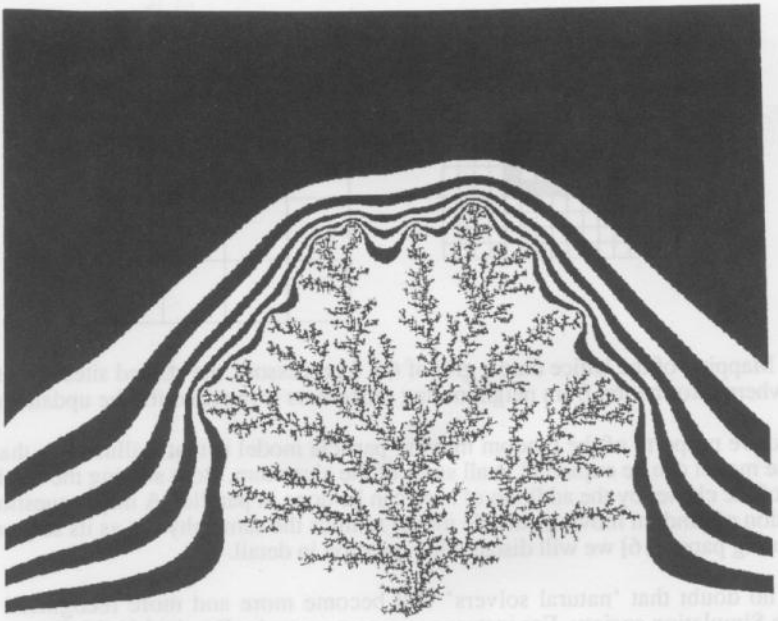

Fig. 3: DLA-cluster generated on a 1000 x 1000 lattice using a linear source of nutrient, located at the top row of the lattice.

The natural solver: The random moving particle model

An alternative method to construct the DLA-cluster is a probabilistic cellular automaton which resides on a square two-dimensional or three dimensional lattice. The growth pattern can be constructed using the following Monte Carlo approach: The first step in the construction is to occupy a lattice site with a seed. After that, particles are released from a source which might be circular shaped (using the seed as a centre) at a large distance from the seed. The particle starts a random walk, the walk stops when the particle leaves the circle or reaches a perimeter site of the seed and sticks. Then more random walkers are released from the source and are allowed to walk until the distance with respect to the cluster with occupied sites becomes too large or it reaches a perimeter site, neighbouring to one of the previous particles, and it sticks.

When this procedure is repeated many times a similar irregular growth pattern as shown in Fig. 3 is generated. It can be demonstrated that in this Monte Carlo method the underlying Laplace equation is correctly solved [13].

The parallel implementation of Diffusion Limited Growth

Modelling and simulation of Diffusion Limited Growth, especially in the 3D case, is computationally very expensive. The development of parallel growth models, especially in the case of DLA is not straightforward (see also references [14, 15]).

The computationally most expensive step, step 1 solving the Laplace equation, of the numerical solver can be done in parallel. In a parallel implementation of Eq. 3 using SOR, the update order of the lattice has to be taken into account. When $c_{i,j}^{n+1}$ is updated in the parallel implementation and is located at the border of the processor (see Fig. 4), its neighbours should be in the correct state (n or $n+1$).

A parallel implementation, with a correct update order, can be made using the checkerboard strategy [5]. In this method the sub-lattice on one processor is subdivided into four blocks. In the parallel SOR update first all red regions are updated, followed by the black regions. Step 2 in the numerical solver, adding sites with probability p is an implicit sequential step.

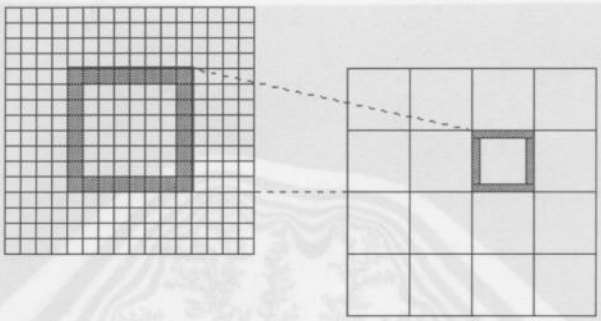

Figure 4: Mapping of the lattice onto a grid of 4 x 4 processors, the shaded sites are situated in a region where information from neighbouring processors is required for the update (after [5]).

The attractive property of the random moving particle model is that it illustrates that the locality present in the model can be exploited in all steps of the algorithm. Both solving the Laplace equation and growth of the cluster by the addition of sites can be done in parallel. A major question is if such a parallel version of random moving particle model mimics the same physics as its sequential version. In a forthcoming paper [16] we will discuss this question in detail.

There is no doubt that 'natural solvers' will become more and more recognised in the High Performance Simulation society. For instance, a recent paper by Dzwinel [16b] discusses the use of Molecular Dynamics to pattern recognition on parallel systems. They concluded that 'the inherently sequential problem, i.e., the global minimum search for the multidimensional criterion, changes for particle dynamics which is—in turn—inherently parallel'.

3. DISTRIBUTED VERSUS PARALLEL EXECUTION

Given a chosen solver and a parallelization strategy we must concentrate on the underlying hardware on which the actual model execution takes place. First we note that in academia and industry interest is renewed in using clusters of high performance workstations for HPC tasks rather than tightly coupled parallel monoliths. There are several reasons for this.

Monoliths are expensive and dedicated, whereas clusters are relatively cheap and general purpose. In addition the 64-bit RISK technology boosts the performance per workstations-node to figures comparable to dedicated monolith nodes (e.g., PowerPC and the Alpha chip). Moreover, new software technologies provide better programming environments (e.g., heterogeneous PVM/MPI), resource management tools (e.g., CODINE, CONDOR). The major reason however seems to be the increasing network bandwidth supporting fast and reliable communication between nodes in a cluster environment, installations with 100, 150 and some even with 622 Mbit/s ATM-SONET Local Area Networks were reported recently by Tolmie at the HPCN95 Europe conference in Milan [17]. In the upcoming Supercomputing '95 conference an experimental networking project will be announced (the so-called 'I-way') aiming at a 622 Mbit/s network of 40 major institutes in the USA [18].

One of the major questions remaining is what the consequences are for the parallel applications that are highly tuned to tightly coupled parallel systems. This is not merely a question of portability, but also one of more fundamental differences in execution behaviour, especially when issues as load balancing are considered. The major differences stem from the fact that cluster computing implies a *dynamic* (and often heterogeneous) computer resource as opposed to the *static* (homogeneous) resources available in monolith computing.

In this section we identify this problem of load balancing in a parallel cluster environment, present some new results and suggest a possible way out [19, 20].

3.1 Introduction

Loosely coupled parallel systems require new programming paradigms and environments that provide the user with tools to explore the full potential of the available distributed resources. Although such cluster computing systems give the user access to large amounts of processing power, their applicability and efficiency is mainly determined by environmental changes like variation in the demand for processing power and the varying number of available processors. To optimise the resource utilisation under these environmental changes it is necessary to migrate running tasks between processors, i.e. to perform dynamic load balancing.

Consider a Finite Element problem where we want to simulate the forging of steel plates. After a straightforward domain decomposition (for instance using bisection methods [21, 22]), we can map the different domains onto separate processors. If the hardware system is homogeneous and monolithic (implying single user), than the simulation will run balanced until completion. The data structure, data decomposition and the assumed topology remain unchanged: We have mapped a static resource problem to a static resource system. If however we use a cluster of multitasking/multi-user workstations we run into problems since the processing capacities per node may change randomly: We have mapped a static resource problem to a *dynamic* resource system, resulting in a potentially complete unbalanced execution. Things can even get worse when we consider the mapping of a dynamic resource problem onto a dynamic resource machine. For instance consider a simulation of a car crash, where during simulation the load changes due to changes in the geometry of the object being modelled [23].

A solution to these problems could be an intelligent system that supports the migration of tasks from a parallel job from overloaded nodes to under loaded nodes at run-time, thus supporting dynamic run-time load balancing without interference from the programmer.

3.2 Case study: Dynamic PVM

Here we describe a scheduling mechanism for the Parallel Virtual Machine (PVM) [24] that supports automatic load balancing for parallel tasks running on loosely coupled parallel systems. The enhanced system is called DynamicPVM. The choice for PVM as the basic parallel programming environment is motivated by the fact that PVM is the most widely used environment to date and is considered the de facto standard. The process migration primitives used in DynamicPVM were initially based on the checkpoint-restart mechanisms found in a well established global scheduling system [25].

PVM: Runtime support system for parallel programs.

PVM provides primitives for remote task creation and Inter Process Communication (IPC). It supports both point-to-point and global communication primitives. Tasks are assigned to available processors using a cyclic allocation scheme. Jobs are placed statically, i.e. once a job is started, it runs on the assigned processors until completion. Each processor in the PVM pool is represented by a daemon that takes care of task creation and all IPC to and from tasks running on the processor. To enable the use of heterogeneous processor pools, messages are encoded using an external data representation. With the current PVM version direct IPC between two PVM processes, without interference of the PVM daemons is supported, thereby enhancing communication performance.

CONDOR Runtime support for job scheduling

The CONDOR system stems from the observation that many of the constantly increasing number of workstations in academic and industrial institutions are lightly loaded on the average. Most workstations are intended for personal usage, which has a typical activity pattern where machines are only used for a small part of the day. As a consequence many computing cycles are unused during the day. Typical Figures of large pools of workstations have a mean idle time of 80% [25]. To address this problem, CONDOR implements a global scheduling based on dynamic load balancing by job migration. CONDOR monitors the nodes in its pool by keeping track of their load. New jobs are spawned on lightly loaded nodes and jobs from heavily loaded machines can be migrated to less loaded ones. When interactive usage is detected of a workstation, all jobs can be evacuated from that workstation in order to retain the sympathy of the workstation's owner. To implement this job migration CONDOR creates checkpoints on a regular basis, which can be restarted on another

machine. Using CONDOR, it is not possible to migrate jobs consisting of co-operating parallel tasks since it does not provide any support for IPC primitives. Combining PVM with an extended version of CONDOR's checkpoint-restart facility makes it possible to apply global scheduling to parallel tasks.

DynamicPVM: Runtime support system for job scheduling parallel tasks

In DynamicPVM we have added checkpoint-restart mechanisms to the PVM environment. Most of PVM's features are compatible with the checkpoint-restart mechanism we use and can be incorporated in DynamicPVM without problems. The Inter Process Communication is an exception to this rule. We present a protocol that ensures that no messages get lost whenever a task is migrated. This protocol involves a special role for the PVM daemon that initiated the computation, the Master daemon. We also included an extension to the PVM IPC routing mechanism in order to redirect messages to tasks that are migrated. DynamicPVM's task migration facility consists of four principal components:

- A global scheduler that initiates job migration.
- Task check pointing, including a method to indicate checkpoint save moments.
- Actual task migration.
- Task restart and updating of routing tables to reflect the task's new location.

These components are briefly described below [see also 19 and 20].

Task check pointing

In order to migrate a process, a dump of the process' data and state, together with some additional information to recreate the process, has to be made. We have implemented two different strategies for dumping this information: direct and indirect. Using direct check pointing, the host where the checkpoint is migrated from opens a TCP connection to the destination host and writes the process' data and status to the destination host. With indirect check pointing, a dump of the process' state and data is made to a shared (NFS-mounted) file system. In this way, the process can be restarted by a machine at a later stage. Since direct check pointing involves only one transfer of the migrating process, compared to two transfers (write/read) when using NFS it is approximately twice as fast. Check pointing *co-operating tasks* introduces new conditions as compared to check pointing stand-alone tasks. For instance, checkpoints should be avoided when a task is communicating with another task. To safely checkpoint DynamicPVM tasks, we introduced the notion of a *critical section* and embed all IPC operations in such sections. Check pointing is prohibited whenever the task is in a critical section; check pointing can only take place when the task is not participating in a communication operation.

Task migration

The main demand on the DynamicPVM task migration facility is transparency, i.e. to allow the movement of tasks without affecting the operation of other tasks in the system. With respect to a PVM task selected for migration, this implies transparent suspension and resumption of execution. With respect to the total of co-operating PVM tasks in a job, communication can be delayed due to the migration of one of the tasks. The first step of the migration protocol is to create a new, empty, process context at the destination processor by sending a message to the daemon representing that node. Next, the Master-Daemon updates it's routing tables to reflect the new location of the process.

The task to be migrated is suspended and messages arriving for that task are refused by the task's original daemon. Such messages are queued by the sending daemon, to be processed after the new location has been broadcasted. In the next phase, the Master-Daemon broadcasts the new location to all nodes, so that any subsequent messages are directed to the task's new location. The last phase is the actual migration of the process. As stated in the previous section, there are two strategies implemented and the user can choose the appropriate mechanism.

Task Restart

The newly created process on the destination processor is requested to restart the checkpoint. If direct check pointing is used, it opens a TCP socket and waits for the check pointing task to begin transmission of the checkpoint. Using indirect check pointing, the task opens the checkpoint file and reads the checkpoint from disk. After the checkpoint is read, the original state of the process is restored (data/stack/signal mask/registers) and the process is restarted with a long jump. Any messages that arrived during the checkpoint-restart phase are then delivered to the restarted process.

What did we learn from this case study?

We have implemented DynamicPVM on a cluster of IBM RS/6000, AIX32 machines and cluster Sun workstations [19, 20], operating under SunOS4 and Solaris. Experiments indicate that DynamicPVM has very efficient migration protocols (migration linear to the size of the program and no additional latencies).

Clearly, distributed computing contaminates the laboriously obtained load balance in the parallel simulation. However by identifying locality in terms of separate tasks, of equal computational complexity, and by using a system like DynamicPVM we can still obtain load balance in a dynamically changing resource environment and preserve locality in our application.

4. HARDWARE CONSEQUENCES

4.1 Introduction

The development of parallel solvers implies a lot of decision making. As discussed in the previous paragraphs a manifold of choices can be made on the parallelization method, data distribution, communication topology, the runtime support system etc. The best choices will depend on the specific application but may very well also depend on the target computer system. For the HPC-simulation developer it is not always clear which parameters will dictate the performance of his simulation. For sequential programs for instance, one might recognise that some type of instructions are executed more often than others. Hence, the performance of the program would change when these type of instructions are faster executed on a processor. For parallel applications this becomes more complex, since the machine parameters involved are interdependent. A complex system in which adding faster communication lines or using faster processors is not necessarily much better for the application execution time.

Therefore in order to obtain insight in the consequences of implementing the application on different architecture's, we need new methods and tools to guide us through the various decisions and to assist in the prediction of performance behaviour. In the next section we briefly outline such a system that has recently been developed by our group within a European project [26a-c].

4.2 Case study: Virtual machine simulation

In the field of parallel computation, there are hardly any analytical methods to guide decisions faced by developers for both hardware and software design. Instead, *simulation* must be used to study the performance of the hardware. For parallel applications additional performance metrics need to be extracted from test runs on a parallel platform (such as: throughput, latency, interconnectivity, node asymmetry, etc.).

There are many approaches in parallel hardware simulation most are based on some form of discrete event simulation where processor cache and memory behaviour are studied through stochastic traces [27], determined by for instance process activity graphs [28]. A major drawback of these simulators is that they are extremely complicated, often dedicated to one type of architecture and require substantial simulation time for - in the eye of the engineer - modest results. Especially when parallel computing is involved the number of experimental parameters to be studied becomes very large. Moreover the engineer would like to perform 'what-if' experiments in order to trace down communicational or computational bottlenecks on different architecture's. Therefore a light weighted highly adaptable simulation system is required.

An approach would be to abstract relevant information of the application and derive a time complexity formula that incorporates all the significant information on computation, communication and data dependency. We can do the same for the underlying hardware by building a machine database that provides a generic machine description which is able to roughly describe the parameters which influence the performance of a computer. This in order to be able to change parameters and create what-if machines so that the performance of an application on different machines can be investigated. Of course, abstracting the machine into machine parameters which are averaged time estimates for operations has its toll. The accuracy will be reduced and carefully selected benchmarks are needed to gain reliable averages for the machine parameters.

Finally the execution time of an application is determined automatically by the machine parameters and the abstracted time complexity formula. If a time complexity formula is used in which the performance parameters of the machine are kept abstract, an interactive performance analysis becomes feasible. As an example in Figure 5 we have shown a simulation of a Molecular Dynamics code.

The code is analysed by the simulation toolset and simulated on a hypothetical workstation. For instance by a simple modification of the parameters cache behaviour could be studied. In this case the experiment clearly predicts irregular behaviour on in a parallel cluster system with CPUs with only small differences in caches.

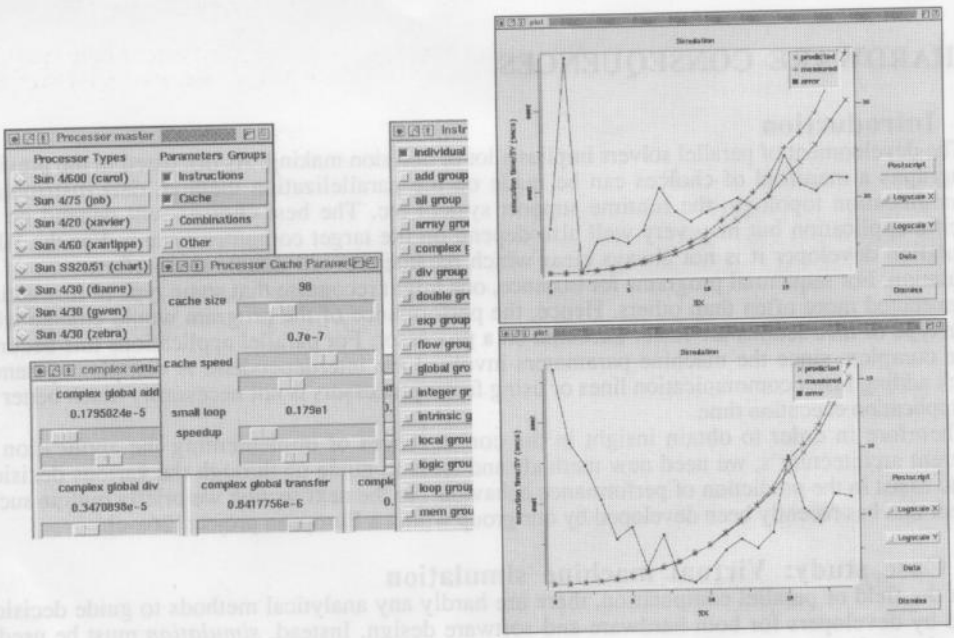

Figure 5: The left picture shows the graphical user interface allowing instantiation of the machine parameters. The upper-window on the right shows the prediction before, and the lower-window after, changing the instruction cache size by only 5 instructions.

This approach of simulating an abstraction of an application on an abstraction of a parallel machine is implemented in the Esprit-III project CAMAS [26a-c], where old dusty deck F77 programs or new F77 with PVM/MPI can be fed into a translator which produces the time complexity and determines the dependencies between all the different parameters in the formula. This time complexity formula can than be simulated on a chosen architecture, producing estimates of the running time of a program for variations in the factors. It was, among others, used in the evaluation of a very large ($> 10^5$ of lines of F77 dusty deck code) car crash worthiness simulation [23].

4. SUMMARY AND CONCLUSIONS

In this paper we have discussed current issues in High Performance Simulation (HPS). We argued that parallelism adds a complete new dimension to the expected behaviour of model execution demanding new paradigms and new software techniques. We have taken a top down helicopter view from application modelling via runtime support systems to the machine level, and identified pitfalls and possible solutions. Clearly we could only touch upon some of the myriad of aspects involved. For instance we have discarded aspects related to Languages (the upcoming High Performance FORTRAN standard), Compiler architecture interactions and aspects of (distributed) Operating Systems.

It seems clear that the best approach to HPS is an holistic view fully incorporating all different levels in the mapping of application architecture to machine architecture.
We believe that HPS has just started and will become a very important research and technology area. Due to the complexity of the systems being simulated in HPS, new tools and methods need to be developed (and used!) that abstract locality information from the application and support in the mapping to HPC systems. The concept of natural solvers in conjunction with hybrid parallel implementation supported by dynamic resource management systems seem to be a very promising way to go. Some pilot results in this area pursued at the University of Amsterdam were discussed in this paper.

Once we can fully exploit the parallel technology we might even expect in the near future real-time simulations guiding real-time complex experiments, a research technique sometimes referred to as 'living simulations'.

ACKNOWLEDGEMENTS

The ideas presented here are -among others- a result of various projects within the Parallel Scientific Computing and Simulation group at the University of Amsterdam. Benno Overeinder and Joep Vesseur are acknowledged for the discussions on DynamicPVM, Jaap Kaandorp on the Natural Solvers concepts, Arjen Schoneveld on the Universal Computing issues and Berry van Halderen, Jan de Ronde and Bob Hertzberger on architectural and virtual machine simulation aspects. Their help in preparing this document is gratefully acknowledged.

The research was partly conducted under funding of the EC Esprit III programme (CAMAS Grant: NB 6756) and the Dutch foundation for fundamental research in material sciences (FOM: Grant: Massive Parallel Computing in Many Particle Simulations).

REFERENCES

1. HPCnet is a Esprit European network of Excellence initiated by the University of Amsterdam and the University of Southampton. URL: http://hpcnet.soton.ac.uk/

2. T. Sterling, P.C. Messina and P.H. Smith 'Enabling Technologies for Peta(FL)OPS Computing,' California Institute of Technology report: Send e-mail requests to techpubs@ccsf.caltech.edu.

3. J. M. Voogd, P. M. A. Sloot, and R. van Dantzig, 'Comparison of vector and parallel implementations of the simulated annealing algorithm,' Future Generation ComputerSystems, special issue HPCN '94 in press (1995).

4. Some of this problems were addressed by J. Dongara at HPCN95 Europe (Milan). An example is the simultaneous solution of a large systems of PDE's that could run in parallel but where the numerical aspects are not fully understood or simply indicate inevitable instabilities. Since this

seems to be a fundamental aspect, Dongara even promoted the standardization of 'limited stable' BLAS libraries!

5. G.C. Fox, M. Johnson, G. Lyzinga, S. Otto, J. Salmon, and D. Walker, 'Solving problems on concurrent processors, Volume I, General techniques and regular problems,' Prentice-Hall International Inc., London (1988)

6. G.C. Fox, R. D. Williams, and P. C. Messina, 'Parallel Computing Works,' Morgan Kaufmann Publishers, Inc. (1994).

7. G.C. Fox and P.D. Coddington, 'Parallel Computers and Complex Systems,' NPAC Technocal report SCCS-370b June (1994).

7b. P.M.A. Sloot, 'Modelling and Simulation,' in Proceedings of the 1994 CERN School of Computing, (Sopron, Hungary), pp. 177-226, ISBN: 92-9083-069-7 (1994).

8. S. Wolfram, 'Cellular Automata and Complexity: Collected Papers,' Addison-Wesley, ISBN: 0-201-62716-7 (1994).

9. P.M.A. Sloot and A. Schoneveld, 'Undecidability and Intractability in Complex Systems,' manuscript in preparation.

10. F.E. Cellier, 'Continuous System Modeling,' New York: Springer-Verlag (1991).

11. R.P. Feynman, 'Simulating Physics with Computers,' International Journal of Theoretical Physics **21** (6/7) (1982).

12. E. Fredkin and T. Toffoli, 'Conservative logic,' International Journal of Theoretical Physics **21** (6/7), 219-253 (1982).

13. L.M. Sander, 'Fractal growth processes,' Nature **322**: pp 789-793 (1986).

14. J. Machta, 'The computational complexity of pattern formation,' Journal of Statistical Physics **70** (3/4): pp 949-967 (1993).

15. J. Machta and R.Greenlaw, 'The parallel complexity of growth models,' Journal of Statistical Physics **77**: 755-781 (1994).

16. A. Schoneveld, J.A. Kaandorp, and P.M.A. Sloot, 'Parallel modelling and simulation of growth patterns,' (in prep.).

16b. W. Dzwinel and J. Blasiak, 'Pattern Recognition via Molecular Dynamics on Vector Supercomputers and Networked Workstations,' in Lecture Notes in Computer Science, High Performance Computing and Networking Conference, Springer Verlag, ISBN: 3-540-59393-4, pp 508 - 513 (1995).

17. High Performance Computing and Networking 1995, Milan, Italy, May 3-5 1995. See also proceedings in Lecture Notes in Computer Science, High Performance Computing and Networking Conference, Springer Verlag, eds L.O. Hertzberger and G. Serazzi. 1995, ISBN: 3-540-59393-4.

18. SuperComputing 95 will take place in San Diego Supercomputer Center (CA. USA) from 4 to 9 December 1995. For more information: URL: http://sc95.sdsc.edu/

19. J. J. J. Vesseur, R. N. Heederik, B. J. Overeinder, and P. M. A. Sloot, ``Experiments in dynamic load balancing for parallel cluster computing," in ASCI 1995 Conference Proceedings 1995, accepted for publication.

20. FGCS has scheduled a special issue on parallel versus cluster computing, it will appear near the end of 1995. For more information contact the author.

21. H.D. Simon, 'Partitioning of unstructured problems for parallel processing,' Computing Systems in Engineering **2**, No 23 pp 135-148, 1991.

22. Lecture notes on Parallel Scientific Computing and Simulation. A hardcopy of these lecture notes can be obtained from the author upon request.

23. G. Lonsdale, J. Clinckemaillie, S. Vlachoutsis, J. F. de Ronde, P. M. A. Sloot, N. Floros and J. Reeve, 'Crashworthiness simulation migration to distributed memory and MIMD machines,' in The 26th International Symposium on Automotive Technology and Automation, (Croydon, England), pp. 237-244, Automotive Automation Limited (1993).

24. V.S. Sunderham, 'PVM: A framework for parallel distributed computing,' Concurrency: Practice and Experience **2** (4) pp 325-339 (December 1990)

25. M.J. Litzkow and M. Livny and M.W. Mutka, 'Condor -- A hunter of idle workstations',' in proceedings 8th International Conference on Distributed Computing Systems, IEEE Computer Society, pp 104-111 (1988).

26a. CAMAS is an Esprit project with EC reference number: NB 6756. A description of the workbench can be found in 26b and 26c.

26b. J. F. de Ronde, P. M. A. Sloot, M. Beemster, and L. O. Hertzberger, 'The CAMAS workbench: Computer aided migration of applications system,' Future Generation Computer Systems **10**, pp. 305--308 (1994).

26c. P.M.A. Sloot and J. F. Reeve, *The CAMAS Workbench*. Technical Report CAMAS-TR-2.3.1.1 (March 1993).

27. A. D. Pimentel, J. van Brummen, T. Papathanassiadis, P. M. A. Sloot, and L. O. Hertzberger, 'Mermaid: Modelling and Evaluation Research in Mimd ArchItecture Design,' in Lecture Notes in Computer Science, High Performance Computing and Networking Conference, Springer Verlag ISBN: 3-540-59393-4, pp 335 - 340 (1995).

28. B.J. Overeinder and P. M. A. Sloot, 'Parallel Performance Evaluation through Critical Path Analyses,' in Lecture Notes in Computer Science, High Performance Computing and Networking Conference, Springer Verlag, 1995, ISBN: 3-540-59393-4, pp 634 - 639 (1995).

29. J. F. de Ronde, A. W. van Halderen, A. de Mes, M. Beemster, and P. M. A. Sloot, 'Automatic performance estimation of SPMD programs on MPP," in Massively Parallel Applications and Development (L. Dekker, W. Smit, and J. C. Zuidervaart, eds.), (Delft, The Netherlands), pp. 381-388, EUROSIM Conference on Massively Parallel Processing Applications and Development, Elsevier, North-Holland (1994).