Parallelisation of a Simulation Tool for Casting and Solidification Processes on Windows Platforms

Carsten Clauss, Silke Schuch, Rainer Finocchiaro, Stefan Lankes, Thomas Bemmerl
Chair for Operating Systems, RWTH Aachen University,
Kopernikusstr. 16, 52056 Aachen, Germany
{carsten, silke, rainer, lankes, bemmerl}@lfbs.rwth-aachen.de

Abstract

Since the beginning of computational engineering, the numerical simulation of physical processes is an essential element in the area of high performance computing. Thus, also the domain of metal foundry demands the computational simulation of casting and solidification processes. A popular software tool for this purpose has been developed by the RWP GmbH in Roetgen, Germany. This tool, named WinCast, is a complete software suite, which contains modules for pre-, main- and post-processing of simulation data sets. A core module of WinCast is TFB, which determines the chronological temperature distribution of a casting process based on a finite-element-method and a Gauss-Seidel solver. With the increasing demand for even higher precision of the simulation results on one hand, and a growing need for even larger data sets on the other hand, the parallelisation of this module became inevitable. In this paper, we present our work accomplished to parallelise the solving algorithm of this module. We have chosen an MPI based master-slave approach for compute clusters by using a self-developed MPI library for Windows platforms.

1. Introduction and Motivation

Traditionally, casting process development is an inherent trial and error workflow. Design and engineering departments develop the guidelines for building a component mainly based on previous experiences. Although the foundry aims to manufacture the components in the desired quality, the initial casting trials of new prototypes often fail due to design mistakes. Therefore, modern casting engineering demands a stronger interlocking of the design and casting process. This is because the later changes are made in this process chain, the more expensive they become.

With the emergence of computer technology, Computer Aided Design (CAD) has also entered the area of casting engineering. The evolution of capable CAD programs has made the drawing not only much easier for the designer, but also helps to check and optimise the resulting draft.

1.1. Computer Aided Casting Simulation

Nowadays, Computer Aided Design is not limited to sketching and drafting, but also helps to create analysable models as needed for computer based process simulation. Since industrial process simulation supports the identification of process issues and thus aids in the formulation of new design rules, it has become an inevitable part of the development chain.

Therefore, also the simulation of casting and solidification processes has become a necessity in founding industry. A prior simulation of the entire manufacturing process makes potential castability problems already become apparent in the earlier design stages. Thus, modelling and simulating the casting process by a capable software tool helps to reduce manufacturing costs and to increase yield and casting quality. [4] [5] [9]

In Section 2 of this paper, we introduce such a simulation tool, which provides modules for pre-, main- and post-processing for the simulation of casting and solidification processes.

1.2. The Need for Parallelisation

Since numerical simulation makes only sense if the obtained prediction is close enough to reality, a simulation tool must be capable of generating reasonable and revisable forecast data. That means, that the applied design software must provide as exact models as possible and that the simulating software must ensure precise data sets and stable solving algorithms.

In order to obtain even more exact simulation results, the problem sizes increase due to finer discretised structures and models. That is the reason why simulation software will forever exploit the available computational power: as processor speeds increase, the sizes of the problems to be solved increase likewise.

Hence, simulation software always works close to the limit of the computational power of contemporary CPUs. The only way to reach beyond this limit is to distribute the load on multiple CPUs and therefore to parallelise the problem. That is how it has been since the beginning of high-performance computing in conjunction with numerical simulation. And in future there will be an even bigger demand for parallelism since the end of the CPU cycle explosion seems to be near.

In the past, only computer centres possessed the computational resources to perform parallel computation on dedicated supercomputers. However, since high-performance computing has found its way from research into computational engineering, also smaller companies started to request for increasingly powerful systems, and thus for distributed solutions.

With the emergence of cluster systems in the field of high performance computing, even more companies could capitalise on those parallel systems. Besides dedicated compute clusters built from commodity hardware components, so-called *Office Grids* became more and more interesting especially for small and medium-sized enterprises (SME). These Office Grids, also known as *Network of Workstations* (NOW), consist of existing workstations which are connected by Ethernet. Especially considering the upcoming *Dual-Core* CPUs [2], there will be more computing power available in one single workstation. This additional CPU power can be utilised when either the workstation is under-worked with office applications or it is actually unused. Thus, users in SME who can not afford or do not want to buy a dedicated cluster, can benefit from parallel applications, too.

1.3. Parallelisation Strategy

Since both of the before-mentioned parallel architectures exhibit a distributed memory nature, intermediate data needs to be transferred as a message via the connecting network. Therefore, a parallelisation of the simulating software according to the message passing paradigm, in particular corresponding to the Message-Passing-Interface standard (MPI) [7], is strongly recommended here. In chapter 3 of this paper, we present our practical experiences gained in parallelising a code for casting process simulations according to an MPI based master-slave approach.

2. The WinCast Simulation Suite

The software suite *WinCast*, which is developed by the RWP GmbH in Roetgen, Germany, provides detailed computer aided simulation of casting and solidification processes. As its name indicates, WinCast is a GUI[1]-based software for Windows operating systems, which are the standard operating systems in casting industry.

The WinCast code is derived from the *SIMTEC* casting simulation suite for UNIX systems. SIMTEC in turn is rooted in a research project at the foundry institute of the RWTH Aachen University, which had the aim to develop a numerical simulation model using the finite element method. The WinCast / SIMTEC suite is made up of five main components:

- initially, an automatic mesh generator for finite element preprocessing,
- a module to simulate the mould filling process,
- the core module TFB to compute the temperature field characteristics,
- a module to calculate stress, distortion and strength, and
- finally, a postprocessing module to display the obtained results.

2.1. A Layered Finite-Element-Model

Similar to SIMTEC, the finite element preprocessing module of WinCast decomposes the three dimensional casting part by dividing it into horizontal slices. When modelling the resulting layers into finite elements and finally re-meshing the structures, the complete part is represented as a Finite-Element-Model (FEM) image in memory.

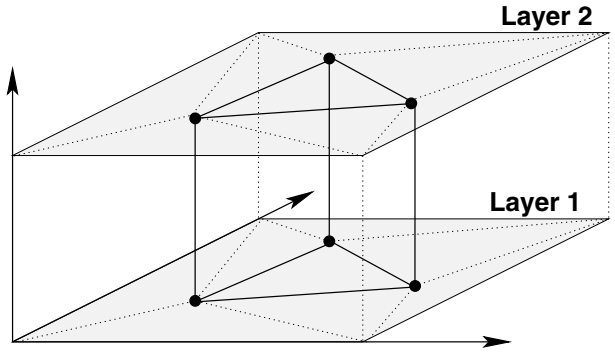

Figure 1. Prisms as Finite Elements

[1] Graphical User Interface

The basic element used in WinCast is a finite prism. Thus, the generated meshes are composed of layers of clustered wedges, as shown in Figure 1. The nature of prisms offers good clustering capabilities and the layered style allows to model even fine *onion skins* in order to manage high temperature gradients, as they frequently emerge especially in casting simulations.

In the WinCast model, three vertices of a prism always belong to an upper slice layer while the other three vertices likewise belong to a lower slice layer. Additionally, the mesh nodes can be spatially shifted in or out of a layer to create a more adapted model mesh. In spite of these possible modifications of the basic element, the rigid vertex assignment and thus the strictly layered style are internally retained. Therefore, it is not feasible to shift a mesh node into a next-but-one layer.

When parallelising the relating solver code, the data storage scheme following from this layered structure will again become a matter of importance, as pointed out in chapter 3.

2.2. Computation of the Temperature Field

The core task of a casting simulation program is to compute the heat distribution in the virtual mould at ascending time steps. In the case of WinCast, this operation is done by the TFB module.

Thereby, the basic heat transfer equation is stated by the law of heat conduction, also known as Fourier's law:

$$\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(\lambda \frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(\lambda \frac{\partial T}{\partial y}\right) + \frac{\partial}{\partial z}\left(\lambda \frac{\partial T}{\partial z}\right) + \dot{Q}$$

This partial differential equation states the temperature T as a function of time t and the Cartesian coordinates x, y, z, where ρ is density, c_p is specific heat at constant pressure and λ is the thermal conductivity. Thus, the time derivative of the temperature at a point is determined by the flow of heat quantity and a term describing heat sources and sinks.

When discretising the partial differential equation by using finite element analysis and considering the boundary conditions, the problem can be reduced to solve one linear equation system per examined time step. Since the resulting system is strictly diagonally dominant, the TFB module uses the applicable Gauss-Seidel method to solve this problem.

In order to provide symmetric results even for the special case of axially symmetric problem models, the solver in TFB iterates alternating through the layers. This proceeding is illustrated in Figure 2 for the time of one Gauss-Seidel iteration step through all layers.

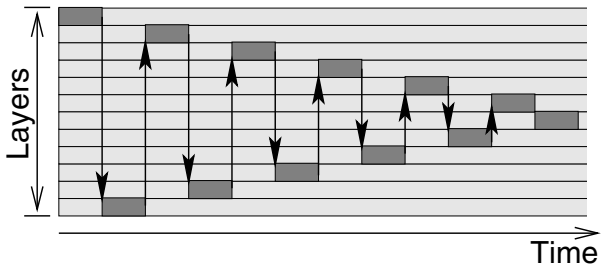

Figure 2. Iteration Order through the Layers

The number of iterations to be performed per time step results from the progression of average error and current variation. The abort criteria are met either when the average error and the maximal variation reach lower limits or when the current number of iterations exceeds a maximum. All these limits are adapted after each time step according to the time span between the last and the next state to be computed.

2.3. Internal Data Structures

Consider the vertex point P in Figure 3, which possesses the four neighbour points N_1 to N_4 in its own layer. In addition, the point P has got the immediate neighbours N_U, N_L and further four neighbours in the upper layer $(N_{U_1} - N_{U_4})$ and in the lower layer $(N_{L_1} - N_{L_4})$, respectively. However, if a vertex point is located on a material boundary, then additional neighbours may exist, so that the number of neighbours can vary.

The linear equation system to be solved provides one equation per unknown temperature of the discrete system. Since the value at each point only depends on the values of those nearest neighbour points, the matrix containing the equation system possesses non-zero elements only on the main and on those corresponding secondary diagonals.

Since most parts of the TFB code such as the solver and other core components are written in FORTRAN 77, most data is stored in static Fortran arrays. This affects especially the data of memory hungry FEM models. Therefore, the WinCast suite contains multiple binary versions of TFB, each with appropriate array sizes for several different sizes of FEM models. This helps to reduce unnecessary memory consumption.

However, in order to avoid further useless waste of memory by a static storage of the equation system, the matrix elements are handled via a C-written interface, performing dynamic memory allocation.

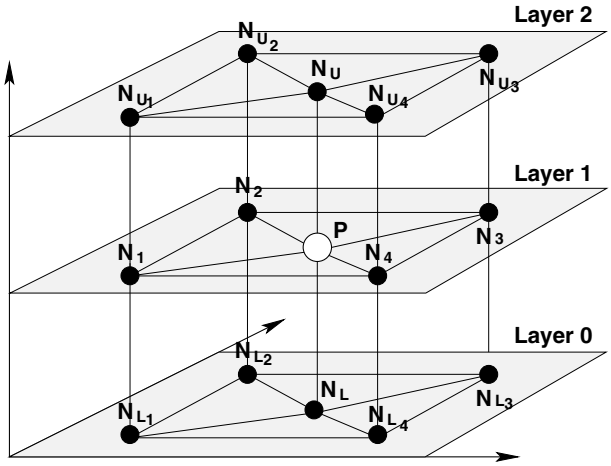

Figure 3. Vertex with its Fourteen Neighbours

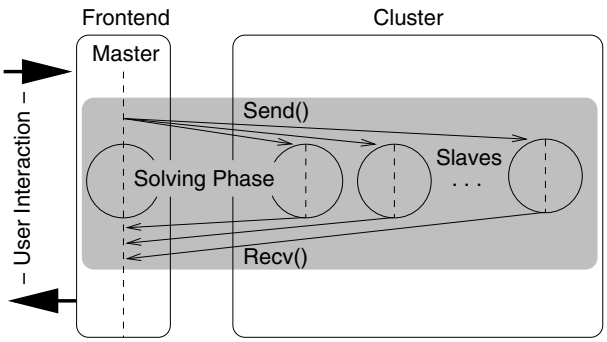

Figure 4. Master-Slave Concept

3. Parallelisation of TFB by Using MPI

As already stated in the introduction, numerical high-end simulation always demands parallel processing. Hence, it is not surprising that also WinCast users started to request a parallel version, at least for the time-consuming main processing part.

3.1. The Master-Slave Approach

One common approach to parallelise an existing code is the master-slave concept. Usually, the serial program is decomposed into two separate subprograms: namely the master and the slave program. With this paradigm [11], the master consists of the inherent serial parts of the original program, like initialisation and data preparation, while the slaves form a group of identical processes that process the actual computational work in parallel, driven by the master.

If dynamic process creation is available, the master can spawn slaves and gather their results later on. This procedure follows the Multiple-Program-Multiple-Data paradigm, which is the most common method especially for computation in a Parallel-Virtual-Machine (PVM) environment [10].

However, the MPI approach is commonly associated with the Single-Program-Multiple-Data paradigm, where the same single program runs on all participating nodes. Although there is only one single program, a master-slave organisation can also be achieved in an MPI environment by letting master and slave select different program parts to execute.

When parallelising the TFB module of WinCast, we have also chosen such a master-slave approach using MPI. The reason for this decision was the fact that the TFB module contains a lot of GUI-based user interaction and I/O operation parts besides the numerical solver algorithm. Thus, when choosing this approach, the master can handle all those non-parallelisable interactions, while the slaves get just active when entering the solving routine. The idea was to run the master process on a frontend node, e.g. the common desktop computer of the user, while the slaves are started for example on dedicated cluster nodes in a server room or on the remote nodes of an office grid, as denoted in Figure 4. All synchronisation and communication between the master and the slaves and among the slaves, respectively, is then done by message passing via a common TCP-enabled network like ethernet.

3.2. Implementation Details

When entering the solving phase, the master initially sets up the whole equation system. This is done independently for all of the slice layers of the FEM model. Hence, this process could be parallelised as well, but it is only marginally time-consuming to let the master do this on its own. A further reason why we have chosen this approach is that the slaves would require much more information and parametrised data, which would have to be sent to each of them after every time step.

The equation system, however, must be sent anyway at least partly to the respective slave processes. In our implementation, the domain decomposition, according to the data structures, is based on the layered style of the FEM model. Thus, each slave acquires a fixed number of slice layers for which it is responsible and solves the associated equations. If the number of layers is not divisible by the number of slaves without remainder, the remaining layers are assigned to the slaves symmetrically to the centre of the model. Since a layer is the finest granularity of decomposition, a small number of

layers per processor may be a source of load imbalance. As in common use cases the number of layers will be orders of magnitude bigger than the numbers of available processors, this represents a minor problem. See also Section 4 for measured effects on runtime.

In order to provide a global load balance in case of a heterogeneous computing environment, e.g. in an office grid, the parallel TFB provides optional dynamic layer assignment. After the completion of each time step, the slaves tell the master how long the last computation cycle took and how long they have been waiting in synchronisation barriers. Thus, a dynamically adapted reassignment for the next time step can be performed.

Before the parallel computation of the first time step, the master has to send all the needed static data and parameters to the slaves. Since this information does not change during iterations and further time steps, this big amount of data needs to be distributed only once per simulation run.

In order to solve the local part of the equation system, the slaves need to know about the current approximation of the values on the border layers adjoining to their own layers. Therefore, after each parallel Gauss-Seidel iteration step, independently performed by the processors on their local layers, this information must be exchanged via point-to-point messages.

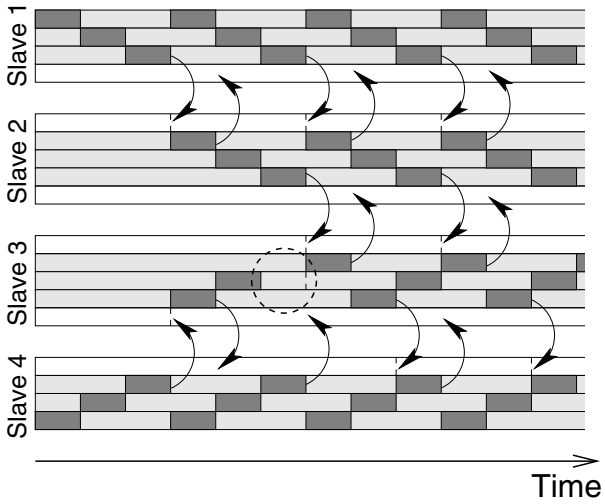

Figure 5. Pipelined Processing of Iterations

This information is needed whenever a process is working on the first or the last of its own layers. In order to obtain the same results as when performing serial processing, the processes must also emulate the alternating iteration order through the layers. Therefore, they work according to a *pipeline* scheme. During the first Gauss-Iteration step, only the slaves assigned to the top and to the bottom of all layers are working. When they have processed their last layer (while iterating in opposite directions), they send this information to their median neighbours, as shown in Figure 5. Now, the next inner slaves can receive these messages and start with the Gauss-Seidel iterations, too. This sequence is then continued to the centre of all slaves and layers. That way, after "*number of slaves DIV 2*" iterations all slaves are working. Whenever a slave has finished the calculation on a local border layer, the results are immediately forwarded to the respective neighbour via a non-blocking send-operation.

Assuming that the time for exchanging the data of one layer is much shorter than the computation time for all the local layers, communication and computation can be overlapped in a way that helps to keep the processes busy at all times. As you can see in Figure 5, "Slave 1" and "Slave 4" have already performed four complete iterations in that time while a serial process would have performed only one iteration step.

Of course, the shown Figure is idealised by assuming zero communication latency. And in fact, pipeline stalls appear if a slave must wait for a message to arrive, as denoted for "Slave 3". But assuming fully asynchronous communication operations, which can be overlaid with computation, those stalls can almost be hidden. Here it must be pointed out that an assumption of a completely overlayable communication may be flawed in real world. However, especially the Windows operating system offers such specific asynchronous and overlapped communication functions, as we state in Section 3.3.

In order to detect the termination of the Gauss-Seidel iterations of a time step, global error and maximal variation must be determined and distributed among the slaves. Therefore, the master and all the slaves perform *Allreduce*-like operations after each iteration step. Thus, each slave can check the abort criteria, and the master is always aware of the progression of the current relaxation and can display those parameters to the user.

3.3. The NT-MPICH Library

In this section, we want to introduce the *NT-MPICH* library, which is also developed at our institute. NT-MPICH, which is a subset of our *MP-MPICH* project [1], enables the user to perform MPI message passing on Windows platforms. Since WinCast is an application for Windows operating systems, the use of this library in order to perform message passing parallelisation seems obvious to us. In particular the deep knowledge about NT-MPICH implementation details enhances the overall performance of WinCast.

The communication device of NT-MPICH is mainly based on TCP communication via sockets. Additionally, on SMP[2] or Multicore systems an appropriate communication via shared memory can also be performed. In order to obtain maximal communication performance, NT-MPICH does *not* use the common Berkeley socket functions (aka BSD socket API), but utilises the native Winsock–2 API [6] of Windows. Although the Winsock API is based on BSD sockets and covers almost all their features, several extensions and enhancements have been integrated. One of the mentionable advanced features is the possibility to perform fully asynchronous data transfer via non-blocking send and receive functions. That means that computation and competing communication can effectively be overlapped. This is the reason why the parallel TFB with its pipelined iteration scheme scales very good, especially when running in an NT-MPICH environment.

Although NT-MPICH uses those native functionalities, the external TCP protocol is of course still interoperable with Berkeley sockets. Thus, it is possible to port the communication device to UNIX platforms as well by emulating the special overlapped communication functions. Actually, by using the *nt2unix* library [8], which has been also developed at our institute, a combined and cooperative operation of Windows and UNIX processes is possible in the same MPI-run.

4. Performance Evaluation

All performance measurements were performed on our 8 node Xeon cluster by testing an appropriately big problem size. The test environment was as follows:

- 8 Nodes, 16 CPUs
- Dual Intel Xeon 2.4 GHz 512 kByte Cache
- 8 GByte Total Memory
- 100 Mbit Fast- and 1000 Mbit Gigabit-Ethernet
- MPI Latency: $\approx 82\mu$s (FE) / $\approx 62\mu$s (GE)
- MPI Bandwidth can be seen in Figure 6

Problem characteristics:

- 52 Time Steps
- about 10137 Points and 30096 Lines per Layer
- 103 x 19960 Elements in 103 Layers
- $\approx$ 350 MB Memory Usage (in a serial run)

[2]Symmetric Multi Processors

In Figure 7 the measured speedup for the big problem is plotted over the number of slave processes, each running on a different cluster node. All nodes of this test run were using Gigabit-Ethernet as interconnect. As the figure shows, the parallelised problem scales quite well with the tested number of processes. Although the number of layers (here 103) is not divisible without remainder onto the eight slaves, the achieved efficiency is still proximate to 80% ($\approx$6,4/8).

A further graph in Figure 7 shows the total measured execution time for this problem when using different numbers of slaves. First of all, it can be seen that a serial run takes more than 36 hours, which is about one and a half day. Whereas, when using the parallel TFB on an 8 node cluster, the execution time can be squeezed down to about six hours, which is in turn smaller than a usual work day. Since shortening the simulation process speeds up the whole work flow, these results impressively legitimate the need for parallelised simulation tools.

In fact, certain users of WinCast have insistently requested a parallelised TFB module, in order to reduce the main processing time from about ten days down to one work day by using a 16 node compute cluster. Since the needed parallel efficiency is just 62,5%, we have forecasted its feasibility.

Unfortunately, we were not able to prove this on our own cluster. Although we possess a cluster with 16 CPUs, it is only an 8 node cluster, which means that the CPUs have to share the memory in pairs. Common problem sizes, even divided by the parallelisation, do not fit into the local cache of the CPUs. Due to this fact, the memory bus is a bottleneck anyway and when using two competing CPUs on an SMP machine, this effect gets doubled.

4.1. Impact of Load Imbalance

The measured load imbalance increases with the number of slaves used, because the layer ratio of disadvantaged slaves to the others rises accordingly. While the loss of speedup due to this imbalance is just 2% for 2 slave processes, it amounts up to 8% when using 8 slaves:

- 2 Slaves: 1x52 + 1x51 layers $\rightarrow$ ratio is 52/51=1,0196
- 8 Slaves: 1x13 + 7x12 layers $\rightarrow$ ratio is 13/12=1,0833

Of course, when using an *adapted* total number of layers, the speedup would scale even better, but we prefer to present achievable results for all common models. Due to this increasing loss of efficiency when scaling up the number of processes, the use of larger systems demands for bigger problems and vice versa.

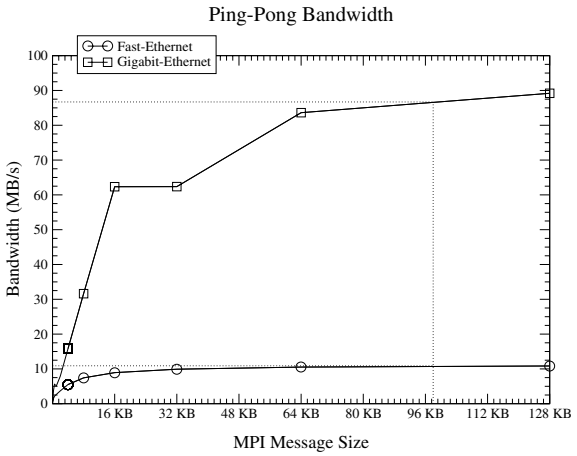

Figure 6. Ping-Pong Bandwidth

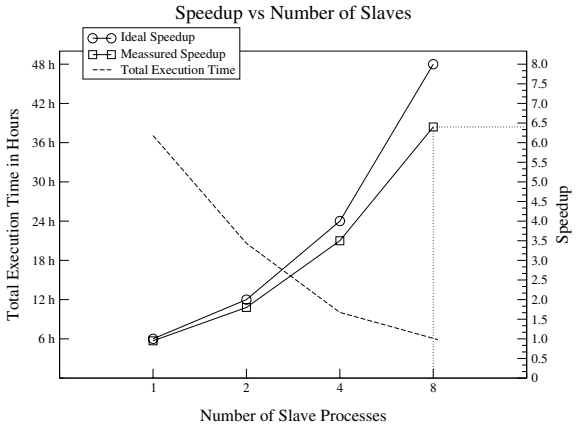

Figure 7. Speedup and Total Execution Time

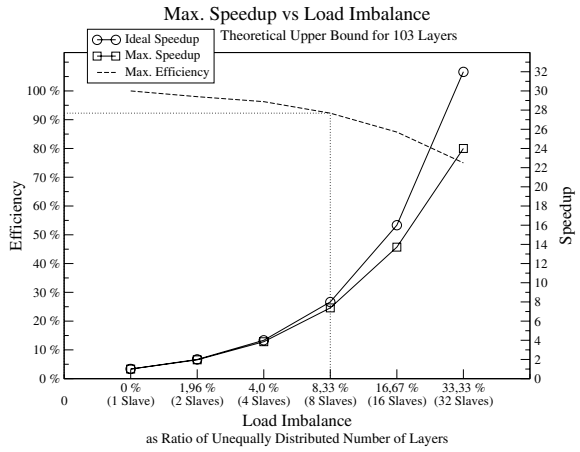

Figure 8. Impact of Load Imbalance

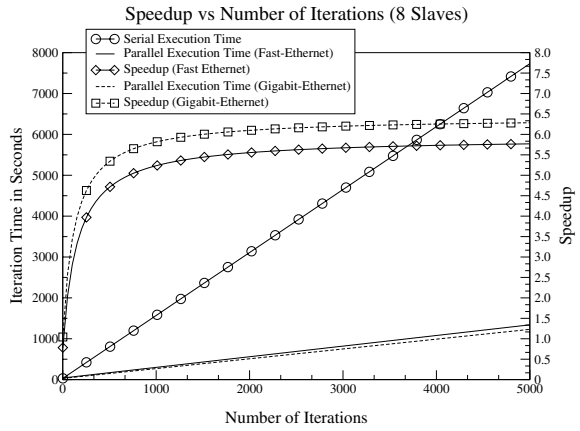

Figure 9. Speedup Depending on Iterations

Figure 8 shows how the presented problem with its 103 layers may scale for even more processes. In this figure, a theoretical upper bound for speedup, just reduced by the effects of load imbalance, is plotted over the number of slave processes and the according ratio of the unequally distributed number of layers.

It can be seen that the theoretically achievable efficiency for 8 slave processes is about 92%, whereas the measured efficiency is just about 80%. This gap is just due to the neglected communication overhead in the theoretical approach of Figure 8.

4.2. Impact of Communication

Hence, a further interesting aspect is the impact of the required communication on the speedup. As already stated in Section 3.2, there are two types of communication within a time step.

Initially, the master distributes the equation system to the slaves and gathers the results later on when the time step is finished. This part of the communication consumes roughly the same time for each time step independent from the number of iterations performed. Its impact on speedup can be recognised in Figure 9. This static portion of communication causes the speedup to increase with number of iterations. If the number of iterations is small, this portion dominates the execution time and holds the speedup down. In turn, for a larger number of iterations the speedup rises and converges to a maximum.

The second communication part is the accumulated data exchange among the slaves during the iterations. In case of the presented problem, an amount of 98 KB of data needs to be exchanged in each iteration per neighbouring slaves. The impact of this communication can be measured by using different interconnects. As can be seen in Figure 9, the maximal speedup decreases when utilising the slower *Fast-Ethernet* instead of *Gigabit-Ethernet*. This reduction of speedup shows

that the assumption of fully overlapped communication and computation is limited in reality. Nevertheless, using the slower network only causes a speedup loss of about 10%, while the bandwidth ratio is about 8:1 for the exchanged message size (see Figure 6). That means that the influence of the slave-to-slave communication on the execution time is still pretty moderate.

5. Related Work

The parallelisation of iterative solvers is a well understood domain of computational engineering and a lot of research and development has already been conducted in this area. There exists a huge amount of literature dealing with those subjects. Thus, citing all this related work would go far beyond the scope of this paper.

Nevertheless, we want to mention two competitive products to WinCast in order to present actual related work in the domain of parallelised casting simulation. The first product is *ProCAST* by the ESI-Group and the second product is the casting simulation suite from *MAGMASOFT*. Both suites possess parallelised solving cores for distributed memory systems and both tools are runnable on Windows platforms as well as on UNIX-like operating systems.

6. Conclusions and Outlook

Since the computer aided simulation of casting and solidification processes has become a necessity in founding industry, also the demand for parallelisation of the respective simulation tools increased in the last years. In this paper, we have presented our work accomplished to parallelise the core module TFB of the WinCast simulation suite. The measured performance results show that the chosen master-slave approach scales quite well on compute clusters. This approach also offers the facility to use the software tool in a familiar manner, where the GUI-based master is running on the user's desktop computer, while the slave processes are working transparently on a cluster or an office grid. Particularly the use of the NT-MPICH library for message passing, which provides support for the enhanced native Windows socket communication functions, offers an efficiently overlapped computation and communication scheme for the slave processes.

Although Windows is not a common cluster operating system, the parallel TFB allows convenient and advanced main processing at the same time. Since WinCast is ported from the UNIX-related SIMTEC suite, even a cooperation of a Windows-based master with UNIX-based slaves is thinkable. Especially the NT-MPICH library is already ported to UNIX and enabled to perform interoperable message passing via TCP sockets.

In future, parallel architectures and parallelised code will become even more important, also in the area of mid-performance and desktop computing. Besides symmetric multi processor systems, *Dual-* and even *Multi-Core* processors will increasingly enter this area of computing. In order to exploit such platforms efficiently, thread-level parallelism should be preferred. Hence, an additional *OpenMP*-based [3] parallelisation of the casting simulation code would be very desirable in future. But since the memory bus is still a bottleneck also in those systems, this approach will demand deeper changes in the solving code in order to obtain a cache adapted parallelisation.

References

[1] Chair for Operating Systems, RWTH-Aachen, University. *MP-MPICH – User Documentation & Technical Notes*.

[2] M. T. Chapman. The Benefits of Dual-Core Processors in High-Performance Computing. White Paper, June 2005.

[3] L. Dagum and R. Menon. OpenMP: An industry-standard API for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1):46–55, 1998.

[4] M. Gremaud and M. Gäumann. Introducing Casting Simulation in Industry: The Steps Towards Success. In *Report of the 133rd Annual Meeting & Exhibition of The Minerals, Metals & Materials Society (TMS)*, 2004.

[5] W. Kapturkiewicz, E. Fras, and A. Burbelko. Why is the computer modelling needed in casting? *Przeglad Odlewnictwa (Foundry Journal)*, 55(1):15–23, 2005. AGH University of Science and Technology, Krakow, Poland.

[6] Microsoft. *Windows Sockets 2 Application Programming Interface, An Interface for Transparent Network Programming Under Microsoft Windows*. Manual.

[7] MPI Forum. MPI: A message-passing interface standard. *International Journal of Supercomputing Applications*, 1994.

[8] S. M. Paas, T. Bemmerl, and K. Scholtyssik. Win32 API Emulation on UNIX for Software DSM. In *Proceedings of the 2nd USENIX Windows NT Symposium*, pages 39–46, Seattle, Washington, USA, August 1998.

[9] R. D. Pehlke. Computer Simulation of Solidification Processes - The Evolution of Technology. *Metall. Mater. Trans. A*, 33A:2251–2275, 2002.

[10] V. S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency, Practice and Experience*, 2(4):315–340, 1990.

[11] B. Wilkinson and M. Allen. *Parallel Programming*. Prentice Hall, 2nd edition, 2005.