II International Conference on Particle-based Methods - Fundamentals and Applications
PARTICLES 2011
E. Oñate and D.R.J. Owen (Eds)

ON THE GPU COMPUTING OF MASSIVE FORMING PROCESS SIMULATIONS

G.-P. Ostermeyer, K. Fischer

Institute of Dynamics and Vibrations (IDS)
Technische Universität Braunschweig
Schleinitzstr. 20, 38106 Braunschweig, Germany
e-mail: gp.ostermeyer@tu-braunschweig.de, ka.fischer@tu-braunschweig.de, www.ids.tu-bs.de

Key words: Particle Systems, Forming Processes, GPU Computing, CUDA

Abstract. This contribution presents a modelling tool for massive forming processes that is based on a particle method. The introduced model is able to represent the realistic behaviour of different types of forming processes. As these systems usually require large amounts of particles, the potential of GPU Computing with CUDA as a possibility for performance enhancement of particle simulations is analyzed as well.

1 INTRODUCTION

The Finite Element Method (FEM) is the common tool to model massive forming processes. This method is able to deliver all the necessary information (cf. figure 1) in order to develop and optimize massive forming processes. Among these are the information e.g. on workpiece's structural defects, on the mould filling behaviour (whether or not a mould could be completely filled under the given circumstances), on the microstructural transformation that is directly correlated to the natural stain, on the material flow and the flow lines or on the stress and temperature distribution throughout the workpieces and tools.

Although delivering quite realistic simulation results in many cases, FEM is not able to reproduce all the upcoming phenomena like mould filling errors or workpiece's structural defects. The reason for this lies in the fact, that a correct macroscopic description of friction and wear during the forming process has not been derived yet [1]. Therefore the idea of modelling massive forming processes with particle methods is motivated, because particle methods do not depend on those macroscopic laws [2].

Despite the problem of mould filling errors, large deformations, flashing or the generation of wear require frequent remeshing during a FEM simulation. This results in a considerable increase of computational effort and therewith worsens the overall performance. The computational effort of particle simulations instead is in principal independent of the type and degree of deformation like flashing or generation of wear.

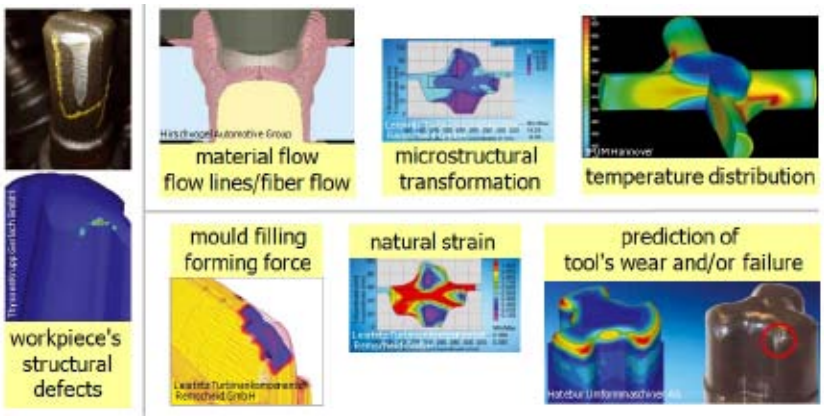

Figure 1: Massive forming process simulation requirements (sources of single pictures are referred to in each picture)

2 MESOSCOPIC PARTICLE METHOD

The particle method used in order to model massive forming processes is the mesoscopic particle method introduced in [3, 4] where the particles have additional hidden degrees of freedom. In general, for the given application, the system is discretized by rigid particles. The system's particles interact with each other via local and predefined interaction laws. These laws have to be chosen in such a way, that a realistic macroscopic material behaviour is represented. There are different approaches to derive them. One possible way is the explicit implementation of a macroscopic stress strain relationship. Another approach considers interaction potentials similar to those used in molecular dynamics like the Lennard-Jones potential. In either case, the discretization results in a usually large amount of additional degrees of freedom, that are of course not visible from the macroscopic point of view. Thus, they are also called submechanical or hidden degrees of freedom.

In order to generate the macroscopic system performance, the information have to be transfered form the chosen discretization scale with its hidden degrees of freedom to the macroscale. Considering forming processes, especially the information on the macroscopic temperature distribution -as a result of heat generation during the forming process- is of utmost importance.

On the atomic scale, molecular dynamics use statistics in order to model the macroscopic heat generation and temperature distribution correctly. But in contrast to the atomic scale, macroscopic properties like temperature can directly be assigned to the particles. Therefore, in order to take into account the mechanical as well as the thermodynamical degrees of freedom, viscous dampers are used to separate these different kinds of energy on the mesoscopic time scale. The dispersion relation separates the high and low frequencies of motion. As a consequence, the dissipated energy of the high-frequency motions is interpreted as heat energy and attached to the corresponding pair of particles. To

store this information, the particles have an inner variable, describing the corresponding macroscopically observable property temperature T. The low-frequency motions instead stay unchanged for long time intervals, compared to the mesoscopic time constant. Depending on the application, additional internal variables like stress or chemical changes can be reasonable or even required.

The standard Lennard Lennard Jones potential is not sufficient to model massive forming processes. These types of systems require the reproduction of additional effects like thermal expansion or local smelting. Therefore Ostermeyer [4] modified the standard Lennard-Jones potential into

$$V_{LJW} = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{2n} - \sqrt{1 - \frac{W^2}{\epsilon^2}} \left(\frac{\sigma}{r}\right)^{n} \right] \tag{1}$$

which explicitly depends on the stored heat energy W. This is quite reasonable since the attractive part of the potential vanishes, when the phase transition temperature is reached. r marks the distance between two neighbouring and interacting particles. ϵ, σ and n are the potential's parameters and have to be chosen according to the material properties like Young's modulus. The inner variable temperature T is connected to the stored heat energy W via

$$W = mc_V T \tag{2}$$

This equation defines heat energy as the product of mass m, specific heat capacity c_V and temperature T.

In order to achieve the equations of motion of such a particle system, the Lagrangian function $L = E_{kin} - E_{pot}$ is considered with E_{kin} and E_{pot} as the kinetic and potential energies of the given system. L depends on the generalized coordinates u and the generalized velocities $\dot{u}$. The dissipation function D takes into the energy dissipation and only depends on the generalized velocities.

$$\frac{d}{dt}\left(\frac{\partial}{\partial \dot{u}} L\right) - \frac{\partial}{\partial u} L = -\frac{\partial}{\partial \dot{u}} D \tag{3}$$

The total energy inside the system including all the heat energy W_T has to be conserved.

$$E_{kin} + E_{pot} + W_T = const \Longrightarrow \frac{d}{dt}\left(E_{kin} + E_{pot} + W_T\right) = 0 \tag{4}$$

Rearranging this equation yields to the following expression, describing the local heat generation within a system considering heat dependent interaction potentials.

$$\dot{W} = \dot{u} \frac{\partial}{\partial \dot{u}} D \cdot \frac{1}{1 - \frac{\partial L}{\partial W}} \tag{5}$$

To describe the overall local heat evalution during the forming process, the model does not only have to consider local heat generation but also has to account for global heat conduction (Fourier) within the workpiece and between workpiece and forging die and in addition heat convection and thermal radiation along the surface during the cooling process outside the die. Equation (5) has to be expanded accordingly.

3 MODELLING MASSIVE FORMING PROCESSES WITH PARTICLE METHODS

The modeling of forming processes is very important, since the plastic deformation of a solid body has a significant impact on the final body's characteristics like strength, strain or vibration resistance. These properties depend e.g. on the evolution of material flow respectively the flow lines during the forming process. The workpiece's characterics are much better along the flow lines than in a lateral direction.

Before applying a particle method to simulate complex forming processes in order to determine realistic process' information, the particle model has to be validated. In order to show its reliability, the simulation results of a standard forming process are generated and compared to experimental results. The process, that is going to be analyzed in the following, is the compression of a cylindrical workpiece. The realistic reproduction of this compression is in fact very important, because it oftentimes marks the first forming step within a complex and graduated forming process.

The aspects used to validate the results achieved by the particle method are the flow lines, the temperature distribution and the natural strain that is directly coupled to the microstructural transformation.

The flow line distribution is a result of the grains' orientation inside the workpiece. Recrystallization leads to a structural change, which is mechanically rearranged during the plastic deformation. The finial orientation depends on the initial grain orientation and on the direction of the forming process. FE simulations generate the flowlines by tracking certain grid nodes (the initial coordinates and their displacement evolution) during the forming process. Problems arise, when a frequent remeshing is required, because a new mesh lacks the information on the predestined material flow. The introduced particle method prevents such a loss of information. Each initial flow line is represented by a chain of assigned particles. This initial assignment stays unchanged during the total forming process. The flow line evolution is therefore observed by tracking the assigned particles as part of the corresponding flow line.

The typical vertical and horizontal flow line distribution of such a compression test (experiment cf. [5]) compared to the simulation results is shown in figure 2 and figure 3. The simulation results show the same typical flow line distributions in vertical and horizontal direction as the experiments. The flow lines are parallel to the lateral edges of the final workpiece's shape. Figures 2 and 3 also show the main advantage of massive formed parts: the typical uninterrupted flow line distribution that allows an excellent behavior considering static and dynamic loading.

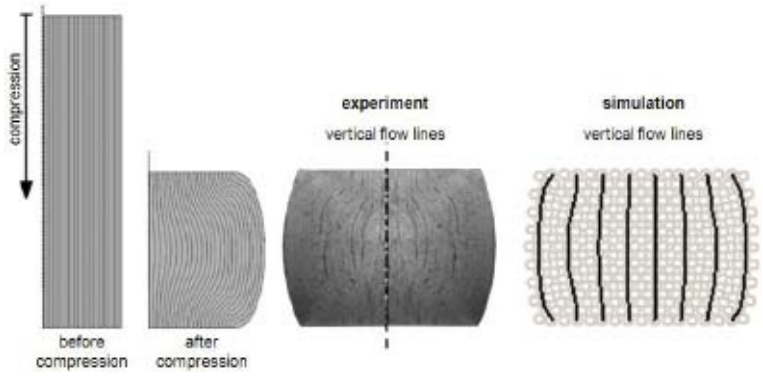

Figure 2: Vertical flow lines: experiment (cf. [5]) versus particle simulation result.

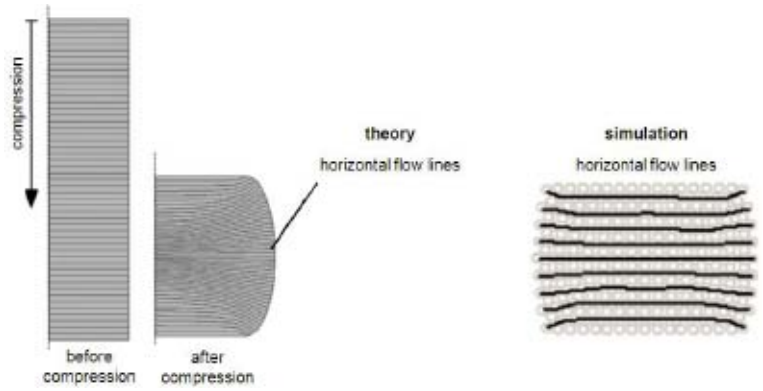

Figure 3: Horizontal flow lines: experiment (cf. [5]) versus particle simulation result.

The forming process considered is an inhomogeneous one. This means that the deformation is not equal for each point of the workpiece. The result is a convexly compressed workpiece (cf. figures 2 and 3). The reason for this is the amount of friction, that is present between workpiece and upper and lower die respectively plate. This friction prevents an even lateral expension at the workpiece's end planes. During the forming process, the major fraction of the generated deformation energy is turned into heat energy. As a result of the inhomogenous forming distribution, the temperature distribution as well as the strain distribution are not homogenous either.

Besides the flow lines, the structure (grain size and distribution) has a vital influence on the final workpieces mechanical properties like ductility or further shape cutting. The final grain size of a compressed workpiece depends on the initial grain size, the type of recrystallization (works against the hardening as a result of plastic deformation), the temperature and the natural strain [5]. A larger strain e.g. leads to smaller grains during the dynamic recrystallization. So, for a given initial grain size, the particle simulation has to generate a realistic temperature and strain distribution to deliver the correct final

grain size distribution. That is the reason, why the validation of temperature distribution and local natural or effective strain is essential.

The natural or effective strain distribution describes the amount of local deformation considering the local neighbourhood. The strain distribution after the compression can be experimentally determined by hardness testing. Figure 4 shows the principal strain distribution as a result of experimental Brinell hardness testing [6]. Three different strain areas can be detected: I with large strain, II with medium strain and III with low strain. The area with the highest amount of strain ranges in diagonal form from the center of the workpiece to its edges. The simulation results are in accordance with the experimetal observations. The three areas could be clearly identified within the particle representation (cf. figure 4).

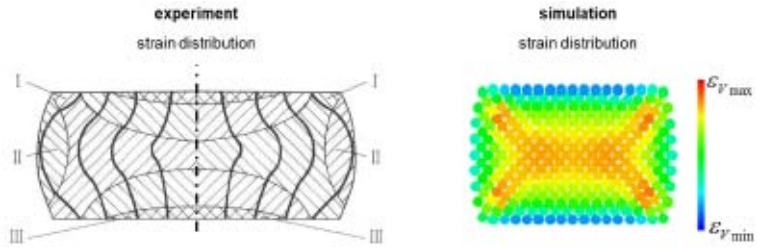

Figure 4: Strain distribution: experimental result (cf. [6]) versus particle simulation result

The temperature measurement inside the workpiece during and directly after a forming process is not quite possible. So, a comparison between experimental data and simulation results has not been carried out. Instead, figure 5 shows the results of the particle simulation compared to the established ones of an FEM simulation (cf. [6]). These FEM-results show an increse in temperature inside the workpiece that is directly correlated to the strain distribution. Those parts inside the workpiece, that experienced the largest degrees of rearrangement (cf. figure 4) are those with the highest final temperatures after the forming process. As a consequence, the temperature distribution shows a formation of three areas comparable to those of the strain distribution. This correlation is also correctly reflected by the results of the particle simulation.

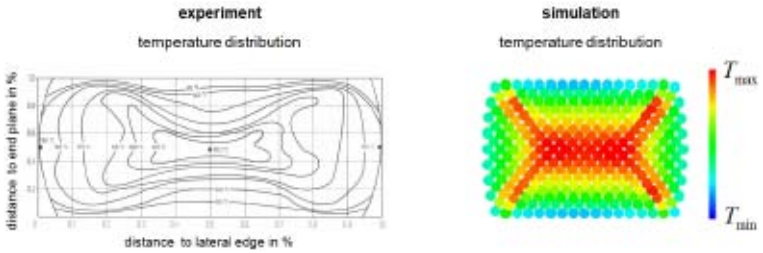

Figure 5: Temperature distribution: FEM result (cf. [6]) versus particle simulation result.

4 GPU COMPUTING AND PARTICLE METHODS

The number of particles used for the modelling depends on the system's size and the required accuracy. The higher the demands on the simulation's quality respectively the chosen size of potential forming errors are, the smaller will the size of a single particle be, which will result in an increase in the total amount of particles. As the analyzed systems usually have very large numbers of particles, algorithm's design and the corresponding performance enhancement has become a major task in algorithm development. Parallelization is a widely spread concept for this purpose. The source code parallelization e.g. on multi-core CPUs or CPU clusters is already very popular among scientists and engineers.

During the past few years, another approach allowing massive parallelization on special graphics processing units (GPUs) has been developed by manufacturers like NVIDIA [7]. As a result of the fast growing video game industry, GPUs have been optimized to perform large numbers of floating-point operations for each video frame. Therefore, GPUs allow for a much higher degree of parallelization than CPUs can do. This development is also shown by the fact, that among the top four supercomputers of the world three are already using NVIDIA GPUs [8].

In 2007, only short time after the launch of its first general purpose GPU, NVIDIA introduced a user-friendly computing architecture called CUDA (Compute Unified Device Architecture). This architecture allows standard C programming with additional runtime functions but without dependence on standard graphics user-interfaces like OpenGL anymore.

Besides the performance, GPU Computing has got some other quite important advantages compared to CPU Computing. The acquisition costs per GFLOP as well as the power consumption per GFLOP are much lower for GPUs than for CPUs. The main challenge of GPU Computing is the fact, that CPUs and GPUs require a completely different algorithm implementation. So, a GPU implementation of a given CPU implementation oftentimes results in an algorithm's redesign. Furthermore, it is the user's task to generate a workload distribution that leads to an optimal usage of GPU's resources.

As a matter of principle, GPU computing makes only sense for those systems that provide a large parallel portion in the corresponding algorithm. Systems with no or only minor parallel parts will not benefit from a GPU implementation, quite the opposite. In these cases, a GPU implementation can even worsen the performance. Particle systems e.g. are ideally suited for parallelization, because the same set of instructions has to be executed for each particle. But since the algorithm's redesign is costly, it has to be determined, if the performance benefit justifys the time and effort.

Therefore, a single CPU (program only executed on one core) as well as a single GPU implementation of the particle system's update algorithm have been realised for three different types of particle systems as well as an evaluation of the performance [9]. The update procedure is identical for each type of particle system and follows a tree step

update-scheme. The first step includes the determination of each particle's neighbourhood and is the most time-consuming part of the total algorithm. Therefore the type of interaction detection algorithm has to be well-chosen. Afterwards, the correponding interaction forces are computed and finally the current set of system's differential equations is generated and solved via time integration.

The analyzed particle systems vary in system's size (number of particles) and range of interaction forces. They therefore have quite significant differences in their computational intensities. The variation of the interaction range leads to three different types of particle systems:

- non-interaction particle system: no interaction among the particles; the system's state update is reduced to the pure time integration
- short-range interaction particle system: particle interaction only within a small neighbourhood; neighbourhood detection, calculation of corrsponding interaction forces and time integration required
- all-pair interaction particle system: each particle interacts with all the other particles; no neighbourhood detection required, only brute-force calculation of interaction forces and time integration

As a consequence, the non-interaction particle system has the lowest computational intensity. The all-pair interaction particle system has the highest computational intensity with a computation of interaction forces of order $O(n^2)$. The reduction of interaction range to small neighbourhoods (short-range interaction system) reduces the effort of force calculation to $O(n)$ but requires an additional neighbourhood detection. A variety of neighbourhood detection algorithms (comparable to the problem of collision detection) have been developed [10]. Their efforts vary from $O(n^2)$ to $O(n \log n)$. In contrast, effort's reduction leads to a rapid increase in memory requirements. Therefore, the neighbourhood detection algorithm, chosen for the following investigations, is the brute-force neighbourhood detection of order $O(n^2)$. It was chosen because it represents a lower limit and so the minimal benefit of performance enhancement of a GPU implementation compared to a CPU implementation for short-range interaction particle systems.

The integration method used is the standard Euler forward integration as it is the basic method. A more time-consuming integration method will lead to similar results since it will affect the implementations on the CPU and on the GPU in a similar way. So, the difference will only be a constant factor. In case of interaction, the particles interact via Lennard-Jones potentials.

The two hardware platforms used for performance evaluation are

- CPU: Intel Core i7-920, 2.70 GHz
- GPU: NVIDIA Tesla C1060 (240 streaming processors)

In order to evaluate the performance of an algorithm implemented and executed on a single CPU core (completely sequential implementation) in contrast to a single GPU implementation (parallelized execution), the computing time required for 100 update steps is measured for both platforms. This time does not consider the time for visualization of the simulation results. If not otherwise indicated, the time required for the GPU implementation does not take into account the time used to copy the data from the CPU to the GPU and finally back to the GPU. The performance is determined by the amount of time, required to complete the update. The more time is spent on the update steps, the slower and therefore worse is the performance of the analyzed implementation.

Figures 6 and 7 show the time in ms required for the GPU and the CPU implementation for the lower and upper limits of computational intensity: the non-interaction particle system and the all-pair interaction system. The blue curve marks the computing time of the CPU implementation and the red curve the one of the GPU implementation.

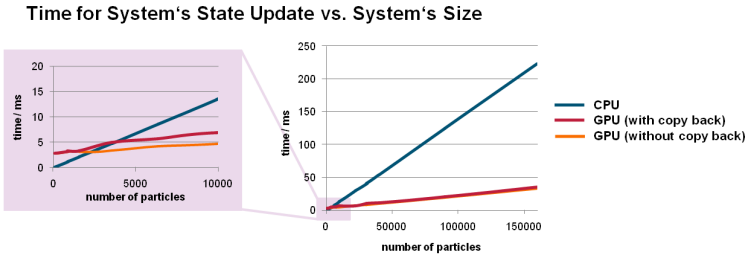

Figure 6: Computing time vs. system's size for the non-interaction particle system.

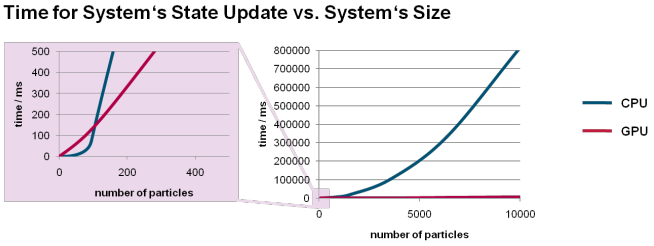

Figure 7: Computing time vs. system's size for the all-pair interaction particle system.

In figure 6, the execution time increases at a constant rate with increasing number of particles for both implementations. This trend was expected, because the non-interaction particle system has a computational complexity of order $O(n)$. In contrast, the all-pair interaction particle system's complexity of order $O(n^2)$ is directly represented by the CPU implementation (cf. blue curve in figure 7). Figure 7 also shows an important aspect of GPU's performance enhancement. The GPU implementation reduces the squarish dependency between computing time and system's size of the sequential implementation

to a linear one. Although the total time required for the all-pair interaction approach is much higher than the one for the non-interaction system, both systems indicate the same behaviour: the GPU implementation outperforms the CPU implementation for systems larger than a critical system's size. The critical system's size is much lower for the computational intensive particle system than for the non-interaction system.

To evaluate the benefit of a GPU implementation in contrast to the CPU implementation, the speed-up is a frequently used quantification value. The speed-up is a factor, that states, how many times faster a GPU implementation is compared to a CPU implementation. It is therefore defined as:

$$\text{speed-up} = \frac{\text{performance(GPU)}}{\text{performance(CPU)}} = \frac{\text{computing time(CPU)}}{\text{computing time(GPU)}} \tag{6}$$

The speed-up depends in general on the problem's computational intensity, the degree of parallelizability, the problem's size and the implementation which requires the optimal usage of GPU's resources. The amount of information that has to be stored on the GPU for the required computations strongly limits the performance. If large particle systems exceed the GPU's memory capacities, the particle system has to be manually partitioned by the user and loaded to the GPU and processed sequentially portion by portion. This procedure is necessary since GPUs do not support virtual memory as CPUs do.

Figure 8 shows the resulting speed-ups for the three analyzed types of particle systems. These speed-up results do not represent the highest possible speed-ups comparing single CPU and single GPU implementations, since the implementations have not been optimized to the dead end yet. Instead, the figure should indicate the potential of GPU computing compared to CPU computing. Figure 8 makes clear, that an enormous performance enhancement can be achieved for all types of particle systems, when implementing the algorithm on a single GPU instead of a single CPU.

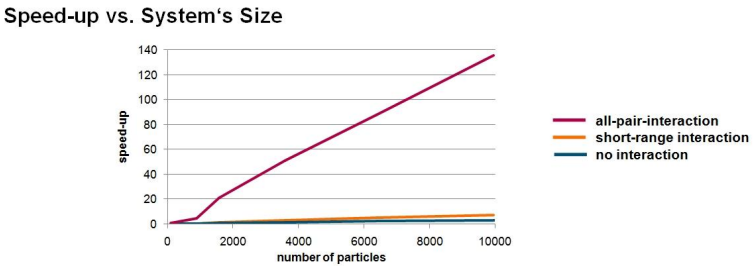

Figure 8: Speed-up

It should be noted, that the speed-up also depends on the chosen accuracy. While double and single precision performances do not lead to considerable differences in CPU's performance, there are large differences in GPU's performance. The reason for this is the fact, that the older CUDA capable GPUs have only been optimized for single precision.

So the support for double precision operations and performance was neglected, which led to a large performance loss for double precision applications (up to 80 % [11]). The new Fermi GPU generation takes care of these insufficiencies and increased not only single but also double precision performance. A double precision GPU implementation is still about 50% slower than a single precision one, but on Fermi GPUs it is able to outperform double precision CPU implementations [12], which was not possible with older GPUs.

5 CONCLUSIONS

- It has been shown, that the modelling of massive forming processes with particle systems delivers a realistic material behaviour considering the evaluated criteria of flow line distribution, strain and temperature distribution. The particle simulation's results turned out to be in good agreement to the experimental ones and to those of standardized FEM simulations. So, the introduced particle model seems to be qualified to be used for further analysis of more complex forming processes.

- GPU Computing was introduced as a tool for performance enhancement of systems with a large parallel portion. Particle systems meet this demand and are therefore predestined for parallel computing. The analysis of three different types of particle systems showed, that the performance of particle systems' simulations implemented on a CPU can be improved by parallelizing the neighbourhood detection as well as the state update on a GPU.

REFERENCES

[1] Herbertz, R. *Kritische Bewertung der Stoffffluss-Simulation, der Verschleißvorhersage und der Werkzeugbelastung in der Massivumformung aus Sicht der Reibung in der Grenzschicht Werkstück/Werkzeug.* Studie im Auftrag des Industrieverbands Massivumformung e.V., (2009).

[2] Ostermeyer, G.-P. *Entwicklungsansätze und -potentiale für neue Reibgesetze in der (Warm-) Massivumformung".* Tagung "Reibung in der Massivumformung"., Hagen, (02.12.2009).

[3] Ostermeyer, G.-P. *Many particle systems.* German Polish Workshop, IPPT PAN, Warszawa, (1995).

[4] Ostermeyer, G.-P. *The mesoscopic particle approach.* Tribology International (2007) **40**:953–959.

[5] Doege, E. and Dittmann, J. *Vorhersage der Mikrostruktur und mechanischen Eigenschaften geschmiedeter Bauteile durch FEM-Simulation.* Mat.-wiss. u. Werkstofftech. (2002) **33**:683–690.

[6] Lange, K. et al. *Umformtechnik Band 2: Massivumformung*. Springer-Verlag, second edition,(1988).

[7] Kirk, D. B. and Hwu, W. W. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers, Vol. I., (2010).

[8] TOP500 List - November 2010 (1-100), http://www.top500.org/list/2010/11/100, (November 2010).

[9] Fischer, K. and Ostermeyer, G.-P. *Massively Parallel Particle Simulation on Graphics Processing Units with CUDA*. GAMM 2011, Graz, (2011).

[10] Ericson, C. *Real-time collision detection*. Morgan Kaufmann series in interactive 3D technology. Elsevier Morgan Kaufmann, Amsterdam, reprint, (2008).

[11] NVIDIA. *NVIDIAs Next Generation CUDA Compute Architecture: Fermi: NVIDIA Whitepaper*. (2009).

[12] NVIDIA. *NVIDIA CUDA C Programming Guide*, (July 2010).