Hardware-Efficient Parallelized Optimization with COMSOL Multiphysics® and MATLAB®

Frommelt Thomas* and Gutser Raphael
SGL Carbon GmbH
*Corresponding author: Werner-von-Siemens Straße 18, 86405 Meitingen, Thomas.frommelt@sglgroup.com

Abstract: Today, new processors provide increasing number of cores at rather constant clocking frequency. In sequential optimization algorithms, the forward model simulation is typically accelerated by multiple cores (shared-memory parallelization, SMP), which provides only limited speed-up and hardware efficiency. However, the Comsol Multiphysics® license includes parametric design capability allowing the application of population-based approaches with simultaneous simulation of multiple models with single cores. In this work, a simple optimizer based on latin hypercube sampling is presented to improve positioning of parts in an industrial-scale graphitization furnace. In a comparative study to sequential Nelder Mead Simplex (fminsearch), considerable performance advantages are demonstrated.

Keywords: Optimization, Matlab®, Parallelization, Latin Hypercube Sampling

1. Introduction

The power dissipation of processors rises exponentially with clocking frequency. Around 2004, the power limit was reached and lead to a paradigm change in processor development [1]. Instead of higher speed by higher clocking frequency, chip manufacturers continuously increase the number of cores per processor at rather constant clocking frequency. In order to address the capability of such hardware, simulation software like Comsol Multiphysics® includes parallelized solvers like MUMPS that allow multi-core computation of a single model by shared-memory parallelization. Yet, splitting up a simulation onto several cores requires more memory and creates organizational overhead. Consequently, shared-memory parallelization does not decrease the duration of a simulation ideally by far. For single simulations and large models, this technique is without alternative. For optimization of smaller models, a more suitable approach for today's hardware architecture is presented for an industrial-scale test model.

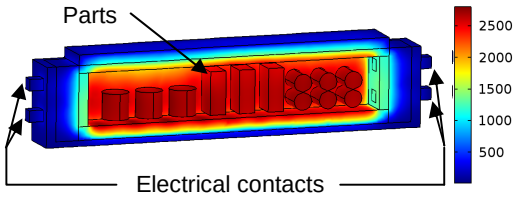

Figure 1. Temperature distribution in an industrial Acheson graphitization furnace. The conductive packing media is hidden to show the parts.

2. Test Model System

In our test model, the transient heating process of carbon parts in an industrial scale graphitization furnace (Fig. 1) is simulated. By applying an electric voltage to the external contacts, current flow is generated through the electrically conductive packing media (hidden in Fig. 1) and the parts. The resistive power dissipation heats the amorphous carbon parts up to graphitization temperature above 2500°C. The model provides a transient simulation of the electro-thermal process including a controller to follow a predefined power curve.

The model is meshed with tetrahedral elements and calculated with 2nd order Lagrange shape functions. Various settings were adjusted to minimize the runtime of a forward simulation by more than 30% and memory consumption by more than 60% with respect to default settings. These figures illustrate the vast potential for performance optimization through direct access to all model properties in Comsol Multiphysics® which is not the focus of this publication.

The tests were performed on a two-socket workstation, with 6 cores per processor, operated with Windows® 7 and Comsol Multiphysics® 4.3b. The 64 GB RAM fulfilled the requirement for simultaneous simulation of up to 12 models without running into memory capacity bottlenecks.

During the analysis of different parallelization techniques, the model parameters and settings are not modified to ensure equal conditions. As a performance indicator, the number of possible model runs MRUN per time period is cal-

culated from benchmark trials. The reference case MRUN=1 corresponds to a single model computed with single core. By repeating the analysis for different configurations, average and range of MRUN were determined for all parallelization techniques.

3. Shared-Memory Parallelization – Single Model, Multiple Cores

In shared-memory parallelization (SMP), multiple cores work on a single model. The speed-up potential was analyzed considering different numbers and allocation strategies of cores. At the example of five allocated cores, the different strategies are visualized in Tab. 1:

a) Occupy one socket first, then the second
b) Occupy both sockets equally
c) Automatic allocation by Comsol Multiphysics® NP command line option

In strategies (a) and (b), a fixed allocation of the cores to the batch simulation process was realized by affinity masks. In strategy (c), the allocation was automatically handled by the operating system.

If speed-up is ideal, MRUN is equal to the number of utilized cores, as e.g. two cores would reduce model runtime by 50% and allow 2 model runs per time period. This ideal limit is shown as a dashed line in Fig. 2 and Fig. 3.

The graphs in Fig. 2 point out that all allocation strategies are far from the ideal limit. For strategy (a), there is a steady increase in MRUN until core 7 – the first core on the second socket – is used. The observed bifurcation of runtimes can be explained by the allocation of RAM for the simulation process on the memory bank of one socket. The remote access of a core from the other socket onto that memory bank via the mainboard's Intel Quickpath Interconnect is slower by up to a factor 2, as compared to the access by local socket cores [2]. An allocation of memory to a socket with few utilized cores deteriorates MRUN significantly. For 7 cores in strategy (a), 6 cores can be on the slower socket which maximizes the bifurcation.

The same effect happens for configurations with uneven number of cores (3, 5, …) in strategy (b): If the additional core is on the wrong socket, MRUN is not increased. Thus, a bifurcation of runtimes is detected for uneven number of cores in strategy (b):

- Additional core on the right socket ➔ lower runtime ➔ higher MRUN
- Additional core on the wrong socket ➔ similar runtime ➔ similar MRUN

The bifurcation is weaker than for strategy (a), as the cores are more evenly distributed onto the slow and fast sockets.

In general, the automatic strategy (c) seems to perform well, except for a strong bifurcation at 5 cores and insignificant increase of MRUN beyond 6 cores. In the Windows® task manager, the NP command homogeneously covers all available cores. For 12 cores, SMP strategies provide the same performance in terms of MRUN.

These results support the conclusion that hardware architecture and parallelization overhead limit the SMP benefits, particularly beyond one socket. Thus, an efficient and reliable (without bifurcations) utilization of state-of-the-art multiprocessor systems cannot be guaranteed.

		Core allocation										
	Socket Number:	1						2				
	Core Number:	1	2	3	4	5	6	1	2	3	4	5	6
Strategy	a) One socket first	X	X	X	X	X							
	b) Two sockets equal	X	X	X				X	X				
	c) Comsol NP command line option	Operating system											

Table 1. Investigated strategies of allocating cores to the simulation process: Example with 5 cores.

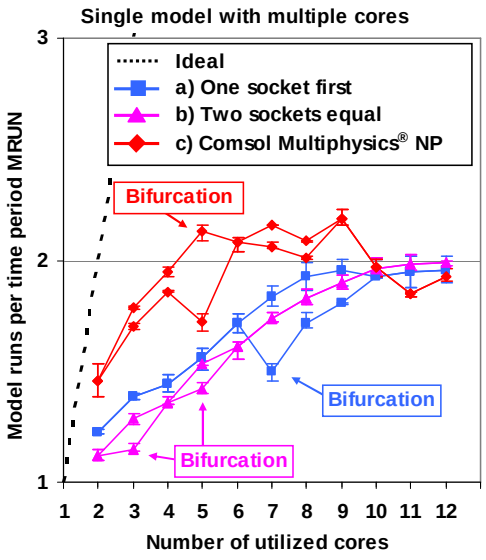

Figure 2. Model runs per time period MRUN by shared-memory parallelization of a single model with multiple cores and different allocation strategies.

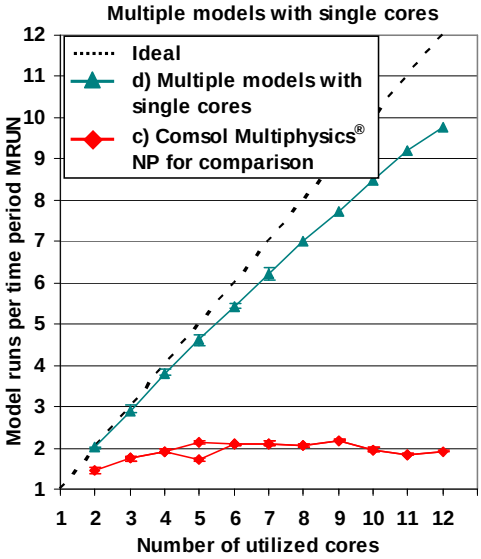

Figure 3. Model runs per time period MRUN by simultaneous simulation of multiple models with single cores (d), as compared to the best shared-memory parallelization strategy (c).

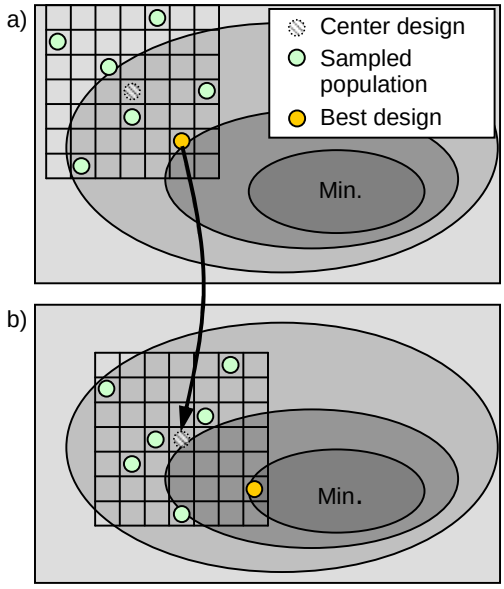

Figure 4. a) Latin hypercube sampling around a center design, b) best design is the center design of the next iteration.

4. Population-based Parallelization – Multiple Models, Single Core

A technique which avoids the inherent limitations of shared-memory parallelization is strategy

d) Simultaneous simulation of multiple models with single cores.

Again, affinity masks allocate single cores to the respective simulation processes. Runtime measurements show that this technique minimizes the interaction between the simultaneous simulation processes which results in an almost ideal behavior. The increase in MRUN for strategy (d) in Fig. 3 is close to the ideal graph and provides roughly 4 times higher peak performance than the best shared-memory strategy (c) without bifurcations.

5. Latin hypercube sampling optimizer

For a sequential optimization algorithm, the benefit of simultaneous simulation is not accessible. In a population-based approach, a set of different samples is generated in each iteration, which can be simulated simultaneously. There are various statistical (e.g. Monte Carlo) and systematic sampling methods (e.g. d-optimal or Koshal) to sample a part of the space of optimization parameters. In this work, latin hypercube sampling (LHS) is used, which enforces a beneficial, homogeneous distribution of samples in the parameter space. The principle is visualized for a 2d parameter space in Fig. 4a. Around the known center point, additional samples are distributed in a window following a latin square that ensures only one sample in each row and column. Given a sufficient number of samples, Latin hypercube sampling realizes the desired uniform distribution in the multidimensional parameter space.

The latin hypercube optimization algorithm LHSOpt takes the best sample of the current iteration as the center point for the sampling window of the next iteration (Fig. 4b). In the case of stagnating improvement of the objective value for the current iteration, the extension of the sampling window is reduced. Several termination criteria were defined:

1. The maximum number of iterations is exceeded
2. Stagnation of the position of the best design in parameter space
3. Stagnation of the objective value

For the given test model, the termination threshold for criterion (2) is defined by the positioning accuracy of parts in production and for criterion (3) by the accuracy of the model.

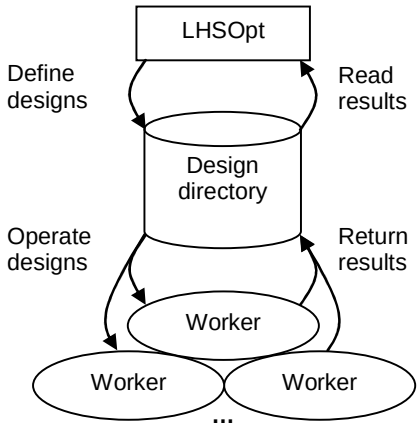

Figure 5. Independent workers (Comsol Multiphysics® servers) operate in parallel the designs which are scheduled by the optimizer LHSOpt.

LHSOpt was implemented in Matlab® but the basic requirements could be met by other platforms as well. The major part for parallelization is a scheduler that queues job definitions for all samples of each iteration in a shared design directory. In Fig. 5, the workflow of placing jobs and reading results is depicted. The parallelization is attained by launching multiple Comsol Multiphysics® servers with different ports and fixed allocation to single cores by affinity masks. Each Comsol Multiphyiscs® server starts a worker for concurrent operation on the design directory. During an iteration, each worker operates designs by

- Looking for an unprocessed design
- Locking the design
- Simulating the model
- Evaluating the objective
- Saving the result in the design directory
- Flagging the design as finished
- And releasing the design

When all designs of an iteration are finished, LHSOpt collects results and starts the next iteration. In principle, one can even start workers on several remote computers for joint operation on a design directory on the network. This allows the Comsol Multiphysics® user to address one optimization task with several single computer licenses.

6. Benchmark of optimizers

In this section, LHSOpt is benchmarked against the state-of-the-art sequential Nelder-Mead-Simplex algorithm fminsearch, which is supplied with Matlab®. For this purpose, 10 evenly distributed starting points were sampled in the parameter space to compare the convergence, results and duration of both optimizers for each starting point. As the global optimum and the objective at the starting point are known, the optimization result is expressed as achieved percentage of possible improvement. Optimization runs that terminate without sufficient result are regarded not converged. In Tab. 2, the results are summarized: The overview illustrates the general robustness of population-based optimization algorithms which find the global optimum – result 100% – regardless of the starting point. The sensitivity of fminsearch to the starting point is indicated by missing convergence for half of the starting points. For the converged starting points, LHSOpt always retrieves a better or equal result within less duration than fminsearch. The speed-up factor is up to a factor 3 but always less than the potential factor 4, enabled by the more efficient hardware utilization (see section 4). The reason for this finding is the general tendency of simple population-based approaches to compute many more unnecessary samples than systematic optimization algorithms like fminsearch.

Starting Point	Fminsearch Duration (h)	Fminsearch Result	LHSOpt Duration (h)	LHSOpt Result	Speed-Up Factor
1	9.6	✗	2.5	100%	
2	22.0	✗	5.7	100%	
3	6.0	✗	6.8	100%	
4	10.9	100%	3.7	100%	2.97
5	9.0	✗	4.4	100%	
6	4.9	93%	3.6	100%	1.35
7	8.8	✗	2.5	100%	
8	5.4	97%	5.1	100%	1.05
9	5.3	100%	3.7	100%	1.43
10	3.9	100%	3.7	100%	1.04
✗: No convergence				Average:	1.57

Table 2. Benchmark runs of fminsearch and LHSopt were performed for different starting points.

7. Conclusions

The presented study points out the limitations of speeding up sequential optimization by simulating a single model with multiple cores (shared-memory parallelization) and proves the performance advantage of a factor 4 by simultaneous simulation of multiple models with single core. Using a simple population-based optimization algorithm, up to 75% of this performance benefit was realized in benchmark optimization

runs vs. sequential optimization with shared-memory parallelization.

Beyond these encouraging findings, there is further potential for improvement: More sophisticated population-based approaches (e.g. adaptive response surface optimization) or parallelized systematic algorithms can access a larger fraction of the hardware performance benefits. Moreover, the presented optimization workflow enables a joint operation of several single user licenses on one optimization task. This approach is particularly interesting for the common situation of modeling teams with several existing single user licenses instead of a floating network license and a cluster for distributed computing.

8. References

1. Laurie J. Flynn: Intel Halts Development of 2 New Microprocessors, *New York Times*, May 8, (2004)

2. Daniel Molka, Daniel Hackenberg, Robert Schöne and Matthias S. Müller: Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System, *18th International Conference on Parallel Architectures and Compilation Techniques* (2009)