Uses of GPU Powered Interval Optimization for Parameter Identification in the Context of SO Fuel Cells ⋆

Stefan Kiel * **Ekaterina Auer** * **Andreas Rauh** **

* *University of Duisburg-Essen, 47057 Duisburg, Germany (e-mail: {kiel,auer}@inf.uni-due.de)*
** *University of Rostock, Justus-von-Liebig-Weg 6, 18059 Rostock, Germany (e-mail: andreas.rauh@uni-rostock.de)*

Abstract: In this paper, we discuss parameter identification for models based on ordinary differential equations in the context of solid oxide fuel cells. In this case, verified methods (e.g. interval analysis), which provide a guarantee of correctness for the computed result, can be of great help for dealing with the appearing uncertainty and for devising accurate control strategies. Moreover, interval arithmetic can be used to discard infeasible areas of parameter space in a natural way and so to improve the results of traditional numerical algorithms. We describe a simulation environment interfacing different verified and floating point based approaches and show how the interchangeability between different techniques enhances parameter identification. Additionally, we give details on a possible parallelization of our version of the global interval optimization algorithm on the CPU and the GPU. The applicability of the method and the features of the environment are demonstrated with the help of different fuel cell models.

Keywords: initial value problems; verified methods; parameter identification; GPU; software design; SOFC.

1. INTRODUCTION

Solid oxide fuel cells (SOFCs) are a promising research field in the area of decentralized energy supply. They convert chemical energy into electricity efficiently and are flexible with respect to the kind of fuel. Most state-of-the-art control strategies for SOFCs are developed under the assumption of stationary operating conditions. To take into account the instationary nature of SOFC operation, robust, accurate and adjustable control techniques are necessary. One possibility to construct them consists in using simplified, ordinary differential equations (ODEs) based models [DARA13]. This is a major goal of a project between the Universities of Rostock and Duisburg-Essen.

To achieve the aims of robustness and accuracy as well as to deal with the appearing uncertainty, we suggest using verified methods, in particular, interval analysis (IA) [AH83]. Such numerical methods provide a guarantee of correctness of the computed result. Usually, they produce an enclosure (e.g. an interval) for which it is proved that it contains the exact result. Additionally, a verified method allows users to propagate bounded epistemic uncertainty through their systems. In the context of SOFCs, a control strategy implemented with IA can guarantee that the operating temperature never leaves the admissible range and thus the expensive cell is not damaged by overheating. To support the design and validation process for these strategies, we develop the integrated software environment VeriCell. It differs from its commercial counterparts such as gFuelCell [1] by assuming ODE based models and interfacing a variety of interval algorithms. It also allows for the use of traditional floating point methods and facilitates adding new models or solvers.

An important problem which has to be dealt with in VeriCell is identification of model parameters. It is known to be computationally expensive, which makes the use of interval global optimization algorithms for this purpose challenging. Although control oriented models such as those developed in [DARA13] reduce the number of parameters in comparison to the traditional ones, they nonetheless depend on more than 20 parameters and must be optimized against more than 19000 measurements in out setting. We presented our first results on the parameter identification for SOFCs in [RDAA12] (where a specialized optimization algorithm is used) and in [AKR12] (where an adapted general purpose solver is applied to the problem).

In this paper, we discuss how the computational power of modern graphic processing units (GPUs) can be exploited to speed up the parameter identification procedure. Additionally, we demonstrate how the integrated environment VeriCell helps to identify better parameter sets by facilitating the interoperability between modern floating point and verified techniques. The paper is structured as follows. In Section 2, we give an overview of the considered SOFC models and the parameter optimization problem for them. In Section 3, the basic principles of global in-

⋆ This work was supported in part by the DFG (German Research Council).

[1] http://www.psenterprise.com/products/gfuelcell/index.html

Copyright © 2013 IFAC

terval optimization are discussed along with our current implementation. In the next section, we describe possible enhancements based on parallelization using general purpose computations on the GPU. The results (obtained in VeriCell, partly by combining verified and non-verified techniques) are discussed in Section 5. Conclusions and an outlook are in the final section of the paper.

2. SOFC MODELS AND PARAMETER IDENTIFICATION

The control oriented SOFC models we consider in this paper describe the temperature of the stack module. The ODEs we use are derived from partial differential equations by a local semi-discretization of the stack into $l_d \times m_d \times n_d$ finite volume elements. For the purpose of parameter identification in this paper, we rely mainly on the models obtained from $1 \times 1 \times 1$ and $1 \times 3 \times 1$ finite volume elements. The former describes the whole stack as a single volume element and results in a one-dimensional non-linear ODE for the output temperature. It provides no information about the temperature inside the stack. For the latter model, the stack module is split into three volume elements resulting in three non-linear ODEs. Both models depend on 23 parameters p (which have to be identified experimentally) and, additionally, on 8 time dependent input variables $u(t)$. The parameters stand for approximations of such temperature dependent quantities as the heat capacities of hydrogen, nitrogen or water vapor. We also touch upon the model resulting from the discretization into nine volume elements which is computationally less feasible in our context. For more details on the modeling, see [AKR12] (the one dimensional case) and [RDAA12, DARA13] (in general).

The identification algorithm tries to find parameter values for which the results obtained by solving the corresponding ODEs are as close as possible to the behavior of the real cell. The usual technique in this case is the least squares optimization between simulation and measurements. Formally, we define the optimization problem as

$$\min_p \varphi(p),\ \varphi(p) = \sum_{i=1}^{N}\sum_{j=1}^{M}\left(y(t_i,p)_j - y_m(t_i)_j\right)^2 \quad . \qquad (1)$$

Here, φ is the objective function and p the set of parameters to be identified. Functions $y(t_i, p)_j$ and $y_m(t_i)_j$ denote the simulated and measured temperature, respectively, for the state j at the time t_i, N is the total number of measurements, and M the number of measurable states. We assume w.l.o.g. that the first M components of $y(t,p) \in \mathbb{R}^{l_d \times m_d \times n_d}$ can be measured directly. Currently, we consider $N = 19964$ equidistant measurements (every second) and either one or two measurable states for the one-dimensional or higher dimensional models, respectively. The large number of measurements makes the evaluation of the objective function rather expensive computationally. For interval based global optimization, we limit the number of parameters to six (where we choose those with the highest impact). The others are initialized with values obtained by floating point methods.

The required computational effort and the accuracy also depend on how the initial value problem (IVP) arising from the corresponding model is solved. Since the analytical solution is not available even for the $1 \times 1 \times 1$ version, numerical methods have to be used. The simplest and cheapest possibility is to calculate an approximation to $y(t_i, p)_j$ by Euler's method with intervals (denoted by bold letters)

$$\boldsymbol{y}^{(k)} := \boldsymbol{y}^{(k-1)} + h \cdot f(\boldsymbol{y}^{(k-1)}, \boldsymbol{p}) \ , \qquad (2)$$

where f is the right-hand side of the initial value problem. The method takes into account rounding errors but ignores the IVP discretization error (verified approximation). Since the step size $h = 1\text{s}$ is two orders of magnitude smaller than the dominant time constant of the underlying process in our setting, Euler's method should provide an acceptable approximation. The objective function becomes

$$\varphi(\boldsymbol{p}) = \sum_{i=1}^{N}\sum_{j=1}^{M}\left(\boldsymbol{y}^{(i-1)}(\boldsymbol{p})_j + f(\boldsymbol{y}^{(i-1)}(\boldsymbol{p}), \boldsymbol{p})_j - y_m(i)_j\right)^2 \ , \qquad (3)$$

where $\boldsymbol{y}^{(0)}$ is the enclosure of the initial temperature at $t = 0$. If necessary, derivatives of φ can be computed using algorithmic differentiation.

A fully verified enclosure of the solution can be obtained by employing, for example, solvers VNODE-LP [Ned06] or ValEncIA-IVP [RA11]. In this case, the computational effort increases considerably. For instance, VNODE-LP needs 1441.68 seconds of CPU time on our test system (cf. Section 5) to compute the solution of the one-dimensional IVP over the whole time interval [0, N] (2661.21 seconds in nine dimensions). Euler's approximation needs only 0.11 seconds in the one-dimensional case. It can be shown that the maximum deviation of Euler's method from the verified enclosure derived by VNODE-LP is less than 0.22K for $h = 1\text{s}$. (Obviously, if we increase the step size, Euler's method deviates more.) In the rest of the paper, we explore the parameter identification using Euler's method for the approximation of the simulated temperature because the objective function evaluation is faster compared to verified solvers and the deviation for $h = 1\text{s}$ is acceptable. However, the use of verified solvers is also promising since we might be able to obtain fully verified results. This remains a topic for our future research.

In addition to the overall simulation error reduction by minimizing $\varphi(p)$, there is a need to make sure that the identified parameters do not lead to inconsistent states during the simulation. A state is considered to be inconsistent if for any time step t_i

$$\boldsymbol{y}(t_i, \boldsymbol{p})_j \cap (y_m(t_i)_j + \boldsymbol{\Delta}) = \emptyset, \qquad (4)$$

where $\boldsymbol{\Delta} = [-15, 15]\text{K}$ is the worst case measurement error and j is the index of a measurable state, e.g. the output temperature [RDAA12]. During the evaluation of the objective function (3), we check this condition and discard inconsistent states. Furthermore, the relation (4) is used to prune $\boldsymbol{y}(t_i, \boldsymbol{p})_j$ at each time step. The condition can be exploited also in affine arithmetic [AKR12] but we do not consider this here because an affine arithmetic library for the GPU is not available at the moment.

3. INTERVAL GLOBAL OPTIMIZATION

To solve the problem (1) with the objective function (3), we employ our framework UniVerMeC[2] [KLD13]. It

[2] Unified Framework for Verified GeoMetric Computations

Copyright © 2013 IFAC

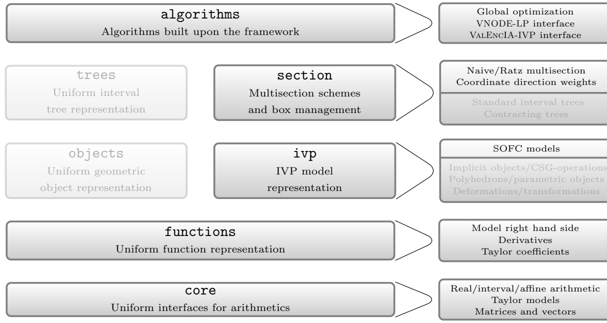

Fig. 1. Relaxed layered structure of the UniVerMeC framework and its application in VeriCell.

offers easy access to different kinds of range arithmetics, a uniform representation of functions independent of the underlying arithmetic for the use with different algorithms, and interfaces for various IVP solvers and optimizers. The framework is based on a relaxed layered architecture (i.e. layers not necessary for the task at hand can be left out) as depicted in Figure 1. The first layer `core` provides uniform access to different verified arithmetics such as IA, affine arithmetic [dFS97] or Taylor models [MB03]. Functions, for example, right sides of IVPs, are abstracted into their homogeneous, data type independent representations by the second layer. It provides interfaces for evaluating functions with all supported arithmetics of the `core` layer and a uniform scheme for their features such as derivatives or Taylor coefficients. The middle layer of the framework consists of several vertical problem dependent layers, of which we use `ivp` in this paper. It is responsible for representing IVPs uniformly and independently of data types for their later use with the available solvers without the need to adapt them to the respective interfaces which are not standardized for verified software. The fourth layer implements several multisection schemes which act as a utility data structure for the optimization algorithms implemented by the topmost layer (our own verified implementation as well as an interface for the floating point solver IPOPT [WB06]). The final layer additionally supplies interfaces to external IVP solvers. The environment VeriCell mentioned in the introduction provides a user friendly GUI for the SOFC-relevant features of the terminal-based UniVerMeC.

Our configurable global optimization algorithm is an adapted version of the one described in [HW04]. Like all interval branch and bound algorithms, it maintains a heuristically sorted working list $\mathcal{L}$ which contains parts of the search domain in the form of interval boxes. In each iteration, the algorithm tries to prune or discard the current box from $\mathcal{L}$ using different contractors. If the box could not be discarded, it is subdivided into smaller parts which are again inserted into the search list. Boxes satisfying a certain user-defined termination criterion are moved into the solution list $\mathcal{L}_{final}$. Widely used criteria are reaching a predefined minimum widths for the box or the objective function codomain as well as a combination of them. For more information about convergence properties, see [RR88, Chapter 3.9].

Figure 2 shows the algorithm implemented in UniVerMeC. To allow users to adapt the algorithm to their problems, it is subdivided into several stages (PHASE_ in the figure), the behavior of which can be changed individually. Following [HW04], we choose contractors and strategies for box reduction in dependence on the feasibility of the current box. That is, it is necessary to distinguish the stages PHASE_POS_INFEAS, PHASE_FEAS, and PHASE_STRICT_FEAS for boxes with unknown, certain and strictly certain feasibility, respectively. The contractors in the stages PHASE_A to PHASE_D are called independently of the feasibility of the current box in each iteration, while PHASE_SPLIT is called on boxes directly after the multisection step.

Input: Search region X_0
Output: Enclosure of minimum ϕ^*

1 $\mathcal{L} := \{X_0\}$; $S := RUNNING$;
2 **while** $S == RUNNING$ **do**
3 **if** $\mathcal{L} = \emptyset$ **then**
4   Get configuration for next phase;
5   **forall** $X' \in \mathcal{L}_{\text{tmp}}$ **do**
6   `apply_contractors`($PHASE_TMP$, X');
7   **end**
8   $\mathcal{L} := \mathcal{L}_{\text{tmp}}$; $\mathcal{L}_{\text{tmp}} := \emptyset$;
9   **if** $\mathcal{L} == \emptyset$ **then** $S := FINISHED$; **continue;**
10 **end**
11 $X := head(\mathcal{L})$; $\mathcal{L} := tail(\mathcal{L})$;
12 `apply_contractors`($PHASE_A$, X);
13 **if** $\neg$`feasible`(X) **then**
14   `apply_contractors`($PHASE_POS_INFEAS$, X);
15 **end**
16 `apply_contractors`($PHASE_B$, X);
17 **if** `feasible`(X) **then** `apply_contractors`($PHASE_FEAS$, X);
18 `apply_contractors`($PHASE_C$, X);
19 **if** `strictly_feasible`(X) **then**
20   `apply_contractors`($PHASE_STRICT_FEAS$, X);
21 **end**
22 `apply_contractors`($PHASE_D$, X);
23 **if** $\mathrm{w}(X) < \epsilon_t$ **then** $\mathcal{L}_{\text{tmp}} \leftarrow X$;
24 **else**
25   $\mathcal{N} :=$ `split`(X);
26   **forall** $X' \in \mathcal{N}$ **do**
27   `apply_contractors`($PHASE_SPLIT$, X');
28   **end**
29   $\mathcal{L} \leftarrow \mathcal{N}$;
30 **end**
31 **end**
32 **forall** $X' \in \mathcal{L}_{\text{final}}$ **do**
33 `apply_contractors`($PHASE_FINAL$, X');
34 **end**
35 $\underline{\phi^*} := \min_{X \in \mathcal{L}_{\text{final}}} (\underline{\phi(X)})$; $\overline{\phi^*} := \min(\max_{X \in \mathcal{L}_{\text{final}}} (\overline{\phi(X)}), \phi^*)$;

Fig. 2. Global optimization in UniVerMeC.

Inside a call to the `apply_contractors` routine, a box may be pruned, completely discarded or moved to $\mathcal{L}_{final}$ if it satisfies the termination criterion. In the latter two cases, the main loop is restarted. Unlike other global optimization algorithms, UniVerMeC maintains not only the lists $\mathcal{L}$ and $\mathcal{L}_{final}$, but also $\mathcal{L}_{tmp}$. If a box is subdivided below a certain minimum width ϵ_t, it is temporarily deleted from $\mathcal{L}$ and moved onto $\mathcal{L}_{tmp}$. This strategy ensures that the problem domain is subdivided more uniformly and prevents heuristics such as best-first from causing a deep subdivision in the wrong region. When $\mathcal{L}$ becomes empty, all boxes from $\mathcal{L}_{tmp}$ are moved back into $\mathcal{L}$, and the user can alter the algorithm stages again. In this way, accelerating devices such as interval Newton method can be configured dynamically.

The algorithm in Figure 2 can be parallelized on a shared-memory architecture, for example, on a modern multi-core system, in a straightforward way. Each thread gets

Copyright © 2013 IFAC

to process the `while` loop (lines 2-31 in Figure 2) for a certain part $\boldsymbol{x}$ of the search space. Because threads work exclusively on their respective boxes $\boldsymbol{x}$ at any time, all contractors can run in parallel as long as they are programmed to be reentrant. Special care is necessary for the shared data between the threads, for example, for the working list $\mathcal{L}$. Details on the shared memory parallelization with OpenMP [DM98] are given in [DK10].

For our simulations, we use the following configuration of the global optimization algorithm:

PHASE_SPLIT bounds the goal function with interval arithmetic and the test condition (4).
PHASE_PA tests the feasibility.
PHASE_POS_FEAS tests the box consistency on constraints.
PHASE_TMP tries to find a verified upper bound on the minimum using the midpoint test.

We use bound constraints $p_i' + [-1, 1]$ for the parameters which we want to identify. Here, p_i' is derived by means of a floating point method also used for the other 17 parameters which are not considered in the verified procedure. The maximum subdivision depth for the multisection scheme is 10^{-2}. After the optimization algorithm delivers the resulting candidates, we calculate the comparison measure

$$e = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\left(y(t_i, \operatorname{mid}\boldsymbol{p})_j - y_m(t_i)_j\right)^2}{N}}\,, \qquad (5)$$

where each box $\boldsymbol{p}$ is replaced by its midpoint $\operatorname{mid}\boldsymbol{p}$. The midpoint with the best measure e is assumed to be the best identified parameter set.

4. MODEL EVALUATION ON THE GPU

Recently, the use of GPUs has received increased attention in the area of scientific computing. These highly specialized units were originally designed for rendering with a *fixed* function pipeline. Today, GPUs are becoming more and more accessible for programming and thus suitable for general tasks. They offer cheap computational power if the problem to be solved can be mapped to the *stream processing model* which GPUs employ.

The stream processing model consists of data elements called *streams* and a program or a function called the *kernel* which is applied to each of the elements. The kernel should be parallelizable, that is, the results for one element should not depend on the results for any other element.

The data to be processed by the kernel have to be transferred from the CPU host program to the GPU global memory. Therefore, the kernel needs to be rather expensive computationally to make up for this input/output overhead. Further limitations on the kernel are imposed by the underlying SIMD (Single Instruction Multiple Data) architecture of the GPU. For example, diverging program branches can slow down the computation significantly.

The kernel should be implemented using a specialized programming language. Currently, CUDA [NVI12] and OpenCL [ope11] are widely used. While the former is a proprietary C dialect with some C++ extensions designed specially for NVIDIA graphic cards, the latter is an open

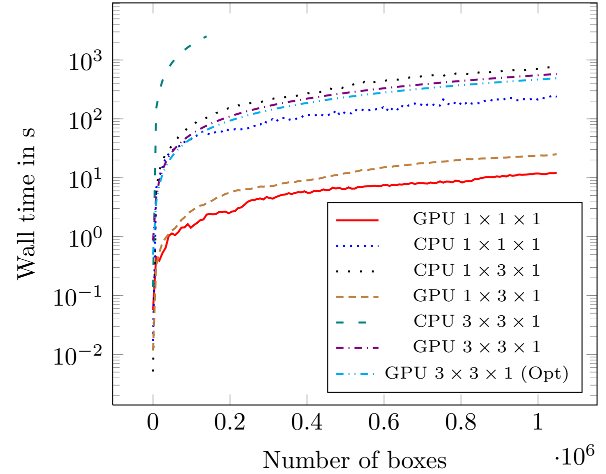

Fig. 3. Benchmarks for the evaluation of the objective function on the GPU and the CPU. The given time is the wall time on a logarithmic scale. For the GPU, it includes the necessary memory transfers.

standard that emerged out of the shader languages used to program GPUs in the computer graphics context. We use CUDA because it has better tool support. In particular, the CUDA SDK offers a ready-to-use IA library.

Note that it is crucial for verified implementations that the employed hardware supports directed rounding according to IEEE 754-2008. Modern GPUs, for example, those from NVIDIA beginning with the Fermi generation, comply with the standard [WFF11] well enough to perform interval computations. Currently, the above mentioned CUDA interval library implements only the basic arithmetic operations $\{+, -, *, /\}$ and the square root. However, this limited functionality is sufficient for us to evaluate the objective function φ. We cannot employ the GPU in the context of parameter identification to speed up a single evaluation of (1) because the values of $y(t_k, p)$ depend on those from the previous step $y(t_{k-1}, p)$. The GPU can be used to evaluate φ in parallel over parameter sets $p^{(1)}, \ldots, p^{(n)}$ which are generated by subdivision strategies of the global optimization algorithm.

The first step is to prepare the device. In this stage, the constant parameters (not to be optimized), the measurements, and the control variables are transferred to the GPU. These data are independent of the number of actual kernel launches during application runtime, so that this transfer is performed only once. After that, we have to move the interval boxes $\boldsymbol{p}^{(1)}, \ldots, \boldsymbol{p}^{()}$ from the CPU host memory to the GPU global memory for a specific kernel launch. Now the GPU kernel can be executed, which produces the values $\varphi(\boldsymbol{p}^{(i)})$ for $i = 1, \ldots, n$. Finally, the results need to be transferred back to the CPU. The transfer of the intervals to and from the GPU is done using their `double` endpoints as the exchange format. This ensures that no conversion error is introduced during this step.

Benchmark results for evaluations of the objective function in (3) for both models are shown in Figure 3. The simulations were performed on the test computer described in the next section. The benchmarks start with $n = 1$ boxes. In every step, we increase n by 8192 until $n > 1048576$. For the simple $1 \times 1 \times 1$ model, the GPU implementation has

Copyright © 2013 IFAC

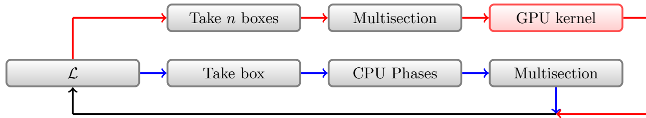

Fig. 4. Integration of the GPU based function evaluation into the optimization algorithm.

a speedup of approximately 19 compared to the parallel CPU one. In the $1 \times 3 \times 1$ case, the achieved speedup is 30. For the nine state model, we terminated the CPU benchmark at approximately 140000 boxes since it needed too long (over 41 minutes). We obtained a GPU speedup of approximately 33 compared to the CPU at this point. The reason for the better speedups (for $1 \times 3 \times 1$ and $3 \times 3 \times 1$) is the higher arithmetic density of the nine state model in comparison to the simple one. Both models use the same data so their input/output overhead for transferring data from and to the GPU and global memory accesses on the GPU is the same. These results are obtained with kernels not especially optimized for the GPU. An optimized version for the $3 \times 3 \times 1$ model featuring asynchronous copies through CUDA streams, register optimizations, exploitation of the memory hierarchy and coalesced memory accesses performed approximately 1.18 times better. The minor speedup indicates that the kernel runtime is used mainly for arithmetic operations.

The integration of the GPU based evaluation of φ into the global optimization algorithm is shown in Figure 4. Basically, one of the CPU threads is responsible for feeding the GPU with data. At every step, it takes n boxes from the global working list $\mathcal{L}$ and performs a multisection step on them. After that, the boxes are transferred to the GPU and the objective function evaluation is performed there. The boxes which are not discarded during this step are copied back to $\mathcal{L}$. The other threads proceed as in the standard CPU version.

5. RESULTS

We identified the parameters of the models derived from $1 \times 1 \times 1$ and $1 \times 3 \times 1$ finite volume elements on our test computer (a Xeon CPU E5-2680 with 8 cores, 64 GB RAM, NVIDIA GeForce GTX 580 graphics card with 512 CUDA cores, Linux). The code was compiled with gcc 4.7.2 with -O2 optimizations. We used OpenMP for parallelization on the CPU and the CUDA toolkit 4.2 for the GPU. The evaluation of the objective function with the solution to the $3 \times 3 \times 1$ model approximated by Euler's method is difficult for IA since there is a large overestimation, and the criterion (4) can be used only for the two measurable states (out of nine). Tighter enclosures can be computed by verified IVP solvers inside the objective function. However, the CPU times would increase significantly and the GPU employment for the evaluation would not be possible (as long as there is no verified solver implemented on the GPU).

Our identification procedure consists of four steps. First, we compute an initial approximation using a floating point optimization method (IPOPT or `fminsearch` in our case). Note that `fminsearch` is derivative free (an implementation of the Nelder-Mead simplex method in MATLAB) whereas IPOPT implements an interior point algorithm

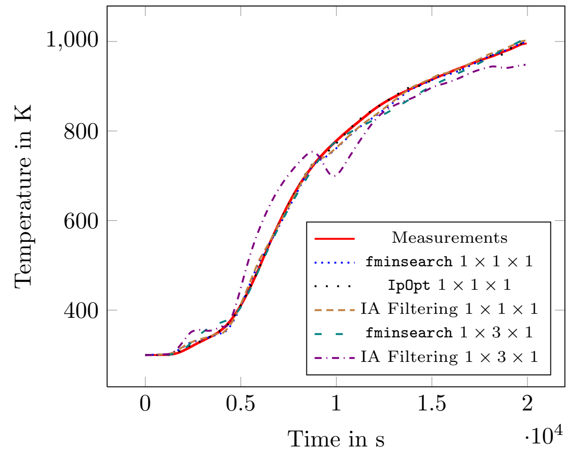

Fig. 5. The upper bound of the modeled temperature for the identified parameter sets in comparison to the measured values for the first measurable state.

requiring second order derivatives. Since it is interfaced in UniVerMeC, we are able to supply the exact first derivative to IPOPT using algorithmic differentiation. It would also be possible to compute exact second derivatives but it takes too much CPU time since the objective function is too complex (Euler's IVP solution approximation at the step k depends on those at all previous steps $0, \ldots, k-1$, all of which depend on the parameters p). That is why we rely on the Hessian approximation method provided by IPOPT. The 23 parameters were initialized with zeros. In the second step, we use a GPU enhanced interval optimization algorithm for six parameters to filter the area around the initial approximation from Step 1 in order to obtain better parameter sets. The regions discarded at this step are guaranteed to be infeasible with respect to the condition (4) and the objective function (3). After Step 1 or/and Step 2, we compute the comparison measure from (5) to select the candidate with the smallest e. In the fourth and final step, we validate the chosen parameter set by running a verified simulation with it in VNODE-LP and checking for the two measurable states whether the obtained curves are inside the bounds given by the condition (4) in each time step.

The statistics on simulations are given in Table 1. We record the comparison measure e and the CPU wall time for purely floating point parameter identification (Step 1) and for the interval based GPU enhanced filtering initialized with these values (Step 2). Note that since `fminsearch` is implemented in MATLAB and not in C++ like all the other components, the CPU wall times for it are larger in the purely floating point part. Additionally, we specify in the columns titled 'consistency' if the obtained parameter set is actually consistent according to the condition (4) (cf. Step 4). Since the parameter set obtained by IPOPT for the one dimensional model directly leads to consistent results, no interval filtering is necessary. The parameter set obtained by `fminsearch` for the one dimensional model can be improved by the interval filtering. For the three dimensional model, the IPOPT performance is rather bad, therefore we did not run interval filtering with this initial estimation. The parameter set obtained with `fminsearch` for the three dimensional model could not be improved by interval filtering under the assumption that

Copyright © 2013 IFAC

Table 1. Statistics on simulation runs during parameter identification for two SOFC models.

	purely floating point			with interval GPU filtering		
Model	e	wall time	consistency	e	wall time	consistency
IPOPT, $1 \times 1 \times 1$	2.39 K	555.5 s	yes	unnecessary		
`fminsearch`, $1 \times 1 \times 1$	8.24 K	$\approx$ 21600 s	no	7.84 K	4130.86 s	no
IPOPT, $1 \times 3 \times 1$	430.55 K	–	no	initial estimation too bad		
`fminsearch`, $1 \times 3 \times 1$	37.57 K	–	no	57.31 K	5021.06 s	no

the subdivision depth is not to exceed 10^{-2} and the maximum iterations number is less than 200001. In Figure 5, the simulated temperature curves for the first measurable state are shown in comparison to the measured values. As expected, the simulation for the consistent parameter set approximates the curve corresponding to the measured values in the best way.

6. CONCLUSIONS AND OUTLOOK

In this paper, we described a GPU enhanced global interval optimization algorithm in the scope of parameter identification for SO fuel cells. It discards infeasible parts of the search domain in a guaranteed way and helps to improve the parameter estimations obtained by floating point methods (cf. results for `fminsearch` and the $1 \times 1 \times 1$ model). The use of the GPU allowed us to speed up the objective function evaluation (and thus the algorithm itself) significantly. Besides, we were able to identify better parameter sets (compared to our previous publications) and validate their consistency by exploiting the interchangeability between floating point and interval methods provided inside the software VeriCell. For example, it could be proved with the help of the verified IVP solver VNODE-LP that the temperature simulated with the one dimensional model for the parameter set obtained by the floating point optimizer IPOPT was inside the acceptable bounds of ± 15 K with respect to the measured value at each point of time. Our future work will be directed toward full verification. In particular, we plan to employ verified IVP solvers inside the objective function to compute its enclosure and the enclosure of its derivative, the basis for which is already laid down by the internal structure of VeriCell. This might reduce the overestimation and allow the optimizing algorithm to verify the global minimum. However, this would also increase the computation time and make the employment of the GPU much less straightforward. A further direction is the GPU based implementation of further parts of the interval global optimization algorithm, since it is well suited for this kind of parallelization. Currently, the implementation is made difficult by the lack of necessary tools (e.g. algorithmic differentiation libraries for intervals on the GPU). As soon as they are available, we can move away from the naive interval evaluation and calculate better enclosures of the objective function on the GPU, for example, by centered forms, which would improve the quality of the interval filtering significantly.

REFERENCES

[AH83] G. Alefeld and J. Herzberger. *Introduction to interval computations*. Academic Press, New York, 1983.

[AKR12] E. Auer, S. Kiel, and A. Rauh. Verified parameter identification for solid oxide fuel cells. In *Proc. of REC 2012*, 2012.

[DARA13] T. Dötschel, E. Auer, A. Rauh, and H. Aschemann. Thermal behavior of high-temperature fuel cells: Reliable parameter identification and interval-based sliding mode control. *Soft Computing*, 2013.

[dFS97] L.H. de Figueiredo and J. Stolfi. *Self-Validated Numerical Methods and Applications*. IMPA, Rio de Janeiro, 1997.

[DK10] E. Dyllong and S. Kiel. Verified distance computation between convex hulls of octrees using interval optimization techniques. *PAMM*, 10(1):651–652, 2010.

[DM98] L. Dagum and R. Menon. OpenMP: an industry standard API for shared-memory programming. *Comp. Sci. Eng.*, 5(1):46 –55, 1998.

[HW04] E. Hansen and G. W. Walster. *Global Optimization Using Interval Analysis*. Marcel Dekker, New York, 2004.

[KLD13] S. Kiel, W. Luther, and E. Dyllong. Verified distance computation between non-convex superquadrics using hierarchical space decomposition structures. *Soft Computing*, 2013.

[MB03] K. Makino and M. Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 4(4):379–456, 2003.

[Ned06] N.S. Nedialkov. VNODE-LP: A validated solver for initial value problems in ordinary differential equations. Technical Report CAS-06-06-NN, McMaster University, 2006.

[NVI12] NVIDIA. CUDA C Program. Guide 4.2, 2012.

[ope11] The OpenCL specification v. 1.2 rev. 15, 2011.

[RA11] A. Rauh and E. Auer. Verified simulation of ODEs and DAEs in ValEncIA-IVP. *Reliable Computing*, 5(4):370–381, 2011.

[RDAA12] A. Rauh, T. Dötschel, E. Auer, and H. Aschemann. Interval methods for control-oriented modeling of the thermal behavior of high-temperature fuel cell stacks. In *Proc. of SysID 2012*, 2012.

[RR88] H. Ratschek and J. Rokne. *New computer methods for global optimization*. Ellis Horwood series in mathematics and its applications. Horwood, 1988.

[WB06] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, 2006.

[WFF11] N. Whitehead and A. Fit-Florea. Precision & performance: Floating point and IEEE 754 compliance for NVIDIA GPUs. Technical report, NVIDIA, 2011.

Copyright © 2013 IFAC