Solid Oxide Fuel Cell Systems — Identification

Ekaterina Auer

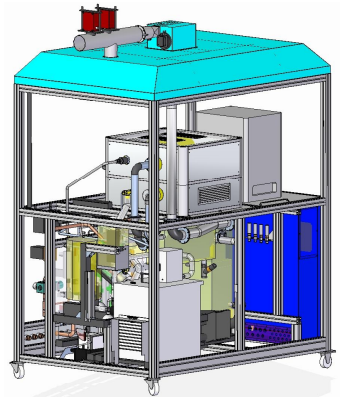
University of Technology, Business and Design Wismar

July 14, 2015

Parameter Identification for SOFC

SOFC: Devices converting chemical energy into electricity

Models: ODEs with unknown parameters to be identified on the basis of real-life data



Identification: Least squares/Global optimization

$$\Phi(p) = \sum_{k=1}^{T} \sum_{j=1}^{n_m} \Big(\underbrace{y_j(t_k, p)}_{\text{solution to ODEs}} - \underbrace{y_{j,m}(t_k)}_{\text{measured data}} \Big)^2 \to \min$$

Tools: UniVerMeC and VeriCell

Types of models: (V&V analysis)

- the used arithmetics (double/interval/other)
- "accuracy" of the solution $y_j(t_k, p)$ (analytic/approximated/exact)

How to Obtain Reliable $y_j(t_k, p)$ Numerically?

The Euler method

- $\mathbf{y}_k := \mathbf{y}_{k-1} + h \cdot f(\mathbf{y}_{k-1}, \mathbf{p})$
- – "Verified approximation" (rounding)
- – Overestimation
- \+ Easy derivatives (AD)
- \+ *Easily portable to the GPU*

Verified IVP solvers

- $y(t_k, p) \in \mathbf{y}_k$
- VNODE-LP, ValEncIA-IVP, etc.
- \+ Verifies the whole model
- – Derivatives require solving an extra ODE
- – High computational effort

Another possibility is to simplify the models on order to obtain analytical solutions, which is possible if heat capacities of gases are modeled with polynomials of order zero or one

In UniVerMeC, we can implement all these three possibilities

Forms of Computerized Models P1.P2

$$\Phi(p) = \sum_{k=1}^{T} \sum_{j=1}^{n_m} \left(y_j(t_k, p) - y_{j,m}(t_k)\right)^2 \rightarrow \text{min}$$

P1: The way the simulated solution $y(t_k, p)$ of the IVP is obtained

1. $y(t_k, p)$ is computed analytically
2. $y(t_k, p)$ is approximated by an analytic expression (e.g. Euler method) and the approximation error is neglected
3. $y(t_k, p)$ is computed using a "black box" numerical solver

P2: The kind of used arithmetic

1. traditional floating point arithmetic
2. interval arithmetic
3. affine arithmetic, Taylor models, etc.

\+ different numbers of volume elements ($\approx$ the dimension)

A Regression: What is Verification and Validation Analysis?

Verification and validation are necessary to safely utilize simulation results

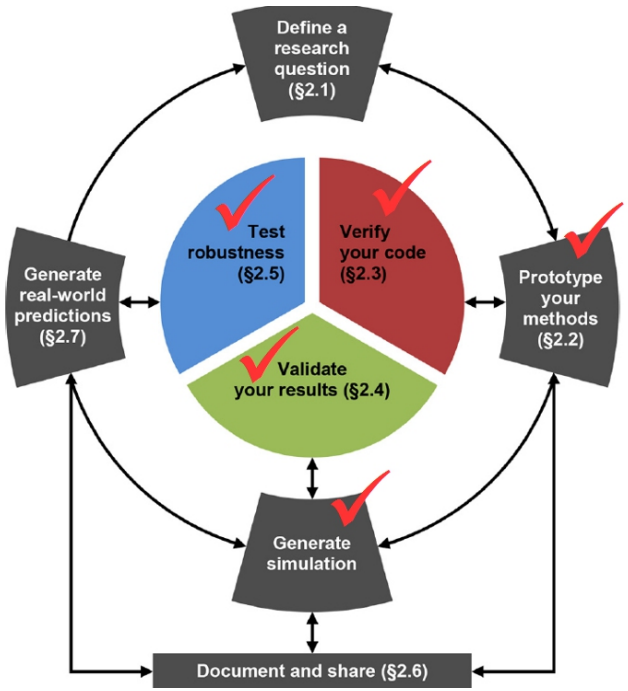
Verification: Are we building the computerized model correctly?

Validation: Did we build the correct model?

Traditional V&V

Beginnings:	In the area of computational fluid dynamics
Standards:	Generally: no true standard, only *guidelines* Software V&V: IEEE 1012
Approaches:	Formal methods for mission-critical tools ($\rightsquigarrow$ expensive, need simplifications) Syntactic methods otherwise
Fact:	Result verification can help!

Traditional View on V&V



Stage 2: Design the methods and create a V&V plan, in particular

- eliminate unnecessary complexity
- identify modeling assumptions
- address uncertainty in experimental data

Stage 3: Verify your software (*code* verification)

- unit testing
- higher-level tests design
- employment of existing software modules

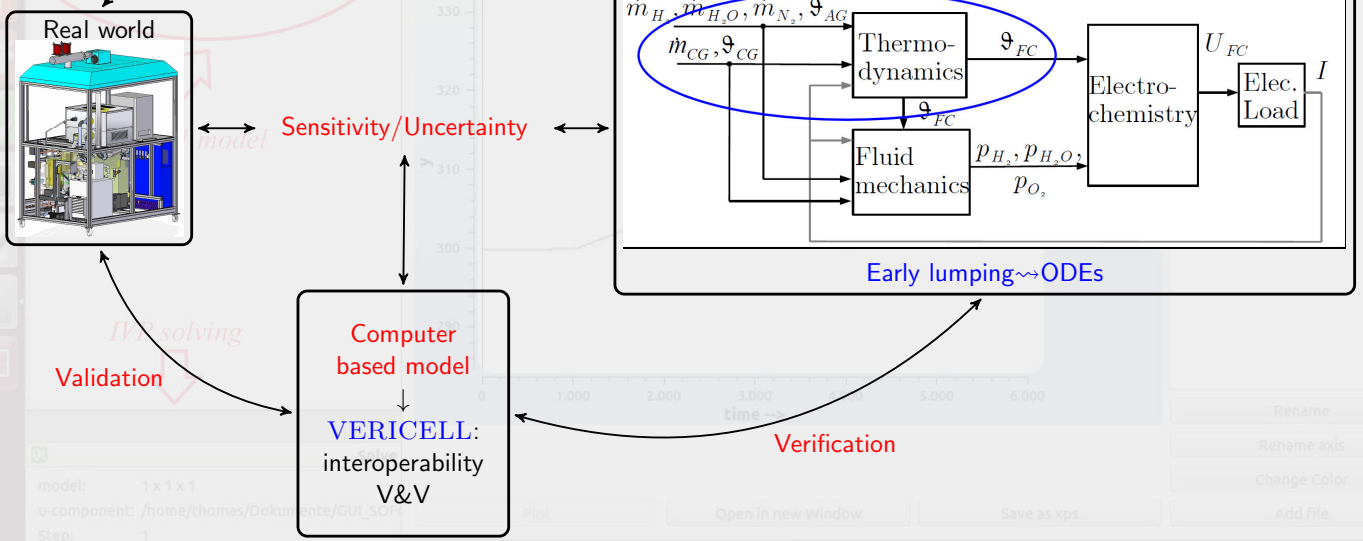
What about the result?

Stages 4-5: Compare the outputs of the simulation to many datasets and evaluate sensitivity

- confidence intervals
- outputs sensitive to inputs

Figure from: Hicks et al., Is My Model Good Enough? DOI: 10.1115/1.4029304

V&V Cycle for SOFC



Interoperability

- Models added "on the fly"
- Verified and non-verified methods combined
- Different verified tools used in parallel

V&V analysis

- Sensitivity/uncertainty via verified methods
- Accurate validation possibilities

Helpful: Verification Degree

From the lowest to the highest verification degree:

- C4 Standard floating (fixed) point arithmetic
- C3 IEEE 754 arithmetic, traditional sensitivity (e.g. Monte-Carlo)
- C2 Subsystems verified
- C1 The whole process verified (IEEE 754/P1788)

Additionally alongside the degree:

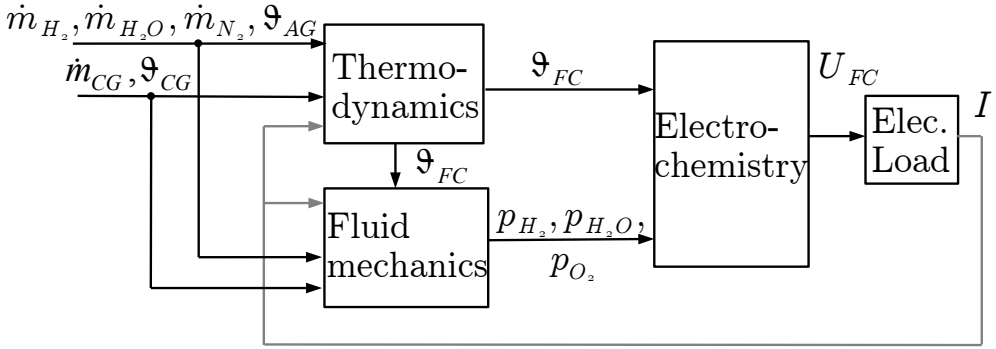
- $+$ code verification
- $-$ no uncertainty/sensitivity analysis

$Q_{\text{id}}q$ a quality indicator, $q \in [0, 1]$, id$\in \{\underline{n}ominal, \underline{u}ncertain\}$,

A V&V Tool: Questionnaire

Description of input data $\rightsquigarrow$ Tolerance of measurements

Initial temperatures ϑ_{CG}, ϑ_{AG} $(n_m = 2)$, mass flows of gases $\dot{m}$



Source: sensors (Eurotherm, Bronkhorst)

Description: ASCII file

Pre-selection: low-pass filter

Accuracy: four digits for ϑ $\pm$1K;
$\pm 0.5\%$ of value ± 0.1 of range for $\dot{m}$

Representation: IEEE-754 double precision

A V&V Tool: Questionnaire

Description of models $\rightsquigarrow$ Verification degree

$\rightarrow$ were/will be given by Andreas and Luise

Formulas: M: ODEs (dim=1,3,9) with different arithmetics (FP/I/...)

P: Parameter identification (1–3)

$$\Phi = \sum_{k=1}^{T} \sum_{j=1}^{n_m} \left(y_j(t_k, p) - y_{j,m}(t_k) \right)^2 \rightarrow \text{min}$$

Parameters: heat capacities of gases, enthalpies

A V&V Tool: Questionnaire

Description of algorithms $\rightsquigarrow$ Verification degree

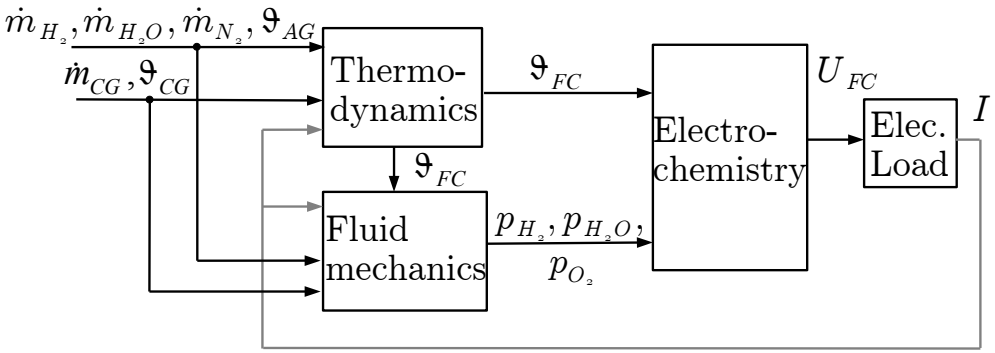
Depends on the kind of arithmetic and the type of identification

For example: M1.P2.A2 $\rightsquigarrow$ GlobOpt (C2)

Type:	global optimization, numeric, iterative
Parallelization:	Φ parallelized for multi-cored CPUs and the GPU
Operations:	interval-based for $+, -, \cdot, /$, sqr
Sub-algorithms:	algorithmic differentiation
Sensitivity:	wide search spaces, many parameters

A V&V Tool: Questionnaire

Description of output data $\rightsquigarrow$ Validation



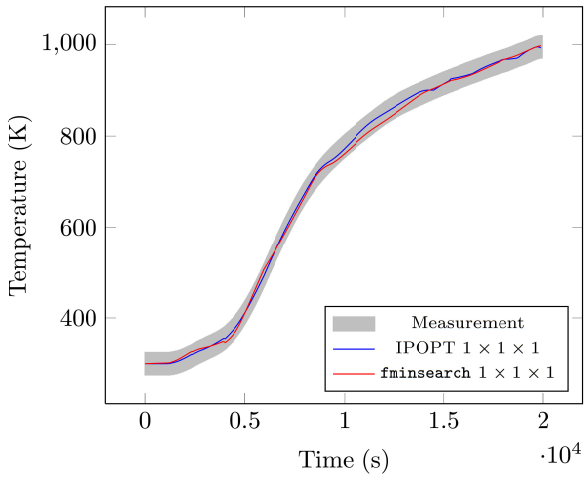
Depends on the kind of arithmetic

Data type/Accuracy: doubles or intervals

High accuracy: adjustment of measured data and simulations for different operating conditions

Exchange: ASCII file of double precision numbers

An Interoperable Scenario: Validation for the SOFC Temperature Model



1. Obtain values for parameters with double precision (e.g. in IPOPT)
2. Simulate ODEs with result verification
3. Check in a verified way, whether the trajectories are contained inside a confidence interval provided by experts

- No switching between tools necessary
- Verification degrees C3, C2
- Reliable validation possible:
  - → Proved that only the parameters obtained by IPOPT are consistent

Global Optimization Strategy

Branch-and-bound based on the Hansen method*

Basic pattern

1. $\mathbf{p} \leftarrow \mathcal{L}$
2. Discard $\mathbf{p}$ if it is infeasible
3. Discard $\mathbf{p}$ if $\underline{\Phi(\mathbf{p})} > \overline{D}$
4. Contract $\mathbf{p}$
5. Update $\overline{D}$
6. Add $\mathbf{p}$ to $\mathcal{L}_{\text{final}}$ if termination criteria are satisfied
7. Split $\mathbf{p}$ and add new boxes to $\mathcal{L}$

Features

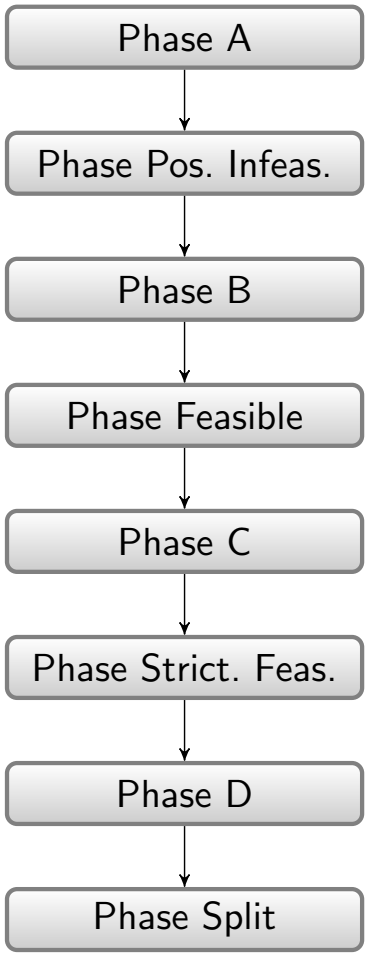
- flexible thanks to UniVerMeC
- derivative-free, if desired
- parallelized $\rightarrow$ fast

Termination criteria

- $\mathrm{w}(\mathbf{p}) \leq \epsilon_p, \epsilon_p > 0$
- $\mathrm{w}(\Phi(\mathbf{p})) \leq \epsilon_\Phi, \epsilon_\Phi > 0$

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

A Configurable Optimization Algorithm in UniVerMeC



Phase A
Use the midpoint test

Phase Feasible
Update the upper bound

Phase D
Linearize and prune using the consistency constraint

Phase Split
Calculate bound on $\Phi(\mathbf{p})$
Check for (in)consistent states

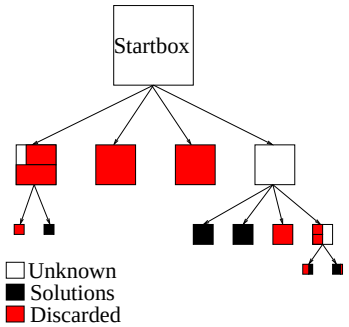
Parallelization: Multicore CPU

Characteristics

A lot of local work

→ objective function evaluation, midpoint test

Workload sharing



The B&B tree is unbalanced

No a priori subproblems

Take the available box from $\mathcal{L}$

Parallelization (OpenMP)

$\mathcal{L}$ is shared between all threads

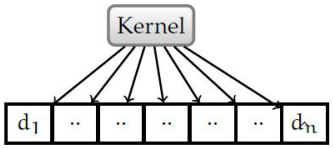
→ for shared-memory only, a bottle neck

GPU-Powered Computations

GPUs are highly specialized programmable units for rendering

Programming language: CUDA (Compute Unified Device Architecture), specialized for NVIDIA graphic cards

→ Better overall tool support



Stream processing model: the same kernel is applied to all data elements, the order in which the data elements are processed is not defined

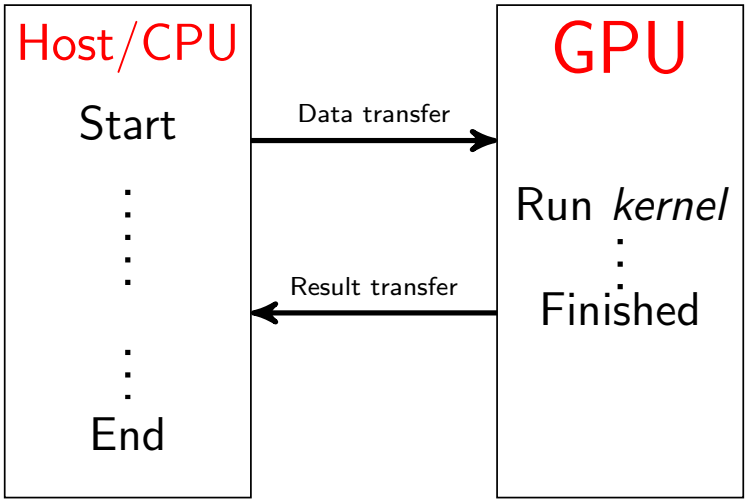
SIMD GPUs employ a single instruction multiple data model (in CUDA, programmers do not need to cope with diverging branches manually)

→ A large amount of branching ist not well suited for GPU in general

Verification angle: Not yet widely supported on the GPU (GPUs are reported to still produce random arithmetic errors!)

Exception: BOOST.INTERVAL library (an improved version is supplied with the CUDA toolkit)

Parallelization: the GPU



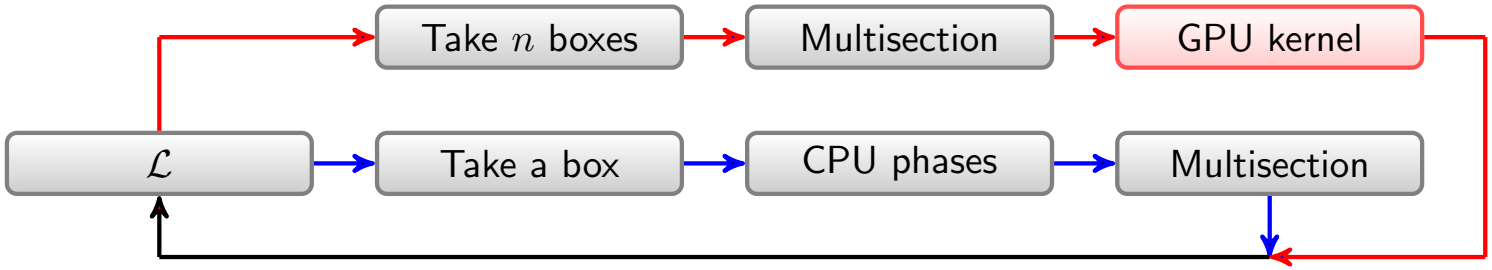
Features:

- the CPU starts the GPU part
- Data transfer is expensive (long latency)
- the CPU can proceed while the GPU computes

→ Parts not well suited for the GPU are executed on the CPU

→ Perform as many computations as possible on the transferred data

Integration into the Optimization Algorithm

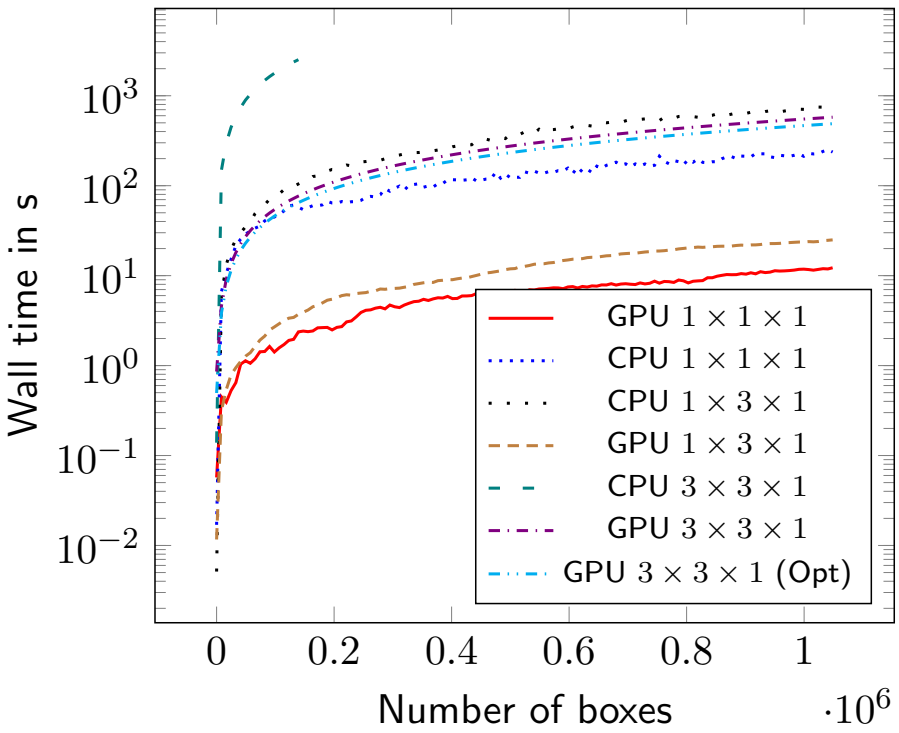


Possible approach:

- Run the GPU and the CPU in parallel
- Store the working list $\mathcal{L}$ in the host memory
- One CPU thread feeds the GPU with data

  $\rightarrow$ currently, only bounds on Φ are derived using the GPU
- The other CPU threads work normally

Benchmarks for Computing the Objective Function



Reference system

Xeon E5-2680, 8 cores
gcc 4.7 on Linux
GeForce GTX 580, 512 cores
CUDA 4.2

Results

$1 \times 1 \times 1$: speedup of 19

$1 \times 3 \times 1$: speedup of 30

$3 \times 3 \times 1$: speedup of 33
(employs much more arithmetic operations on the same input data!)