Parallel Multiple-Shooting and Collocation Optimization with OpenModelica

Bernhard Bachmann[3], Lennart Ochel[3], Vitalij Ruge[3],
Mahder Gebremedhin[1], Peter Fritzson[1], Vaheed Nezhadali[2], Lars Eriksson[2], Martin Sivertsson[2]

[1]PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
[2]Vehicular Systems, Dept. Electrical Engineering
Linköping University, SE-581 83 Linköping, Sweden
[3] Dept. Mathematics and Engineering, University of Applied Sciences,D-33609 Bielefeld, Germany

{bernhard.bachmann,lennart.ochel,vitalij.ruge}@fh-bielefeld.de,
{peter.fritzson,mahder.gebremedhin,vaheed.nezhadali,lars.eriksson,marsi}@liu.se

Abstract

Nonlinear model predictive control (NMPC) has become increasingly important for today's control engineers during the last decade. In order to apply NMPC a nonlinear optimal control problem (NOCP) must be solved which in general needs high computational effort.

State-of-the-art solution algorithms are based on multiple shooting or collocation algorithms, which are required to solve the underlying dynamic model formulation. This paper describes a general discretization scheme applied to the dynamic model description which can be further concretized to reproduce the multiple shooting or collocation approach. Furthermore, this approach can be refined to represent a total collocation method in order to solve the underlying NOCP much more efficiently. Further speedup of optimization has been achieved by parallelizing the calculation of model specific parts (e.g. constraints, Jacobians, etc.) and is presented in the coming sections.

The corresponding discretized optimization problem has been solved by the interior optimizer Ipopt. The proposed parallelized algorithms have been tested on different applications. As industrial relevant application an optimal control of a Diesel-Electric power train has been investigated. The modeling and problem description has been done in Optimica and Modelica. The simulation has been performed using OpenModelica. Speedup curves for parallel execution are presented.

Keywords: Modelica, Optimica, optimization, multiple shooting, collocation, parallel, simulation

1 Introduction

This paper presents efficient parallel implementations and measurement results of solution methods for nonlinear optimal control problems (NOCP) relevant for nonlinear model predictive control (NMPC) applications.

NMPC as well as NOCP have become increasingly important for industrial applications during the last decade [3], [4]. State-of-the-art solution algorithms [4] are based on multiple shooting or collocation algorithms, which are needed to solve the underlying dynamic model formulation. This paper concentrates on parallelizing these time-consuming algorithms, which finally lead to a very fast solution of the underlying NOCP. Moreover, a general discretization scheme applied to the dynamic model description is introduced, which can be further concretized to reproduce the common multiple shooting or collocation approach [7] and can also be refined to represent total collocation methods [4] in order to solve the underlying NOCP much more efficiently. The modeling and problem description is done in Modelica [2] extended with optimization goal functions and constraints specified as in Optimica [15]. The simulation is performed using OpenModelica [1]. Speedup curves for parallel execution are presented for application examples.

Section 2 describes the underlying mathematical problem formulation including the objective function and constraints to the state and control variables. The general discretization scheme applied is discussed in Section 3. This approach can be further refined to represent multiple shooting or collocation algorithms for the solution process, which is described in Section 4.

In section 5 the general discretization scheme is further developed towards total collocation methods.

Industrial relevant Modelica applications are presented in Section 6. Parallel execution of the constraint equations of the NOCP is performed in Section 7. The results show reasonable speedups of the optimization time when it comes to time consuming calculation of the model equations. The necessary implementations are partly realized in the OpenModelica Compiler, which is described in Section 8. The paper concludes with a summary of the achieved results.

2 The Nonlinear Optimal Control Problem (NOCP)

The numerical solution of NOCP is performed by solving the following problem formulation [7][8]:

$$\min_{u(t)} J(x(t),u(t),t) = E_{t_f}\left(x(t_f)\right) + \int_{t_0} L(x(t),u(t),t)\,dt \tag{2.1}$$

subject to

$$x(t_0) = h_0 \tag{2.2}$$
$$\dot{x}(t) = f(x(t),u(t),t) \tag{2.3}$$
$$g(x(t),u(t),t) \geq 0 \tag{2.4}$$
$$r\left(x(t_f)\right) = 0 \tag{2.5}$$

where $x(t) \in \mathbb{R}^{\eta_x}$ and $u(t) \in \mathbb{R}^{\eta_u}$ are the state and control variables, respectively. The receding time horizon is given by the interval $[t_0, t_f]$. The constraints (2.2), (2.3), (2.4) and (2.5) describe the initial conditions, the nonlinear dynamic model description based on differential algebraic equations (DAEs, Modelica), the path constraints ($g(x(t),u(t),t) \in \mathbb{R}^{\nu}$) and the terminal constraints.

Support for time-optimal control and corresponding terminal constraints is work-in-progress and are not yet provided by the current implementation.

2.1 Boundary Value Problems

The objective function (2.1), that needs to be minimized, includes conditions at the boundary time point t_f stated by the function $E\left(x(t_f)\right)$ as well as conditions taking into account the whole time horizon stated by $\int_{t_0}^{t_f} L(x(t),u(t),t)\,dt$.

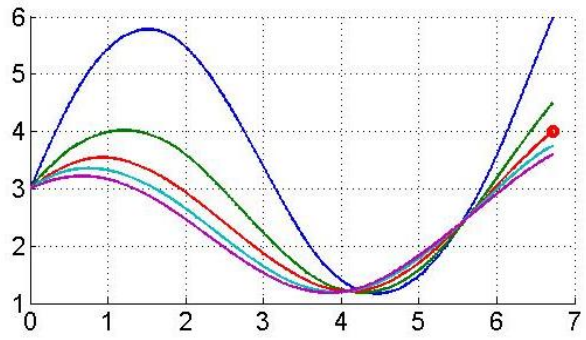

Figure 1. Different trajectories achieved by varying control variables. Only one trajectory fulfills the terminal constraint (red dot).

The function $E\left(x(t_f)\right)$ describes conditions that should be fulfilled at the final time point similar to the terminal constraint (2.5). Since $E\left(x(t_f)\right)$ is part of the objective function $J(x(t),u(t),t)$ the applied optimization methods may not find a solution that fulfills the corresponding terminal constraints, but should be very close to it. The trajectories are influenced by changing the control variables. Different trajectories using different control variables are visualized in **Figure 1**.

On the other hand, different trajectories could fulfill the same terminal constraints. Taking into account the whole time horizon by minimizing the second part $\int_{t_0}^{t_f} L(x(t),u(t),t)\,dt$ of the objective function will lead to the selection of the optimal trajectory. This behavior is visualized in **Figure 2**.

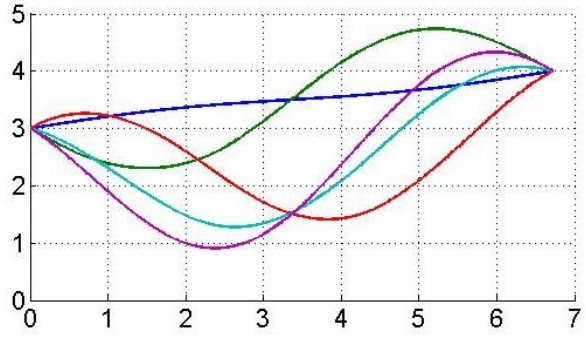

Figure 2. Different trajectories that fulfill the terminal constraint.

3 General Discretization Scheme

In order to apply a general discretization scheme the NOCP formulation is rewritten to a general form which later can be used to derive the different possible numerical algorithms e.g. multiple shooting, multiple or total collocation algorithm, etc. [6]. Equations (2.2) and (2.3) can be rewritten as follows:

$$x(\tau) = h_0 + \int_{t_0}^{\tau} f(x(t),u(t),t)\,\mathrm{dt}. \tag{3.1}$$

When discretizing the time horizon $[t_0, t_f]$ into a finite number of intervals $[t_0, t_1], \ldots, [t_{n-1}, t_n]$ (e.g. equidistant partitioning: $t_j := t_0 + l \cdot j,\ j = 0, \cdots, n,\ l := \frac{t_f - t_0}{n}$) integral in (3.1) can be reformulated to

$$\int_{t_0}^{\tau} f(x(t), u(t), t)\ \mathrm{dt} = \sum_{i=0}^{n-1} \int_{t_i}^{t_{i+1}} f(x(t), u(t), t)\ \mathrm{dt}. \tag{3.2}$$

Each integral

$$\int_{t_i}^{t_{i+1}} f(x(t), u(t), t)\ \mathrm{dt} \tag{3.3}$$

on a subinterval can now be treated independently, if additional constraints are added to the NOCP formulation to force the calculation of an overall continuous solution. Therefore, locally the problem reduces to a boundary value problem [5] stated by

$$x_i(t_{i+1}) = h_i + \int_{t_i}^{t_{i+1}} f(x_i(t), u(t), t)\ dt \tag{3.4}$$

where $x_i(t) := x(t)$ for $t \in [t_i, t_{i+1}]$, $i = 0, \cdots, n-1$. It yields $x_i(t_i) = h_i$ and continuity is forced by additional constraints $x_i(t_{i+1}) = h_{i+1}$ added to the NOCP formulation, which finally leads locally to a boundary value problem. Each sub-problem (3.4) can be solved independently and in parallel, if multiple shooting/collocation is applied. By varying the control variable $u(t)$ in each sub-interval the solution of (3.4) can be influenced in order to fulfill the overall continuity constraints. In the current approach it is assumed that $u(t) = u_i$ is constant for each subinterval $[t_i, t_{i+1}]$.

4 Multiple Shooting or Collocation

Different numerical methods are available to solve equation (3.4). The first approach presented within this paper is the reformulation of (3.4) to an ordinary differential equation

$$\dot{x}_i(t) = f(x_i(t), u_i, t) \tag{4.1}$$

with the initial condition $x_i(t_i) = h_i$.

In order to solve equation (4.1) an appropriate (e.g. explicit/implicit) integration algorithm can be applied that is already available in OpenModelica. A schematic view of the algorithmic dependencies is presented in **Figure 3**.

Alternatively, equation (3.4) or (4.1) can locally be solved using collocation methods, which also can be interpreted as numerical treatment of integration. Detailed descriptions of the multiple shooting algorithm using local collocation can be found in [7]. The solution process for equation (3.4) in each subinterval can be performed in parallel. The necessary calculation time depends certainly on the chosen integration method. In case of an explicit integration algorithm, e.g. Runge-Kutta based, more intermediate integration steps might be necessary for certain accuracy than using an implicit integration method, e.g. local collocation methods. On the other hand, explicit integration methods just perform at each intermediate step an evaluation of the model equations, whereas implicit methods in general need to solve a system of non-linear equations, which might also be time consuming. Nevertheless, when the underlying system of ordinary differential equations is stiff, implicit methods need to be applied.

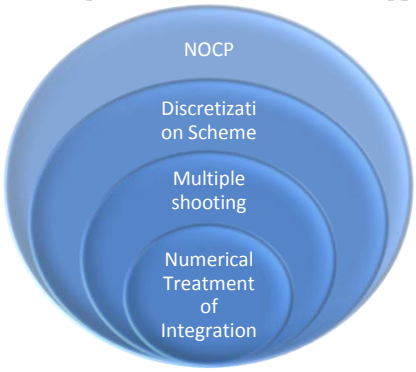

Figure 3. Schematic view of the algorithmic dependencies.

Although, equation (3.4) can be solved in parallel a lot of time is used for finding exact solutions to a locally defined problem, which might not be relevant for the over-all problem stated by the (NOCP) formulation (2.1)-(2.5). Therefore, the solution process for the NOCP still needs a lot of computation time. The next section describes methods to overcome this deficiency by adding the locally derived residual equations (based on locally applied collocation methods) to the over-all NOCP formulation.

5 Total Collocation

Applying collocation methods for solving equation (3.4) locally leads in general to a system of non-linear equations for each sub-interval. The solution process of these equations might be time consuming and with respect to the NOCP not efficient. If the corresponding non-linear equations are added to the NOCP formulation and corresponding optimization algorithms have access to the intermediate points used by the local collocation method a more efficient solution process can be formulated [4]. This section presents two different collocation methods.

Based on the common Lagrangian polynomial $p_j(z)$ for interpolation purposes, following abbreviations are introduced for $j = 0, \cdots, m$ and $k = 1, \cdots, m$:

$$p_{k,j} \coloneqq p_j(z_k) \coloneqq \prod_{\substack{l=0 \\ l \neq j}}^{m} \frac{z_k - z_l}{z_j - z_l},$$

$$\partial p_{k,j} \coloneqq \frac{\partial p_j}{\partial z}(z_k) \coloneqq \sum_{\substack{l1=0 \\ l1 \neq j}}^{m} \frac{1}{z_{l1} - z_j} \cdot \prod_{\substack{l2=0 \\ l2 \neq j \\ l2 \neq l1}}^{m} \frac{z_k - z_{l2}}{z_j - z_{l2}} \quad \text{and}$$

$$\int p_{k,j} \coloneqq \int_0^{z_k} p_j(z)\, dz$$

where $z_0, \cdots, z_m$ are the supporting points within the reference interval [0,1]. Further abbreviations are defined by $t_{j,i} = t_i + l \cdot z_j$, $x_{m,0} \coloneqq h_0, x_{j,i} \coloneqq x_i(t_{j,i})$, and $f_{j,i} \coloneqq f(x_{j,i}, u_i, t_{j,i})$.

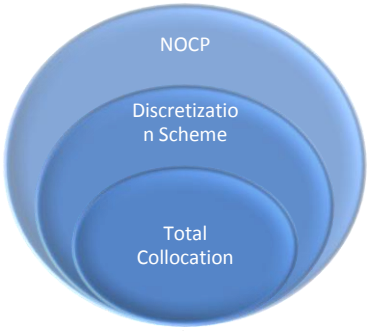

Figure 4. Schematic view of the algorithmic dependencies.

The first variant is dealing with the approximation of the states which leads to the following formulas:

$$\begin{aligned} x_{k,i} &= p_{k,0} \cdot x_{m,i-1} + \sum_{j=1}^{m} p_{k,j} \cdot x_{j,i} \\ l \cdot f_{k,i} \approx \phi_{k,i} &\coloneqq \partial p_{k,0} \cdot x_{m,i-1} + \sum_{j=1}^{m} \partial p_{k,j} \cdot x_{j,i} \end{aligned} \tag{5.1}$$

In case of $m = 1$ this approach reduces to the implicit Euler formula with approximation order 1.

The second variant is dealing with the approximation of the derivatives of the states and leads to the formulas:

$$\begin{aligned} x_{k,i} \approx \psi_{k,i} &\coloneqq x_{m,i-1} + l \cdot \sum_{j=0}^{m} \int p_{k,j} \cdot f_{j,i} \\ f_{k,i} &= \sum_{j=0}^{m} p_{k,j} \cdot f_{j,i} \end{aligned} \tag{5.2}$$

In case of $m = 1$ this approach reduces to an implicit Runge-Kutta formula (trapezoidal rule) with approximation order 2.

The discretized NOCP using total collocation and corresponding Gaussian quadrature formula for the integral part of the goal function is finally described by:

$$\begin{aligned} &\min_{u(t)} J(x(t), u(t), t) = E(x_{m,n}) + \\ &l \cdot \sum_{i=1}^{n} \sum_{j=0}^{m} \omega_j \cdot L(x_{j,i}, u_i, t_{j,i}) \end{aligned} \tag{5.3}$$

subject to

$$\begin{aligned} s(x_{k,i}, u_i, t_{k,i}) &= 0 \\ u(t_{k,i}) &= u_i \\ g(x_{m,i-1}, u_i, t_{k,i}) &\geq 0 \\ r(x_{m,n}) &= 0 \end{aligned} \tag{5.4}$$

for $i = 1, \cdots, n$, $k = 1, \cdots, m$. For variant 1 the supporting points $z_1, \cdots, z_m$, and weights $\omega_1, \cdots, \omega_m$ are given based on Radau formulas. $z_0 = 0$, $\omega_0 = 0$. $s(x_{k,i}, u_i, t_{k,i}) = l \cdot f_{k,i} - \phi_{k,i}$ are the additional residual equations from (5.1). For variant 1 the supporting points $z_0, \cdots, z_m$, and weights $\omega_0, \cdots, \omega_m$ are given based on Lobatto formulas. $s(x_{k,i}, u_i, t_{k,i}) = x_{k,i} - \psi_{k,i}$ are the additional residual equations from (5.2).

6 Modelica Applications

To investigate the performance of the proposed optimization algorithm, industrial relevant optimal control problems are solved and corresponding results are presented in this section.

6.1 Batch Reactor

We begin by considering a simple model from the chemical reactor described in [7] to maximize the yield of $x_2(t)$ by manipulation the reaction temperature $u(t)$, with the following problem formulation:

$$\min_{u(t)} J(x(t), u(t), t) = -x_2(1) \tag{6.1}$$

subject to

$$\begin{aligned} \dot{x}_1(t) &= -\left(u(t) + \frac{u^2(t)}{2}\right) \cdot x_1(t) \\ \dot{x}_2(t) &= u(t) \cdot x_1(t) \end{aligned} \tag{6.2}$$

$$\begin{pmatrix} x_1(t) \\ 1 - x_1(t) \\ x_2(t) \\ 1 - x_2(t) \\ u(t) \\ 5 - u(t) \end{pmatrix} \geq 0 \tag{6.3}$$

$$x(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{6.4}$$

where $x(t) = (x_1(t), x_2(t))^T$ and $t \in [0,1]$.

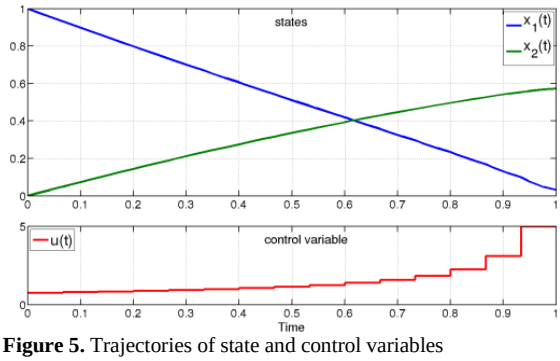

Figure 5. Trajectories of state and control variables

6.2 Optimal control of Diesel-Electric powertrain

The Diesel-electric model based on [10] is presented in Appendix A. This concept is modeled according to a nonlinear mean value engine model (MVEM) containing four states and three control inputs while the generator model is simplified by considering constant efficiency and maximum power over the entire speed range.

In a Diesel-electric powertrain the operating point of the Diesel engine can be freely chosen which would potentially decrease fuel consumption. Moreover, the electric machine has better torque characteristics. These are the main reasons making the Diesel-electric powertrain concept interesting for further studies.

To investigate the fuel optimal transients of the powertrain from idling condition to a certain power level while the accelerator pedal position is interpreted as a power level request, the following optimal control problem is solved:

$$\text{states } x = \begin{pmatrix} \omega_{ice} \\ p_{im} \\ p_{em} \\ \omega_{tc} \end{pmatrix}, \text{ controls } u = \begin{pmatrix} u_f \\ u_{wg} \\ P_{gen} \end{pmatrix}$$

$$\min \int_0^T \dot{m}_f \, dt$$

subject to

$$\begin{aligned}
\dot{x}_1 &= f_2(x_2, x_3, u_1, u_3) \\
\dot{x}_2 &= f_3(x_1, x_2, x_4) \\
\dot{x}_3 &= f_4(x_1, x_2, x_3, u_1, u_2) \\
\dot{x}_4 &= f_5(x_2, x_3, x_4, u_2) \\
0 &= f_6(x_2, x_4) - f_7(x_1, x_2) \\
0 &= f_7(x_1, x_2) + f_8(x_1, u_1) - f_9(x_3) - f_{10}(x_3, u_3) \\
0 &= \frac{f_{11}(x_3) - f_{12}(x_1)}{f_{13}(x_4)} - f_{14}(x_4)
\end{aligned}$$

$$\begin{aligned}
54\,rps &\leq x_1 \leq 220\,rps \\
0.8\,P_{\text{amb}} &\leq x_2 \leq 2P_{\text{amb}} \\
P_{\text{amb}} &\leq x_3 \leq 3P_{\text{amb}} \\
300rps &\leq x_4 \leq 10000\,rps \\
0 &\leq u_1, u_2 \leq 1
\end{aligned}$$

and boundary conditions are:

$$\text{at } t = 0, \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \text{ idle operating values,}$$

$$\text{at } t = T, \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{pmatrix} = 0, \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \text{desired values,}$$

and $u_3 = P_{\text{required}}$.

The constraints are originated from components' limitations and the functions f_{i} are described in the appendix [10].

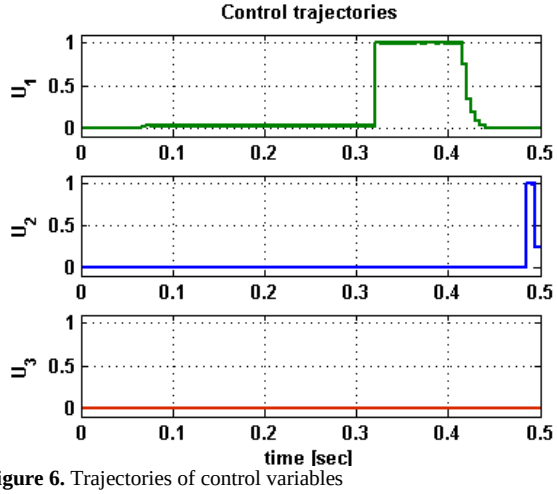

Figure 6. Trajectories of control variables

In this work, we try to find the fuel optimal control and state trajectories in a certain time interval [0,0.5]. For simplicity, only diesel operating condition is assumed which means ($u_3 = P_{gen} = 0$).

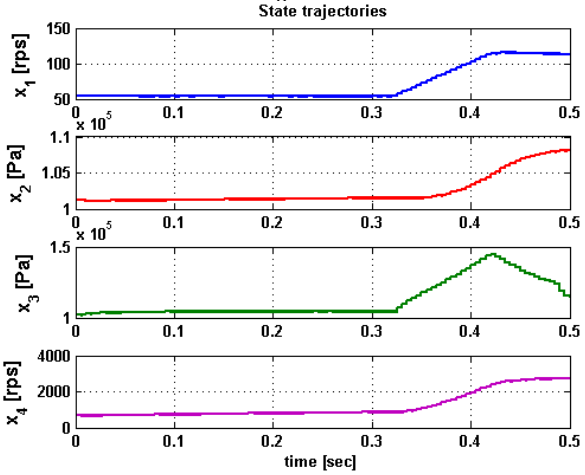

Figure 7. Trajectories of state variables

The dynamic system is solved after it is discretized into subintervals. **Figure 6** and **Figure 7** show the ob-

tained control and state trajectories. As it is expected, the fuel optimal results happen when engine is accelerated only near the end of the time interval ($t \approx 0.32\ s$) to meet the end constraints while minimizing the fuel consumption.

In section 7 it is shown how the parallel execution increases the performance of the optimization process.

7 Parallel Execution and Performance Measurements

We have performed measurements for the different algorithms (multiple shooting/collocation and total collocation with variant 1 and 2) applied to the above described applications. The C/C++ source code has been compiled by gcc version 4.6.3 (GCC) with OpenMP support. The measurements are done on an Intel Core i7 CPU 870 with 8 cores @ 2.93 GH (4 real cores and 4 virtual cores).

The corresponding optimization problem is solved by the interior point optimizer Ipopt [16]. **Figure 8** shows the different functions and derivative information that need to be provided to Ipopt for the solution process. In the current implementation the Hessian matrix of the corresponding Lagrangian formulation is calculated numerically by Ipopt. The other information (see **Figure 8**) is provided numerically by external routines. When calculating the Jacobian and Hessian matrices the treatment of the sparsity patterns, is important for the performance of the multiple shooting and total collocation methods [9]. This has been realized for the Jacobian matrix calculation.

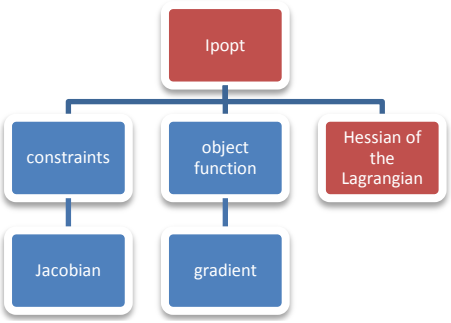

Figure 8. Schematic view of the required components of Ipopt

The multiple shooting algorithm uses an explicit Runge-Kutta formula of order 3 as well as 3 steps within each interval. The multiple collocation method uses 3 intermediate interval points based on Radau formulas. The total collocation uses variant dependent intermediate interval points as described in section 5. The tests have been performed using 128 intervals when dealing with sparse matrix representation. The user defined functions (see blue boxes of **Figure 8**) have been parallelized.

7.1 Batch Reactor

The speedups obtained and the computation times for the batch reactor are shown in **Table 1** and **Figure 9**.

	multiple shooting		multiple collocation	
threads	Ipopt	jac_g	Ipopt	jac_g
1	1,5742s	28,93ms	18,47s	343,3ms
2	1,0164s	16,77ms	10,25s	188,3ms
4	0,6691s	9,37ms	5,825s	104,7ms
8	0,6539s	8,52ms	5,055s	89,57ms

Table 1. Computation times for the Jacobian of the constraints and the over-all optimization using multiple shooting/collocation method for the batch reactor

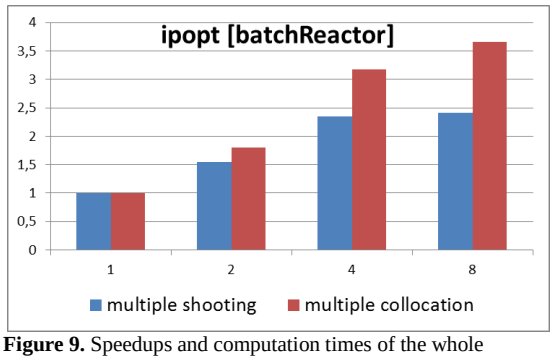

Figure 9. Speedups and computation times of the whole optimization process

Table 1 shows that multiple collocation is much more expensive than the multiple shooting. Reason for this is the computational time needed to solve non-linear systems coming from the implicit discretization. Therefore, by parallelizing the user defined functions a better speedup (**Figure 9**) for the whole optimization can be performed for the multiple shooting method, whereas the speedup for the user defined function (e.g. **Figure 10**) is comparable.

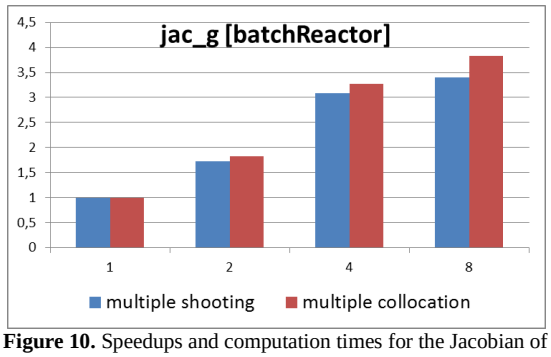

Figure 10. Speedups and computation times for the Jacobian of the constraints

7.2 Diesel Model

The solution process for the diesel model using multiple shooting and multiple collocation is quite time consuming (see **Table 2** and **Table 3**). Especially, the multiple collocation algorithm was only performed with 32 intervals in order to reduce execution time to an ac-

ceptable level. Although, parallelization of the user defined function leads to a great speed up, the overall performance of the multiple shooting or collocation method is still poor. The total collocation variants are superior with respect to the over-all performance as can be seen in **Table 3**.

	multiple shooting		multiple collocation	
threads	Ipopt	jac_g	Ipopt	jac_g
1	1518,4s	1,8196s	368,07s	2,6007s
2	917,17s	0,9671s	196,04s	1,3832s
4	608,29s	0,5286s	108,33s	0,7625s
8	508,71s	0,3861s	87,027s	0,6110s

Table 2. Computation times for the Jacobian of the constraints and the over-all optimization using multiple shooting/collocation method for the diesel model

	total collocation 1		total collocation 2	
threads	Ipopt	jac_g	Ipopt	jac_g
1	15,40s	8,215ms	14,07s	9,947ms
2	11,49s	4,356ms	10,10s	5,281ms
4	10,19s	2,553ms	8,342s	2,987ms
8	9,452s	1,713ms	7,897s	1,965ms

Table 3. Computation times for the Jacobian of the constraints and the over-all optimization using total collocation method for the diesel model

The speed-up regarding the user-defined function is comparable to the multiple shooting or collocation methods (see **Figure 12**). The speed-up of the whole optimization process is not optimal due to the serial computation and dense treatment of the Hessian matrix calculated internally by Ipopt (see **Figure 11**).

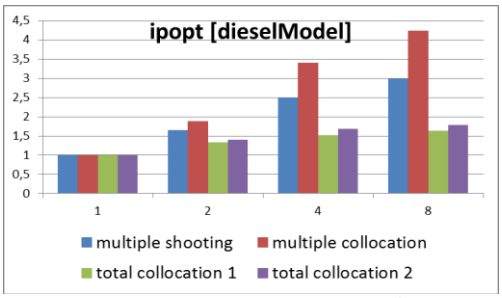

Figure 11. Speedups and computation times of the whole optimization process

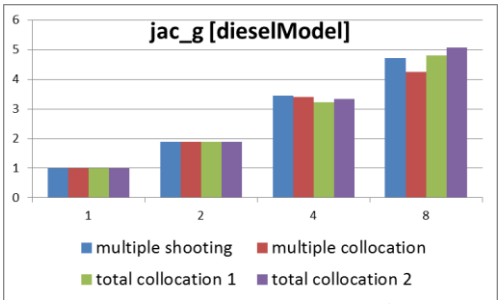

Figure 12. Speedups and computation times for the Jacobian of the constraints

8 Integration with OpenModelica

Support for specifying optimization goal functions and constraints together with Modelica models has now been implemented in OpenModelica. Such integrated models can now be exported via XML to tools such as CasADi [12] which can act as a frontend to ACADO [13].

In the current OpenModelica prototype all aspects of the tool chain are not yet completely implemented. For example, we are currently using numerically derived Gradients, Jacobians and Hessians since the automatic differentiation machinery in OpenModelica has not yet been extended to operate on the optimization problem goal function.

However, the prototype is complete enough to do the measurements of the included model applications on a parallel platform to obtain the speedup curves for parallel execution on 1-8 cores.

The OpenModelica compiler has been extended to export Modelica Models to XML based on an extended version of the FMI XML schema from [14]. The XML export, in addition to the standard Modelica syntax, supports the Optimica extensions from Jmodelica [15]. Theses extensions allow users to formulate dynamic optimization problems to be solved by a numerical algorithm. The extensions include several constructs including a new specialized class optimization, a constraint section, etc. See the batch reactor example below as well as the Optimica manual for complete information.

```
optimization BatchReactor
             (objective = -x2(finalTime),
              startTime = 0, finalTime =1)
  Real x1(start=1,fixed=true,min=0,max=1);
  Real x2(start=0,fixed=true,min=0,max=1);
  input Real u(free=true, min=0, max=5);
equation
  der(x1) = -(u+u^2/2)*x1;
  der(x2) = u*x1;
end BatchReactor;
```

The XML generated for flattened Optimica Models can be imported into other non-Modelica Optimization tools like ACADO.

Currently the OpenModelica compiler does not yet use the optimization problem formulation internally as input to automatic differentiation. The Modelica plus Optimica model description is flattened, some common compilation phases are applied e.g. syntax, semantics and type checking, simplification, constant evaluation etc. and then the complete flat model is exported to XML.

9 Conclusions

In this paper parallelized implementations of several different algorithms for solving NOCP have been presented. The well-known multiple shooting or collocation as well as total collocation methods are derived using a general discretization scheme. Total collocation methods have proofed at least in the current implementation and for the tested applications to be superior to the other algorithms.

The corresponding discretized optimization problem has been solved by the interior optimizer Ipopt. Further speedup of the optimization process for all described algorithms have been achieved by parallelizing the calculation of model specific parts (e.g. constraints, Jacobians, etc.). So far the evaluation of derivatives have been done numerically. This will be further improved using the already available symbolic differentiation capabilities of OpenModelica [10]. Finally, this work will be continued by applying the proposed algorithms on more industrial relevant applications together with a thorough testing on advanced parallel hardware architectures.

10 Acknowledgements

This work has been partially supported by Serc, by SSF in the EDOp project and by Vinnova as well as the German Ministry BMBF (BMBF Förderkennzeichen: 01IS09029C) in the ITEA2 OPENPROD project. The Open Source Modelica Consortium supports the OpenModelica work.

References

[1] Open Source Modelica Consortium. *OpenModelica System Documentation Version 1.8.1*, April 2012. http://www.openmodelica.org

[2] Modelica Association. *The Modelica Language Specification Version 3.2*, March 24th 2010. http://www.modelica.org. Modelica Association. *Modelica Standard Library 3.1*. Aug. 2009. http://www.modelica.org.

[3] Jasem Tamimi, Pu Li. A combined approach to nonlinear model predictive control of fast systems. Journal of Process Control, 20, pp 1092–1102, 2010.

[4] Biegler, Lorenz T. 2010. Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes. s.l. : Society for Industrial Mathematics, 2010.

[5] Munz, Claus-Dieter and Westermann, Thomas. 2009. Numerische Behandlung gewöhlicher und partieller Differenzialgleichungen. Berlin Heideberg : Springer Verlag, 2009

[6] Heuser, Harro. 2006. Gewöhnliche Differentialgleichungen. Wiesbaden : Teubner Verlag, 2006.

[7] Tamimi, Jasem. 2011. Development of Efficient Algorithms for Model Predictive Control of Fast Systems. Düsseldorf: VDI Verlag, 2011.

[8] Friesz, Terry L. 2007. Dynamic Optimization and Differential Games. US: Springer US, 2007.

[9] Folkmar, Bornemann und Deuflhard, Peter. 2008. Numerische Mathematik: Numerische Mathematik 2: Gewöhnliche Differentialgleichungen: Bd II: [Band] 2. s.l. : Gruyter, 2008.

[10] Martin Sivertsson and Lars Eriksson, Time and Fuel Optimal Power Response of a Diesel-Electric Powertrain. ECOSM'12 – IFAC Workshop on Engine and Powertrain Control, Simulation and Modeling, Paris, France, 2012.

[11] Braun, Willi, Ochel Lennart and Bachmann Bernhard. Symbolically Derived Jacobians Using Automatic Differentiation - Enhancement of the OpenModelica Compiler, Modelica Conference 2011

[12] Joel Andersson; Johan Åkesson; Moritz Diehl, CasADi - A symbolic package for automatic differentiation and optimal control, Proc. 6th International Conference on Automatic Differentiation, 2012.

[13] Houska, B., Ferreau, H.J., and Diehl, M. (2011). ACADO toolkit - an open source framework for automatic control and dynamic optimization. Optimal Control Applications & Methods, 32(3), 298-312.

[14] Functional Mock-up Interface: http://www.functional-mockup-interface.org/index.html

[15] Johan Åkesson. Optimica—An Extension of Modelica Supporting Dynamic Optimization. In 6th International Modelica Conference 2008. Modelica. Association, March 2008

[16] Interior Point OPTimizer (Ipopt) https://projects.coin-or.org/Ipopt

11 Appendix A

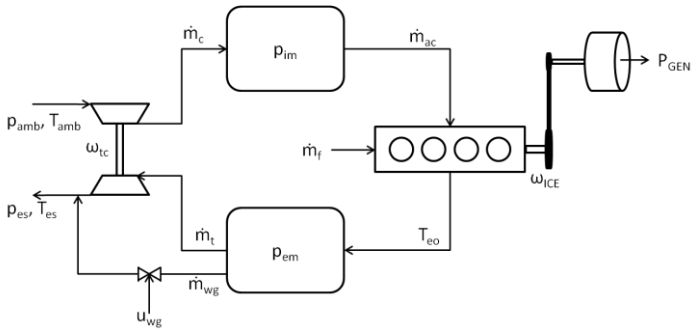

Figure 13. Diesel Engine Model

Powertrain model

$$\dot{\omega}_{ice} = \frac{P_{ice} - P_{gen}}{\omega_{ice}\, J_{genset}}$$

Intake System

- Compressor

$$\Pi_c = \frac{p_{im}}{p_{amb}}, \qquad \Pi_{c,max} = \left(\frac{{\omega_{tc}}^2 {R_c}^2 \psi_{max}}{2 c_p T_{amb}} + 1\right)^{\frac{\gamma_a}{\gamma_a - 1}}, \quad \dot{m}_{c,corr} = \dot{m}_{c,corr,max}\sqrt{1 - \left(\frac{\Pi_c}{\Pi_{c,max}}\right)^2},$$

$$\dot{m}_c = \frac{\dot{m}_{c,corr}\, p_{amb}/p_{ref}}{\sqrt{T_{amb}/T_{ref}}}, P_c = \frac{\dot{m}_c c_{pa} T_{amb}\left(\Pi_c^{\frac{\gamma_a}{\gamma_a - 1}} - 1\right)}{\eta_c}, \quad \eta_c = c_c$$

- Intake manifold

$$\dot{p}_{im} = \frac{R_a T_{im}}{V_{is}}(\dot{m}_c - \dot{m}_{ci}), \quad T_{im} = T_{cool}$$

Cylinder

- Gas Flow

$$\dot{m}_{ci} = \frac{\eta_{vol} p_{im} \omega_{ice} V_D}{4\pi R_a T_{im}}, \quad \eta_{vol} = c_{vol}, \quad \dot{m}_f = \frac{10^{-6}}{4\pi} u_f \omega_{ice} \eta_{cyl}, \quad \lambda = \frac{\dot{m}_{ci}}{\dot{m}_f} \frac{1}{(A/F)_s}$$

- Torque

$$T_{ice} = T_{ig} - T_{fric} - T_{pump}, T_{pump} = \frac{V_D}{4\pi}(p_{em} - p_{im}), T_{ig} = \frac{u_f 10^{-6} n_{cyl} q_{HV} \eta_{ig}}{4\pi}, \qquad \eta_{ig} = \eta_{ig,ch}\left(1 - \frac{1}{r_c^{\gamma_{cyl}-1}}\right)$$

$$T_{fric} = \frac{V_D}{4\pi} 10^5 \left(c_{fr1} \left(\frac{\omega_{ice} \frac{60}{2\pi}}{1000} \right)^2 + c_{fr2} \left(\frac{\omega_{ice} \frac{60}{2\pi}}{1000} \right)^2 + c_{fr3} \right)$$

- Temperature

$$\Pi_c = \frac{p_{em}}{p_{im}}, \quad \mathrm{q}_{in} = \frac{\dot{m}_f q_{HV}}{\dot{m}_f + \dot{m}_{ac}}, \quad x_p = \frac{p_3}{p_2} = 1 + \frac{q_{in} x_{cv}}{c_{va} T_{im} r_c^{\gamma_a - 1}}$$

$$T_{eo} = \eta_{sc} \Pi_e^{\left(1 - \frac{1}{\gamma_a}\right)} r_c^{(1-\gamma_a)} x_p^{\left(\frac{1}{\gamma_a} - 1\right)} \left(\mathrm{q}_{in} \left(\frac{1 - x_{cv}}{c_{pa}} + \frac{x_{cv}}{c_{va}} \right) + T_{im} r_c^{(\gamma_a - 1)} \right)$$

Exhaust System

- Exhaust Manifold:

$$\dot{p}_{em} = \frac{R_e T_{em}}{V_{em}}\left(\dot{m}_{ci} + \dot{m}_f - \dot{m}_t - \dot{m}_{wg}\right), \quad T_{em} = T_{eo}$$

- Turbine

$$\Pi_t = \frac{p_{es}}{p_{em}}, \quad \Pi_t^* = max\left(\sqrt{\Pi_t}, \left(\frac{2}{\gamma_e - 1}\right)^{\frac{\gamma_e}{\gamma_e - 1}}\right), \quad \psi_t(\Pi_t^*) = \sqrt{\frac{2\gamma_e}{\gamma_e - 1}\left((\Pi_t^*)^{\frac{2}{\gamma_e}} - (\Pi_t^*)^{\frac{\gamma_e + 1}{\gamma_e}} \right)},$$

$$\dot{m}_t = \frac{p_{em}}{\sqrt{R_e T_{em}}} \psi_t A_{t,eff}, \quad P_t = \dot{m}_t c_{pe} T_{em} \eta_t \left(1 - \Pi_t^{\frac{\gamma_e - 1}{\gamma_e}}\right), \; \eta_t = c_t, \; J_{tc}\dot{\omega}_{tc} = \frac{P_t - P_c}{\omega_{tc}} - \omega_{fric}{\omega_{tc}}^2$$

- Wastegate

$$\Pi_{wg} = \frac{p_{es}}{p_{em}}, \quad \Pi_{wg}^{*} = max\left(\Pi_{wg}, \left(\frac{2}{\gamma_e+1}\right)^{\frac{\gamma_e}{\gamma_e-1}}\right), \quad \psi_{wg} = \sqrt{\frac{2\gamma_e}{\gamma_e-1}\left(\left(\Pi_{wg}^{*}\right)^{\frac{2}{\gamma_e}} - \left(\Pi_{wg}^{*}\right)^{\frac{\gamma_e+1}{\gamma_e}}\right)}, \quad \dot{m}_{wg} = \frac{p_{em}}{\sqrt{R_e T_{em}}}\psi_{wg} u_{wg} A_{wg,eff}$$

Model Constants

Symbol	Description	Value	Unit
p_{amb}	Ambient pressure	*1.011 e5*	*Pa*
T_{amb}	Ambient temperature	*298.46*	*K*
c_{pa}	Specific heat capacity of air, constant pressure	*1011*	*J/(kg.K)*
c_{va}	Specific heat capacity of air, constant volume	*724*	*J/(kg.K)*
γ_a	Specific heat capacity ratio of air	*1.3964*	-
R_a	Gas constant, air	*287*	*J/(kg.K)*
c_{pe}	Specific heat capacity of exhaust gas, constant pressure	*1332*	*J/(kg.K)*
γ_e	Specific heat capacity ratio of exhaust gas	*1.2734*	-
R_e	Gas constant, exhaust gas	*286*	*J/(kg.K)*
γ_{cyl}	Specific heat capacity ratio of cylinder gas	*1.35004*	-
T_{im}	Intake manifold temperature	*300.6186*	*K*
p_{es}	Pressure in exhaust system	*1.011 e5*	*Pa*
$(A/F)_s$	Stoichiometric oxygen-fuel ratio	*14.57*	-
q_{HV}	Diesel heating value	*42.9 e6*	*J/kg*

Model Parameters

Symbol	Description	Value	Unit
n_{cyl}	Number of cylinders	*6*	-
V_D	Engine displacement	*0.0127*	m^3
r_c	Compression ratio	*17.3*	-
J_{genset}	Inertia of the engine-generator	*3.5*	kgm^2
V_{is}	Volume of intake system	*0.0218*	m^3
R_c	Compressor radius	*0.04*	m
ψ_{max}	Max. compressor head parameter	*1.5927*	-
$\dot{m}_{c,corr,max}$	Max. corrected compressor mass flow	*0:5462*	-
η_c	Compressor efficiency	*0.5376*	*J/(kg.K)*
η_{vol}	Volumetric efficiency	*0.8928*	-
$\eta_{ig,ch}$	Combustion chamber efficiency	*0.6774*	-
c_{fr1}	Friction coefficient	*8.41 e-5*	-
c_{fr2}	Friction coefficient	*-5.6039 e-3*	-
c_{fr3}	Friction coefficient	*0.4758*	-
η_{sc}	Non-ideal Seliger cycle compensation	*1.054*	-
x_{cv}	Ratio of fuel burnt during constant volume	*0.4046*	-
V_{em}	Volume of exhaust manifold	*0.0199*	m^3
J_{tc}	Turbocharger inertia	*1.9662 e-4*	kgm^2
ω_{fric}	Turbocharger friction	*2.4358 e-5*	kgm^2/rad
$A_{t,eff}$	Effective turbine area	*9.8938 e-4*	m^3
η_t	Turbine efficiency	*0.7278*	-
$c_{wg,1}$	Wastegate parameter	*0.6679*	-
$c_{wg,2}$	Wastegate parameter	*5.3039*	-
$A_{wg,eff}$	Effective wastegate area	*8.8357 e-4*	m^3