A parallel algorithm for large-scale dynamic optimization

Arndt Hartwich, Wolfgang Marquardt[1], RWTH Aachen University, Aachen, Germany

Abstract

In nowadays' market conditions in the chemical industry, multi-product facilities, either multi-purpose batch processes or continuous processes operated in production campaigns, become more and more common. Hence, the operation of these processes does not only require optimal stationary but also optimal dynamic modes of operation. Dynamic optimization is considered a powerful tool to determine optimal control trajectories. However, the application of this method is bothered by the very high computational times in case of considering realistic models of complete chemical plants. Therefore, besides continuing algorithmic improvement, parallel computing strategies have to be exploited. Such parallelization strategies are presented in this contribution. Three examples of very different problem sizes are discussed. The speedup for the largest example even obtains a value close to the theoretical limit of the parallelization approach.

1 Introduction

Dynamic optimization can be considered as a powerful tool, where most applications are more or less time-critical. In particular, the applicability of real-time implementations strongly depends on the computational times until the solution to the dynamic optimization problem is obtained. Applying the *sequential approach*, the infinite dimensional optimization problem is transformed into a finite dimensional optimization problem by means of an appropriate discretization, where the computation of the corresponding gradients, also termed sensitivities, of the latter optimization problem results in the largest part of computational times. Thus, the numerical integration of the combined state and sensitivity equation system represents the largest portion of computational effort. Nowadays' developments in computer hardware are not directed towards faster CPUs, but towards parallel computers. Hence, further reductions in computational times can only be accomplished if parallel computing is accounted for. Keeping and Pantelides (1998) and Zhu and Petzold (1999) have exploited BDF-type numerical integration routines in order to allow for distributed memory parallelization. Schlegel *et al.* (2004) have extended the linear-implicit extrapolation routine LIMEX (Deuflhard and Bornemann, 1994; Deuflhard *et al.*, 1987) in order to allow for the solution of sensitivity equation systems. The parallel implementation of this sensitivity integration algorithm is described in this work. Nowak *et al.* (1998) have developed a parallel algorithm of LIMEX. As however the greatest part of the arising computational effort is due to the sensitivity equation system, the parallelization as suggested in this work focuses on the additional computational effort due to the sensitivity equation system similar to the works of Keeping and Pantelides (1998) and Zhu and Petzold (1999). As the operating system Windows Server 2008 has been chosen.

2 Numerical Solution methods of dynamic optimization

The dynamic optimization involves the determination of the optimization variables, control trajectories $\mathbf{u}(\text{t})$ and final time t_f, such that an objective function Φ has to be minimized subject to the model equations and additional path and endpoint constraints. Mathematically this leads to a non-linear optimal control problem and can be stated as

$$\min_{\mathbf{u}(t),t_f} \quad \Phi = \Phi(\mathbf{y}(t_f), \mathbf{u}(t_f)) \tag{CDYNOPT}$$

$$\begin{aligned} s.t.\ \mathbf{B}\dot{\mathbf{y}} &= \mathbf{f}(\mathbf{y}(t), \mathbf{u}(t)), && t \in [t_0, t_f], && (1a)\\ 0 &= \mathbf{y}(t_0) - \mathbf{y}_0, && && (1b)\\ 0 &\geq \mathbf{h}^y(\mathbf{y}(t), \mathbf{u}(t)), && t \in [t_0, t_f], && (1c)\\ 0 &\geq \mathbf{e}(\mathbf{y}(t_f), \mathbf{u}(t_f)), && && (1d) \end{aligned}$$

[1]Corresponding author: wolfgang.marquardt@avt.rwth-aachen.de

$$\mathbf{u}(t)^{min} \leq \mathbf{u}(t) \leq \mathbf{u}(t)^{max}, \qquad t \in [t_0, t_f]. \tag{1e}$$

This formulation will be denoted as the continuous dynamic optimization problem (CDYNOPT) in the following. $\mathbf{y}(t) \in \mathcal{R}^{n_y}$ denotes the vector of the state variables, defined on the time horizon $t \in [t_0, t_f]$. The differential-algebraic process model of index one is given by Equation 1a. The time-dependent control variables $\mathbf{u}(t) \in \mathcal{R}^{n_u}$ are degrees of freedom for the optimization problem minimizing the objective function Φ. The state variables are constrained by path constraints $\mathbf{h}^y(\cdot) \in \mathcal{R}^{n_{h^y}}$. Endpoint constraints are denoted by $\mathbf{e}(\cdot) \in \mathcal{R}^{n_e}$.

2.1 Numerical solution

Binder *et al.* (2001) and Srinivasan *et al.* (2003) have discussed in detail the specific advantages and disadvantages of alternative strategies for the solution of dynamic optimization problems. The chosen *direct method* solves the dynamic optimization problem (CDYNOPT) numerically according to the *control vector parameterization approach*, which is also termed the *sequential approach*. This solution method is well-known for its ability to solve very large-scale and stiff problems.

2.1.1 Tranformation into NLP

The dynamic optimization problem (CDYNOPT) is transcribed into a non-linear program (NLP) by time-discretizing the controls $\mathbf{u}(t)$ on the time horizon $t \in [t_0, t_f]$ and using, e.g. piecewise constant or piecewise linear approximations. The discretization and the formulation of the NLP for the piecewise constant approximation reads as $\mathbf{u}_i(t_k) = c_{\mathbf{u}_{i,k}}, k = 1, ..., N, i = 1, ..., n_u$, where N is the number of discretization intervals and n_u the number of control variables. Choosing the discretized controls and the final time $z := [c_{u_{1,1}}, ..., c_{u_{n_u,N}}, t_f]$, as the n_z optimization variables, the dynamic optimization problem can be transcribed into the NLP

$$\min_{\mathbf{z}} \Phi = \Phi_M \left(\mathbf{y}\left(\mathbf{z}, t_f\right)\right) \tag{NLP}$$

$$\text{s.t.} \quad \mathbf{0} \geq \mathbf{h}^{\mathbf{y}}(\mathbf{y}(\mathbf{z})), \tag{2a}$$

$$\mathbf{0} \geq \mathbf{e}\left(\mathbf{y}\left(\mathbf{z}\right)\right), \tag{2b}$$

$$\mathbf{z}^{min} \leq \mathbf{z} \leq \mathbf{z}^{max}. \tag{2c}$$

The approximation and reformulation for piecewise linear intervals is straightforward. The path constraints (1c) of the continuous formulation (CDYNOPT) are relaxed by a pointwise formulation on the discretized grid. The NLP is solved by employing a standard SQP algorithm (Gill *et al.*, 1998). The optimization algorithm requires repetitive function evaluations and gradients.

2.1.2 Numerical methods for the computation of first-order sensitivities

In order to obtain the gradients for the NLP, one can differentiate the DAE system (1a) with respect to each parameter z_i, which yields additional n_z sensitivity equation systems of the form

$$\mathbf{B}\dot{\mathbf{s}}_i^y = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}\mathbf{s}_i^y + \frac{\partial \mathbf{f}}{\partial z_i}, \; i = 1, ..., n_z, \tag{3}$$

where $\mathbf{s}_i = \frac{d\mathbf{y}}{dz_i}$ denotes the sensitivity of $\mathbf{y}$ with respect to the parameter z_i. The sensitivity systems (3) for each parameter z_i form DAE systems as well, which are independent of each other, but obviously depend on the solution of the state equations (1a). Since for each degree of freedom z_i a sensitivity equation system has to be solved, the computational cost of this approach is rangely proportional to the number of decision variables n_z.

2.1.3 Linearly-implicit Euler extrapolation for sensitivity analysis

This section briefly summarizes the linearly-implicit extrapolation algorithm. The reader is referred to the works of Deuflhard (1983), Deuflhard and Bornemann (1994), Deuflhard *et al.* (1987) and Ehrig *et al.* (1996) for more details on the basic numerical integration routine and to Schlegel *et al.* (2004) for more details on the extensions for simultaneous state and sensitivity integration, that computes the vector $\mathbf{Y} := [\mathbf{y}^T, \mathbf{s}_1^T, ..., \mathbf{s}_{n_z}^T]$.

The key idea of an extrapolation method is a repeated integration over the basic step H for decreasing internal stepsizes

$$h_j = H/m_j,\ j = 1, \ldots, j_{max}, \tag{4}$$

where the harmonic sequence $\{m_j\} = \{1, 2, 3, ...\}$ is utilized. A simplified algorithmic structure is depicted by Algorithm 1, where the extrapolation tableau is computed according

$$T_{j,k} = T_{j,k-1} + \frac{T_{j,k-1} - T_{j-1,k-1}}{m_j/m_{j-k+1} - 1}, \qquad k = 1, \ldots, j. \tag{5}$$

Algorithm 1 Simultaneous state and sensitivity integration (SLIMEX)

```
Compute Jacobian $\mathbf{A}_n = \frac{\partial}{\partial \mathbf{y}}(\mathbf{f}(\mathbf{y}_n, \mathbf{z}, t_n) - \mathbf{B}\dot{\mathbf{y}})$
for $j = 1, \ldots, j_{max}$ while convergence criterion not satisfied do
  $h_j = H/m_j$
  $\mathbf{LU} = \mathbf{A}_n - \frac{\mathbf{B}}{h_j}$
  for $k = 0, \ldots, j-1$ do
    $\mathbf{y}_{k+1} = \mathbf{y}_k - (\mathbf{LU})^{-1}\mathbf{f}(\mathbf{y}_k, \mathbf{z}, t_k)$
    $\Gamma = \left(\mathbf{A}(\mathbf{y}_k)\mathbf{s}_{i,k} + \frac{\partial \mathbf{f}}{\partial z_i}(\mathbf{y}_k)\right)$
    $\mathbf{s}_{i,k+1} = \mathbf{s}_{i,k} - (\mathbf{LU})^{-1}\Gamma, i = 1, \ldots, n_z$
  end for
  $\mathbf{T}_{j,1} = \mathbf{Y}_j$
  if $j > 1$ then
    compute $\mathbf{T}_{j,j}$ via (5) and check convergence
  end if
end for
$\mathbf{Y}(H) = \mathbf{T}_{j,j}$
```

As described by Schlegel *et al.* (2004), a staggered integration by Algorithm 2 has been omitted. Error control is only applied to the states, but not to the sensitivity equation system. A staggered approach first converges the states and afterwards computes the sensitivities on a converged sequence of order $\hat{j}$ and step-sizes H. Hence, eventually redundant extrapolations of the sensitivity system could be avoided by only integrating the sensitivities if convergence in the states has been accomplished. However, housekeeping of the LU decompositions, Jacobians etc. has been considered too much effort for the rather rare occasion of a step-size reduction. For reasons explained below the staggered approach is parallelized.

It should be noted that the first LU decomposition of each step is computed employing the MA28AD routine, whereas every subsequent LU decomposition is computed by the MA28BD routine (Duff, 1979), which reuses the pivot sequence of MA28AD and substantially saves computational times. In some cases, it can be computationally beneficial if even the first LU decomposition on each step is first attempted using MA28BD and afterwards using MA28AD, if the first one has failed. This is done for the third example, which will be discussed in Section 4.

3 Parallel dynamic optimization

According to Algorithm 1, the same Jacobian matrix $\mathbf{A}$ is required for the state system and for the sensitivity equation systems. The numerical integration scheme requires one Jacobian matrix per step for a pure state integration, whereas a sensitivity integration requires significantly more than one Jacobian, as the Jacobian is required for the construction of the right hand side of the sensitivity equation system. The computation of a Jacobian can

Algorithm 2 Staggered state and sensitivity integration (SLIMEX)

```
Compute Jacobian $\mathbf{A}_n = \frac{\partial}{\partial \mathbf{y}}(\mathbf{f}(\mathbf{y}_n, \mathbf{z}, t_n) - \mathbf{B}\dot{\mathbf{y}})$
for $j = 1, \ldots, j_{max}$ while convergence criterion not satisfied do
  $h_j = H/m_j$
  $\mathbf{LU} = \mathbf{A}_n - \frac{\mathbf{B}}{h_j}$
  for $k = 0, \ldots, j-1$ do
    $\mathbf{y}_{k+1} = \mathbf{y}_k - (\mathbf{LU})^{-1}\,\mathbf{f}(\mathbf{y}_k, \mathbf{z}, t_k)$
  end for
  $\mathbf{T}^y_{j,1} = \mathbf{Y}_j$
  if $j > 1$ then
    compute $\mathbf{T}^y_{j,j}$ via (5) and check convergence
  end if
  $\hat{j} = j$
end for
$\mathbf{y}(H) = \mathbf{T}^y_{j,j}$
for $j = 1, \ldots, \hat{j}$ do
  $h_j = H/m_j$
  $\mathbf{LU} = \mathbf{A}_n - \frac{\mathbf{B}}{h_j}$
  for $k = 0, \ldots, j-1$ do
    $\Gamma = \left(\mathbf{A}(\mathbf{y}_k)\mathbf{s}_{i,k} + \frac{\partial \mathbf{f}}{\partial z_i}(\mathbf{y}_k)\right)$
    $\mathbf{s}_{i,k+1} = \mathbf{s}_{i,k} - (\mathbf{LU})^{-1}\Gamma, \quad i = 1, \ldots, n_z$
  end for
  $\mathbf{T}^s_{j,1} = \mathbf{s}_{i,j}$
  if $j > 1$ then
    compute $\mathbf{T}^s_{j,j}$ via (5)
  end if
end for
$\mathbf{s}_i(H) = \mathbf{T}^s_{j,j}$
```

be very time consuming - depending on the model under consideration. Furthermore, the additional effort required for sensitivity evaluation when compared to pure state integration comprises n_z additional evaluations of the sensitivity residuals and n_z additional back substitutions with the already available LU decomposition. The error control is only applied to the states. It is important to note that the sensitivity equation system i is independent of the system $j \neq i$. The implementation is done using the Message Passing Interface (MPI). Windows Server 2008 has been chosen as the operating system.

3.1 Parallelization strategies

For a wide range of case studies also wide ranges of the number of sensitivity equation systems n_z and the size of the DAE model n_y have to be accounted for in order to obtain high efficiency for each case study at hand. It is expected, that increasing computational times will allow for more communication between the processes, whereas very small examples might even result in no speedup in a parallel setting. In the following, three different parallelization strategies are discussed, where the extent to which information is communicated, increases. These three different strategies are expected to cover the wide range of possible case studies.

3.1.1 Distributed sensitivities (DS)

The main idea for parallelization is the distributed computation of sensitivities, since the sensitivity equations are independent of each other and only dependent on the state system. All processes are assigned an equal number of sensitivity equation systems. This approach is very simple and requires communication only once per sensitivity integration, i.e. after each integration sensitivity information of each process is broadcasted to all other processes such that each process contains the complete sensitivity information. The processes will be denoted as *distributed* *sensitivity integrators* (DSI) in the following. All processes have to perform the state integration, the computation of the Jacobians and the LU decomposition.

3.1.2 Distributed sensitivities with centralized Jacobian (CJ)

The fundamental idea of this approach is to decouple the state integration and the sensitivity integration. The state integration requires one Jacobian per integration step H, i.e. $\mathbf{A}_n$, whereas the sensitivity integration requires additional evaluations of the Jacobian $\mathbf{A}(\mathbf{y}_k)$ for each iteration $k = 1, \ldots, j-1$ on the extrapolation tableau. However, the states $\mathbf{y}_k$, from which the Jacobians can be computed, are independent of the sensitivity system and can be obtained by a pure state integration. Hence, a preceding state integration can deliver the states $\mathbf{y}_k$ and the Jacobians $\mathbf{A}(\mathbf{y}_k)$ can be computed by separate processes. Processes which integrate the sensitivity equations follow the state integration with a small time delay and receive the Jacobians from separate processes that compute the Jacobians. Thus, the computational burden to obtain Jacobian matrices for the sensitivity integration can be reduced to the communication overhead. However, additional processes that compute the Jacobians and communicates the entire information, respectively, are required. One of the drawbacks of this approach is the very high frequency of communication, which has to take place after each step of the integrator. This strategy, which has been suggested similarly by Keeping and Pantelides (1998), will require four different types of processes:

- *Communicator* (CO): This process communicates the entire information.
- *State integrator* (STI): This process carries out the sole state integration and communicates the states $\mathbf{y}$, etc. to the communicator.
- *Jacobian calculator* (JC): This process receives the state vectors $\mathbf{y}$ from the Communicator (CO) and returns the computed Jacobians $\mathbf{A}(\mathbf{y})$.
- *Sensitivity integrator* (SEI): Analogously to the distributed sensitivities strategy this process performs a sensitivity integration of a subset of sensitivities and additionally receives the required Jacobians from (CO). In contrast to the distributed sensitivities strategy, the states are not included in the numerical integration, but are communicated to the SEI to allow for the computation of the LU decomposition. As in the original algorithm error control on the sensitivities has been omitted, the order j and the step-size H are sent to the sensitivity integrator from the state integrator via the communicator.

The staggered approach is parallelized and the entire communication of information has to account for the fact, that states $\mathbf{y}_k$, that are computed, have to be omitted, if a step size reduction has to be performed. The staggered approach has been considered too much effort for the serial code since step size reductions are rather rare and housekeeping of LU decompositions and Jacobians, etc. has been considered too much effort. However, the parallel code has to store the LU decompositions and Jacobians anyway and therefore the latter argument is not valid for the parallelization. Hence, the staggered approach is parallelized as it saves eventually redundant basic steps on the sensitivity equations.

3.1.3 Distributed sensitivities with centralized Jacobian and communicated LU decomposition (CJLU)

This approach follows the idea of (CJ), but the LU decompositions, which are computed by the state integrator, are additionally communicated to the sensitivity integrators via the communicator. The size of the LU decomposition is significantly larger compared to the size of the Jacobians. Hence, the communication overhead will increase significantly compared to (CJ). Furthermore, the efficient implementation of LIMEX, which reuses the same pivot sequence over one step, can result in higher computational times for (CJLU) compared to (CJ). MA28AD is employed only once to compute the LU decomposition on one step. All subsequent computations of the LU decompositions are computed by calls to MA28BD, which substantially saves computational effort. Therefore, the overhead due to communication might be equal to the potential savings.

3.2 Load balancing

In order to obtain the optimal degree of speedup, the appropriate numbers of CPUs assigned to each task has to be determined. The state integrator has to be sufficiently fast to allow for computation of the Jacobians in time, when the sensitivity integrators require the Jacobians. The theoretical limit of the reduction in computational time is the pure state integration, which follows from Amdahls law. Hence, there exists an upper limit for the total number of CPUs for the chosen approach of parallelization. Nowak *et al.* (1998) presented a parallel version of the LIMEX state integrator. Merging both approaches would result in a lower theoretical limit.

3.3 Memory issues

During one of the case studies, which will be described in more detail in Section 4, the required memory space for sensitivity integration exceeded the allowed limit of 3GB for 32Bit software on Windows. Due to the fact that parts of the code is commercial and at that point in time not available in 64Bit, the sensitivity integration had to be split into several sequential sensitivity integrations, where each integration computes a part of the entire sensitivity information. Hence, the parallelization using MPI greatly contributes to the efficiency of the software, since all sensitivities can be computed by one parallel sensitivity integration, as the sensitivity load is distributed on several processes, each of which can address up to 3GB. The assessed numerical example features a rather small number of sensitivities, which can be computed by one serial sensitivity integration in order to allow for a comparison of the serial code to the parallel one. However, for the real case study the speedup is not only larger due to the higher overall load of sensitivities and a longer time horizon, the real speedup is even higher, since the serial code would have to compute the sensitivities by repeated sensitivity integrations on a reduced load.

4 Numerical Experiments

In this section the algorithmic performance is investigated by means of three case studies. The efficiency

$$\Theta = \frac{\Psi}{n_p} \tag{6}$$

is of interest not only for very large-scale models, but also when considering small-scale problems. n_p denotes the number of CPUs and Ψ speedup

$$\Psi = \frac{t_s}{t_p}, \tag{7}$$

where t_s denotes the serial computational time and t_p the parallel one. The examples are chosen over a wide range of applications. The chosen approach for parallel computing is targeted at large problems, where the model consists of many equations and many sensitivities have to be computed. The numerical experiments have been conducted on Xeon 2.66 GHz double quad core machines employing 16 GByte of random access memory running on Windows Server 2008. Since only 8 CPUs on one node have been available and the network between the nodes has not been considered to be sufficiently fast, the numerical experiments have not yet assessed the parallelization beyond 8 CPUs.

4.1 Case studies

4.1.1 Example 1: Polymerization process (236 DAEs)

The controls are discretized by 68 parameters. The model is considered very small, which is suggested by the very low computational times for one single sensitivity integration. The applied strategy, computational times required for one sensitivity integration, speedup and efficiency are depicted in Table 1. For strategies (CJ) and (CJLU) also the number of CPUs per task are given. Since strategy (DS) only employs one type of process, all CPUs are used as distributed sensitivity integrators. The number in brackets shows the computational time for one single state integration. This number represents the theoretical limit for the parallelization. Only strategy (DS) has been successfully applied to this example, since the other approaches have resulted in higher computational times compared to the serial code because of large overhead. Even though this example is very small and features only a small number of sensitivities, a speedup in the range of two has still been achieved.

Strat.	CO	STI	JC	SEI	#CPU	Time [sec]	Ψ	Θ
serial	0	0	0	0	1	2.54(0.5)		
DS	0	0	0	0	2	1.67	1.52	0.84
DS	0	0	0	0	3	1.44	1.76	0.48
DS	0	0	0	0	4	1.51	1.68	0.38
DS	0	0	0	0	5	1.34	1.90	0.27
DS	0	0	0	0	6	1.31	1.94	0.28
DS	0	0	0	0	7	1.29	1.97	0.18
DS	0	0	0	0	8	1.26	2.02	0.16

Table 1: Example 1: Strategy, number of processors and computational times

4.1.2 Example 2: Polymerization process (2104 DAEs)

The controls are discretized by 197 parameters. For this example, strategy (CJ) turns out to be the fastest strategy (Tab. 2), whereas DS gives slightly worse results and CJLU is significantly worse compared to the other approaches. In total, a speedup of around 3 is obtained. Only one JC is considered sufficient, since the major computational effort is required for the computation of the sensitivities. Strategy DS does not even show major improvements in the total computational time if the number of employed CPUs is increased beyond six.

Strat.	CO	STI	JC	SEI	#CPU	Time [sec]	Ψ	Θ
serial	0	0	0	0	1	33.27(2.3)		
DS	0	0	0	0	2	17.68	1.88	0.94
DS	0	0	0	0	3	13.04	2.55	0.85
DS	0	0	0	0	4	12.97	2.57	0.64
DS	0	0	0	0	5	12.01	2.77	0.55
DS	0	0	0	0	6	11.57	2.88	0.48
DS	0	0	0	0	7	11.34	2.93	0.42
DS	0	0	0	0	8	11.28	2.95	0.37
CJ	1	1	1	1	4	29.19	1.14	0.28
CJ	1	1	1	2	5	15.73	2.12	0.42
CJ	1	1	1	3	6	12.53	2.66	0.44
CJ	1	1	1	4	7	11.4	2.92	0.42
CJ	1	1	1	5	8	10.74	3.10	0.39
CJLU	1	1	1	1	4	33.29	1.00	0.25
CJLU	1	1	1	2	5	20.98	1.59	0.32
CJLU	1	1	1	3	6	18.83	1.77	0.29
CJLU	1	1	1	4	7	17.95	1.85	0.26
CJLU	1	1	1	5	8	17.55	1.90	0.24

Table 2: Example 2: Strategy, number of processors and computational times

4.1.3 Example 3: Large-scale intermediate chemicals process (13956 DAEs)

The controls are discretized by 75 parameters. This example is considered the most important example for the studied parallelization strategies. One of the most interesting aspects of this case study is, that all physical properties are evaluated by external functions, i.e. so-called physical property packages. Hence, they have a strong impact on the computational time for the Jacobians, whereas however they result in a rather small Jacobian, since the corresponding equations are solved within the property packages and do not show up in the model. These physical properties require many floating point operations, whereas the corresponding LU decomposition is computed comparably fast, since the LU decomposition is only dependent on the size and structure of the Jacobian.

Considering the discussion in Section 3.3 the speedup would be larger for a higher number of parameters. However in this case, the serial performance could not be determined due to memory issues. Since several sequential sensitivity integrations with a reduced load would have to be performed, the real speedup of the parallel code is significantly larger, such that even the efficiency Θ can exceed 100%.

In contrast to the other examples, the parallelization performance almost reaches its theoretical limit, which is the state integration without sensitivities. Strategy CJ with 4 CPUs assigned to JC (489 sec) almost reaches the limit (435 sec). The communication overhead that occurred for this example is comparably small, since the general computational times are large enough. Furthermore, the frequency of communication decreases.

Strat.	CO	STI	JC	SEI	#CPU	Time [sec]	Ψ	Θ
serial	0	0	0	0	1	2080 (435)		
DS	0	0	0	0	2	1997	1.04	0.52
DS	0	0	0	0	3	1914	1.09	0.36
DS	0	0	0	0	4	1866	1.11	0.28
DS	0	0	0	0	5	1832	1.14	0.23
DS	0	0	0	0	6	1755	1.19	0.20
DS	0	0	0	0	7	1738	1.20	0.17
DS	0	0	0	0	8	1701	1.22	0.15
CJ	1	1	1	1	4	1375	1.51	0.38
CJ	1	1	2	2	6	698	2.98	0.50
CJ	1	1	3	3	8	543	3.83	0.48
CJ	1	1	2	4	8	735	2.83	0.35
CJ	1	1	4	2	8	489	4.25	0.53
CJLU	1	1	3	3	8	611	3.40	0.43
CJLU	1	1	2	2	8	776	2.68	0.34
CJLU	1	1	1	1	8	1434.58	1.45	0.18
CJLU	1	1	2	4	8	838.54	2.48	0.31
CJLU	1	1	4	2	8	593.65	3.50	0.44

Table 3: Example 3: Strategy, number of processors and computational times

4.2 Discussion of results

The above examples show that the chosen approaches work very well for large-scale problems and still quite efficient for small and medium-scale problems. Strategy (CJ) performs best for the larger two examples and (DS) for the smallest one, which was expected to a certain extent. The difference between (CJ) and (CJLU) is rather small for the largest example. However, for this example (CJ) works better, since the LU decomposition is rather small compared to the computational effort for each Jacobian.

5 Conclusion

In this paper, the parallelization of the combined state and sensitivity integration of a dynamic optimization approach has been suggested. The one-step extrapolation code of Schlegel *et al.* (2004), that requires by far the largest computational effort in the chosen method of the *sequential approach*, has been parallelized by means of an exploitation of the algorithmic structure. Three numerical examples were examined, which assess the three implemented approaches to parallelize the algorithms. The simple distributed computation of the sensitivities turns out to be the best approach for very small examples, whereas the additional centralized computation of the Jacobians gives considerable extra speedup, even though the overhead increases significantly. For the largest example, the theoretical limit has almost been reached. The additional communication of the LU decomposition turns out to be not efficient for the chosen examples, since the communication overhead of the enormous amount of data is larger than the computational times for the LU decomposition at all. For future work, different implementations for the computation of the LU decomposition will be assessed, but also different possibilities of the parallelization of the LU decomposition. Furthermore, OpenMP can be employed in order to speed up each process of the illustrated algorithm. Since only nodes with 8 CPUs have been employed, the usage of OpenMP has been omitted for this work, since already 4 tasks are required to be divided onto the available CPUs.

References

Binder, T., L. Blank, H. G. Bock, R. Bulirsch, W. Dahmen, M. Diehl, T. Kronseder, W. Marquardt, J. P. Schlöder and O. von Stryk (2001). Introduction to model based optimization of chemical processes on moving horizons. In: *Online Optimization of Large Scale Systems* (M. Grötschel, S. O. Krumke and J. Rambau, Eds.). Springer-Verlag Berlin Heidelberg. pp. 297–339.

Deuflhard, P. (1983). Order and stepsize control in extrapolation methods. *Numerische Mathematik* (41), 399–422.

Deuflhard, P. and F. Bornemann (1994). *Numerische Mathematik II.* De Gruyter. Berlin.

Deuflhard, P., E. Hairer and J. Zugck (1987). One-step and extrapolation methods for differential-algebraic systems. *Numerische Mathematik* (51), 501–516.

Duff, I.S. (1979). Ma28 - a set of fortran subroutines for sparse unsymmetric linear equations. Technical Report AERE-R8730. AERE Harwell.

Ehrig, R., U. Nowak and P. Deuflhard (1996). Highly scalable parallel linearly-implicit extrapolation algorithms. Technical Report TR 96–11. Konrad-Zuse-Zentrum für Informationstechnik. Berlin.

Gill, P.E., W. Murray and M.A. Saunders (1998). SNOPT: An SQP algorithm for large-scale constrained optimization. Technical report. Stanford University. Stanford, USA.

Keeping, B.R. and C.C. Pantelides (1998). A distributed memory parallel algorithm for the efficient computation of sensitivities of differential-algebraic systems. *Math. comp. Sim.* **44**, 545–558.

Nowak, U., R. Ehrig and L. Overdieck (1998). Parallel extrapolation methods and their application in chemical engineering. *High-Performance computing and networking lecture notes in computer science* **1401**, 419–428.

Schlegel, M., W. Marquardt, R. Ehrig and U. Nowak (2004). Sensitivity analysis of linearly-implicit differential-algebraic systems by one-step extrapolation. *Appl. Num. Math.* **48**, 83–102.

Srinivasan, B., S. Palanki and D. Bonvin (2003). Dynamic optimization of batch processes I. Characterization of the nominal solution. *Comp. Chem. Eng.* **27**, 1–26.

Zhu, W.J. and L. Petzold (1999). Parallel sensitivity analysis for daes with many parameters. *Concurrency-practice and experience* **11**, 571–585.