An asynchronous Oneshot method with Load Balancing

Torsten Bosse

Argonne National Laboratory, IL
Design Optimization

Problem

\[ \min_{(u,y) \in \mathbb{R}^{m+n}} f(u, y) \text{ subject to } c(u, y) = 0, \]

for sufficiently smooth functions \( f : \mathbb{R}^{m\times n} \to \mathbb{R} \) and \( c : \mathbb{R}^{m\times n} \to \mathbb{R}^n \)

Assumption \( \exists \) a slowly convergent **fixed-point solver** \( G : \mathbb{R}^{m\times n} \to \mathbb{R}^n \) for \( c \):

\[ y^{k+1} = G(u, y^k) \quad [c(u, y) = 0 \iff y = G(u, y)] \]

with a global **contraction rate** \( \|G_y(u, y)\| \leq \rho < 1 \), e.g. \( \rho \approx 0.999 \)

Lagrange function with adjoint variable \( \bar{y} \in \mathbb{R}^n \)

\[ L(u, y, \bar{y}) \equiv f(u, y) + \bar{y}^\top [G(u, y) - y] \]

**Find** \((u^*, y^*, \bar{y}^*)\) satisfying first order necessary / **KKT conditions**

\[
\begin{align*}
0 &= L_{\bar{y}}(u^*, y^*, \bar{y}^*) \quad \equiv \quad G(u^*, y^*) - y^* \quad \text{(primal)} \\
0 &= L_y(u^*, y^*, \bar{y}^*) \quad \equiv \quad f_y(u^*, y^*) + [G_y(u^*, y) - I]^\top \bar{y}^* \quad \text{(adjoint)} \\
0 &= L_u(u^*, y^*, \bar{y}^*) \quad \equiv \quad f_u(u^*, y^*) + G_u^\top(u^*, y^*) \bar{y}^* \quad \text{(design)}
\end{align*}
\]
Fixed-Point Solver for the State Equation

Assume  Optimal control $u^*$ and $y^*$ is known.

Goal   Find state $y^*$ satisfying $c(u^*, y^*) = 0$.

Solution Use slowly convergent state fixed-point solver $G : U \times Y \rightarrow Y$, i.e.,
start with an initial approximation $y^0 \in Y$ and iterate

$$y^{k+1} = G(u^*, y^k) \quad \text{for } k = 0, 1, 2, \ldots$$

such that $y^* = \lim_{k \to \infty} y^k$ solves $y^* = G(u^*, y^*) \iff c(u^*, y^*) = 0$

due to global contraction rate $\|G_y(u, y)\| \leq \rho < 1$ and Banach FPT.

Notation

$$\cdots \rightarrow \text{(state)} \rightarrow \text{(state)} \rightarrow \text{(state)} \rightarrow \cdots$$

Interpretation Fixed-point iteration is some solver for a PDE/numerical model that

describes a physical process represented by the equality $c(u, y) = 0$
Assume Optimal control $u^*$ is known.

Goal Find state $y^*$ satisfying $c(u^*, y^*) = 0$
and adjoint solution $\bar{y}^*$ with

$$0 = f_y(u^*, y^*) + [G_y(u^*, y^*) - I]^\top \bar{y}^*$$

BUT only allowed to use Jacobi-vector, vector-Jacobi products -
evaluation/inversion/factorization of $G_y$ might be to costly!

Idea For an initial approximation $\bar{y}^0$ and given $y$ the adjoint iteration

$$\bar{y}^{k+1} = f_y(u^*, y) + G_y^\top(u^*, y) \bar{y}^k$$

is also a fixed point iteration due to $\|G_y(u, y)\| \leq \rho < 1$.

Updating scheme according to Christianson

$$\cdots \to \text{(state)} \to \text{(state)} \to \cdots \to \text{(adjoint)} \to \text{(adjoint)} \to \ldots \text{ (B.C.)}$$
Problem  The B.C. updating is **sequential**, i.e., converge primal iteration

\[ y^{k+1} = G(u^*, y^k) \quad \text{for } k = 0, 1, 2, \ldots \]

to a solution \( y^* = G(u^*, y^*) \) (or an approximation) before iterating

\[ \bar{y}^{k+1} = f_y(u^*, y^*) + G_y^\top(u^*, y^*)\bar{y}^k \quad \text{for } k = 0, 1, 2, \ldots \]

from initial \( \bar{y}^0 \) until \( \bar{y}^* = \lim_{k \to \infty} \bar{y}^k \) with \( L_y(u^*, y^*, \bar{y}^*) = 0 \).

Alternative approach (Griewank): Update \((y^k, \bar{y}^k)\) in **parallel**, i.e.,

\[
\begin{align*}
  y^{k+1} &= G(u^*, y^k) \\
  \bar{y}^{k+1} &= f_y(u^*, y^k) + G_y^\top(u^*, y^k)\bar{y}^k
\end{align*}
\]

for \( k = 0, 1, 2, \ldots \)

or, in short, use the Piggyback method

\[ \cdots \to (\text{state,adjoint}) \to (\text{state,adjoint}) \to \ldots \quad (\text{A.G.}) \]
Observations

Observation I  Changes of $y^{k+1} = G(u^*, y^k)$ in primal update cause a **lag-effect** of the adjoint iteration $\bar{y}^{k+1} = f_y(u^*, y^k) + G_y^\top (u^*, y^k) \bar{y}^k$, i.e., $\|L_y(u^*, y^k, \bar{y}^k)\|$ might grow while $y^k$ is not 'sufficiently' converged.

Observation II  Multiple primal for/before an adjoint update reduce the lag effect

Observation III  Primal precision/number of forward iterations unknown for B.C.

**Question**  What are the consequences and can we do better?

**Motivation**  Primal and adjoint updates have usually different execution times
Blurred Piggyback - Best of both worlds

Idea I  Run primal and adjoints in an **asynchronous** parallel fashion, i.e.,

\[ \cdots \rightarrow (\text{state}) \rightarrow (\text{state}) \rightarrow (\text{state}) \rightarrow (\text{state}) \rightarrow \cdots \]

\[ \cdots \rightarrow (\text{adjoint}) \rightarrow (\text{adjoint}) \rightarrow (\text{adjoint}) \rightarrow (\text{adjoint}) \rightarrow \cdots \]

Idea II  Always use the **latest available information** \( y \) for the adjoint update.

**Quasi-Code**

1. **Input:** \( u = u^*, y = t_y, \bar{y} = t_{\bar{y}}; t_y \) shared
2. **while** \((y, \bar{y})\) not converged **do**
3.  Process 0:
4.     Compute state \( t_y = G(t_u, y) \)
5.     Update state \( y = t_y \)
6.  Process 1:
7.     Compute adjoint \( t_{\bar{y}} = f_y(t_u, t_y) + \bar{y}^\top G_y(t_u, t_y) \)
8.     Update adjoint \( \bar{y} = t_{\bar{y}} \)
9. **end while**
10. **Output:** \( y, \bar{y} \)

**Convergence**  Based on \( \|G_y(u, y)\| \leq \rho < 1 \) and smoothness of \( f, G \)
Numerical results for Laplace problem

Objective Tracking type functional

\[ f(u, y) = \frac{1}{2} \| y - y_d \|^2 + \frac{\mu}{2} \| u \|^2, \]

Constraint Laplace

\[ -\Delta y = u^*, \quad y|_{\partial\Omega} = 0 \text{ over } \Omega = [0, 1] \]

Slow Contraction Jacobi-iteration

\[ y^{k+1} = D^{-1}(u^* - Ry^k) \text{ for } A = D + R, \ D \text{ diagonal} \]

Measurements of the runtime seem to be inconsistent with the convergence history

<table>
<thead>
<tr>
<th></th>
<th>A.G.</th>
<th>B.C.</th>
<th>B.P.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PURE</td>
<td>1.35978293e-02</td>
<td>1.5283478e-01</td>
<td>2.7537666e-02</td>
</tr>
<tr>
<td></td>
<td>1.7725375e-02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Numerical results for Laplace problem**

**Objective** Tracking type functional $f(u, y) = \frac{1}{2} \|y - y_d\|^2 + \frac{\mu}{2} \|u\|^2$, $y_d$ triangle

**Constraint** Laplace $-\Delta y = u^*$, $y|_{\partial \Omega} = 0$ over $\Omega = [0, 1]$

**Slow Contraction** Jacobi-iteration $y^{k+1} = D^{-1}(u^* - Ry^k)$ for $A = D + R$, $D$ diagonal

**Measurements** of the runtime seem to be **inconsistent** with the convergence history

<table>
<thead>
<tr>
<th></th>
<th>Pure</th>
<th>A.G.</th>
<th>B.C.</th>
<th>B.P.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
<td>1.35978293e-02</td>
<td>1.5283478e-01</td>
<td>2.7537666e-02</td>
<td>1.7725375e-02</td>
</tr>
<tr>
<td>$G$</td>
<td>$G, f_y + G_y \bar{y}$, sync</td>
<td>$G$ or $f_y + G_y \bar{y}$</td>
<td>$\lambda_1 \ast [G]$ and $\lambda_2 \ast [f_y + G_y \bar{y}]$</td>
<td></td>
</tr>
</tbody>
</table>
Load Balancing

Assumption I  The functions $f$, $G$ and their derivatives can be implemented as **parallel programs** and only scale up to a certain degree.

Assumption II  **Finite amount** of computational resources available $\lambda \in \mathbb{R}^+$

**Idea**  Use **load balancing** $\lambda = \lambda_1 + \lambda_2$ and find a speed up of overall process by optimal distribution $(\lambda_1, \lambda_2)$ of the primal/adjoint updates on available resources.

**Goal**  Minimize computational time $T(\lambda_1, \lambda_2)$ required to find $(y^*, \bar{y}^*)$.

**Example**  of a static load balancing for three different tuple $(\lambda_1, \lambda_2)$.
Dynamic Load Balancing

Automatic setup based on information about $f$, $G$ and their derivatives, residual, scalability, extra cost for data transfer, communication, experience,... develop self adjusting method that efficiently distributes the computational resources during the course of ‘optimization’

Example Consider the following dynamic load balancing

- Phase I: Mainly reduce primal error (B.C. style)
- Phase II: Reduce primal error and adjoint error (A.G./B.P. style)
- Phase III: Mainly reduce adjoint error (B.C. style)
State and Adjoint equation can be solved by three different fixed-point approaches:

**Christianson**

\[ \cdots \rightarrow \text{(state)} \rightarrow \text{(state)} \rightarrow \cdots \rightarrow \text{(adjoint)} \rightarrow \text{(adjoint)} \rightarrow \ldots \quad (\text{B.C.}) \]

**Griewank**

\[ \cdots \rightarrow \text{(state,adjoint)} \rightarrow \text{(state,adjoint)} \rightarrow \ldots \quad (\text{A.G.}) \]

**Blurred**

\[ \cdots \rightarrow \text{(state)} \rightarrow \text{(state)} \rightarrow \text{(state)} \rightarrow \text{(state)} \rightarrow \text{(state)} \rightarrow \ldots \quad (\text{B.P.}) \]

\[ \cdots \rightarrow \text{(adjoint)} \rightarrow \text{(adjoint)} \rightarrow \text{(adjoint)} \rightarrow \text{(adjoint)} \rightarrow \text{(adjoint)} \rightarrow \ldots \]

- **(New) Blurred Piggyback method** (asynchronous + blurred state information for the adjoint) seems to be favorable (at least for this example, this setting,...)
- A.G. and B.C. can be recovered as a **special case** of B.P. by load balancing.
- (Dynamic) load balancing to reduce execution time - that’s the important thing!!!
**Oneshot methods**

**Goal** Solve optimization problem \( \min_{(u, y) \in \mathbb{R}^{m+n}} f(u, y) \) s.t. \( y = G(u, y) \).

**Updates**

\[
\begin{align*}
    u^+ &= u - \alpha B^{-1} L_u (u, y, \bar{y}), \\
    y^+ &= G(u, y), \text{ and} \\
    \bar{y}^+ &= \bar{y} + L_y (u, y, \bar{y})
\end{align*}
\]

**Sequencing** Combine the three update steps into an optimization method. Choose order of state, adjoint and design updates, their multiplicity \( s \), parameters \( B \) and \( \alpha \) to ensure (optimal) contraction of the combined method towards stationary points.

**Scenarios**

\[
\begin{align*}
    &\cdots \rightarrow \text{(design,state,adjoint)} \rightarrow \text{(design,state,adjoint)} \rightarrow \cdots \quad \text{(Jacobi)} \\
    &\cdots \rightarrow \text{(design)} \rightarrow \text{(state)} \rightarrow \text{(adjoint)} \rightarrow \text{(design)} \rightarrow \cdots \quad \text{(Seidel)} \\
    &\cdots \rightarrow \text{(design)} \rightarrow \text{(state,adjoint)} \rightarrow \text{(design)} \rightarrow \cdots \quad \text{(Mixed)} \\
    &\cdots \rightarrow \text{(design)} \rightarrow \text{(state)}^s \rightarrow \text{(adjoint)}^s \rightarrow \text{(design)} \rightarrow \cdots \quad \text{(Multi-Step)}
\end{align*}
\]
Most wanted Oneshot methods

**Jacobi** method by Griewank/Gauger/Slawig/Kaland...

\[ \cdots \rightarrow (\text{design}, \text{state}, \text{adjoint}) \rightarrow (\text{design}, \text{state}, \text{adjoint}) \rightarrow \cdots \]

with doubly augmented Lagrangian preconditioner

\[ B \approx L_{uu} + \alpha G_u^\top G_u + \beta L_{uy} L_{yu} \]

**Seidel** (multi-step) by B/Lehmann/Griewank

\[ \cdots \rightarrow (\text{design}) \rightarrow (\text{state})^S \rightarrow (\text{adjoint})^S \rightarrow (\text{design}) \rightarrow \cdots \]

with projected Lagrangian preconditioner (or modifications)

\[ B \approx \begin{bmatrix} I & Z^\top \end{bmatrix} \begin{bmatrix} L_{uu} & L_{uy} \\ L_{yu} & L_{yy} \end{bmatrix} \begin{bmatrix} I \\ Z \end{bmatrix}, \quad Z = (I - G_y)^{-1} G_u. \]

**Generalization** Both/(almost)\(^a\) all considered one-shot methods are a special case of the following general one-shot method and can be derived by a suitable load balancing schemes

\(^a\)to the best of my knowledge one can even skip 'almost'
Algorithm 1 General Oneshot Method

1: **Input:** $u, y, \bar{y}, \alpha, B$
2: **while** $(u, y, \bar{y})$ not converged **do**
3:     Process 0:
4:         Monitor processes, determine allocation of resources
5:     Process 1:
6:         Update $\alpha$ and $B$
7:         Compute design $t_u = u - \alpha B^{-1} L_u(u, y, \bar{y})$
8:     Process 2:
9:         Compute state $t_y = G(u, y)$
10:    Process 3:
11:         Compute adjoint $t_{\bar{y}} = L_y(u, y, \bar{y}) + \bar{y}$
12:    Process 4:
13:         Update design $u = t_u$
14:    Process 5:
15:         Update state $y = t_y$
16:    Process 6:
17:         Update adjoint $\bar{y} = t_{\bar{y}}$
18: **end while**
19: **Output:** $u, y, \bar{y}, \alpha, B$
Load balancing

Resources  need to be distributed for 0+1+1+1+3 processes.

Jacobi  method and corresponding load balancing

\[ \cdots \rightarrow (\text{design}, \text{state}, \text{adjoint}) \rightarrow (\text{design}, \text{state}, \text{adjoint}) \rightarrow \cdots \]

M.Seidel  method and corresponding load balancing

\[ \cdots \rightarrow (\text{design}) \rightarrow (\text{state})^S \rightarrow (\text{adjoint})^S \rightarrow (\text{design}) \rightarrow \cdots \]

Problem  Both methods (Jacobi, Multi-Seidel) are not optimal:

idle vs. cross-communication
Blurred Oneshot method

Idea Run all three processes (design,state,adjoint) in parallel but in an asynchronous fashion and allow blurred updates analogous to the blurred piggyback approach.

WARNING We have a coupling: $u$ depends on $y, \bar{y}$ and vice versa.

Solution Use dynamic load balancing to enforce convergence.

Load balancing should be mostly stable for efficiency reasons but may still vary.

Intuition In the worst case, one can slow down the control updates by reducing the computational resources for this process and crawl along/to the manifold primal and adjoint feasible Manifold $M$. 
Blurred Oneshot Algorithm

1: **Input:** $u = t_u$, $y = t_y$, $\bar{y} = t_{\bar{y}}$, $\alpha$, $B$, $(t_u, t_y, t_{\bar{y}}$ shared )
2: **while** $(u, y, \bar{y})$ not converged **do**
3:   Process 0:
4:     Monitor processes, determine allocation of resources
5:   Process 1:
6:     Update $\alpha$ and $B$
7:     Compute design $t_u = u - \alpha B^{-1} L_u(u, t_y, t_{\bar{y}})$
8:     Update design $u = t_u$
9:   Process 2:
10:    Compute state $t_y = G(t_u, y)$
11:    Update state $y = t_y$
12:   Process 3:
13:    Compute adjoint $t_{\bar{y}} = f_y(t_u, t_y) + \bar{y}^\top G_y(t_u, t_y)$
14:    Update adjoint $\bar{y} = t_{\bar{y}}$
15: **end while**
16: **Output:** $u$, $y$, $\bar{y}$, $\alpha$, $B$
Numerical results for Laplace design optimization problem

**Setup**  Blurred Oneshot with L-BFGS approximation of projected Hessian, no load balancing (not yet), $\alpha$ by quasi-Backtrack

**Residuals**  of a typical optimization evaluation

![Graph showing residual vs iteration](image-url)
**Problem**

\[
\min_{u,y} \frac{1}{T} \int_0^T f(u, y(\tau)) d\tau \quad \text{such that}
\]

\[
\frac{\partial y(\tau)}{\partial \tau} + c(y(\tau), u) = 0 \quad \forall \tau \in [0, T], \quad y(0) = y_0^* 
\]

**State**

\( y : [0, T] \rightarrow Y \) is a function that varies over a time interval \([0, T] \subset \mathbb{R}\)

**Time**

Discr. \( 0 = \tau_0 < \tau_1 < \cdots < \tau_N = T \) and approx. \( y \approx y_i = y(\tau_i) \in \mathbb{R}^n \)

**Assume I**

Time integration of the constraint is a one-step method\(^a\) and state \( y_i \) at time \( t_i \) is implicitly given by state \( y_{i-1} \) at previous time \( t_{i-1} \), i.e.,

\[
R(u, y_i, y_{i-1}) = 0, \quad i = 1, \ldots, N 
\]

**Assume II**

that there exists a contractive fixed-point solver

\[
y_i^{k+1} = G(u, y_i^k, y_{i-1}) : U \times Y \rightarrow Y 
\]

to compute solutions \( \lim_{k \rightarrow \infty} y_i^k \) of the previous integration scheme \( R \) with contraction rate \( \| G_{y_i^k}(u, y_i^k, y_{i-1}) \| \leq \rho_G < 1 \).

\(^a\)E.g. implicit Euler - although higher order methods are also possible
Extended Fixed-Point iteration

Extended Mapping \( G : U \times Y \rightarrow Y \) with \( Y = Y_N = Y \times \cdots \times Y \) defined by

\[
G(u, y) = \begin{bmatrix}
G_1(u, y) \\
G_2(u, y) \\
\vdots \\
G_N(u, y)
\end{bmatrix} = \begin{bmatrix}
G(u, y_1, y_0) \\
G(u, y_2, G(u, y_1, y_0)) \\
\vdots \\
G(u, y_N, G(u, y_{N-1}, G(\cdots G(u, y_1, y_0)) \cdots)))
\end{bmatrix}
\]

Contraction is inherited by the extended mapping \( G \), i.e.

\[
\|G_y(u, y)\| \leq \rho_G = \rho_G < 1.
\]

Extended Problem can be formulated

\[
\min_{u, y} f(u, y) \text{ such that } y = G(u, y) \text{ with } y_0 = y(0)
\]

where the objective objective functional is approximated by

\[
f(u, y) = \sum_{i=1}^{N} \Delta t_i f_N(u, y_i) \approx \frac{1}{T} \int_0^T f(u, y(t))
\]
Interpretation  Extended fixed-point mapping can be understood as parallel update on the discrete state approximation for each time-step.

Lag-Effect and different execution times of $G$

Blurred updates with **Load Balancing** for different time-steps strongly recommended!
Blurred Oneshot Method for time-dependent Problems

1: **Input:** \( u = t_u, y = (t_{y_1}, \ldots, t_{y_N}), \bar{y}_i = t_{\bar{y}_i}, \alpha, B \)

2: **while** \((u, y_1, \ldots, y_N, \bar{y}_1, \ldots, \bar{y}_N)\) not converged do

3:    Process 0: Monitor processes, determine allocation of resources

4:    Process 1:

5:        Update \( \alpha \) and \( B \)

6:        Compute design \( t_u = u - \alpha B^{-1} L_u(u, t_{y_1}, \ldots, t_{y_N}, t_{\bar{y}_1}, \ldots, t_{\bar{y}_N}) \)

7:        Update design \( u = t_u \)

8:    Process 2:

9:        Compute state \( t_{y_1} = G(t_u, y_1, y_0) \)

10:       Update state \( y_1 = t_{y_1} \)

11:    Process 3:

12:        Compute state \( t_{y_2} = G(t_u, y_2, t_{y_1}) \)

13:       Update state \( y_2 = t_{y_2} \)

14:        \ldots

15:    Process N+1:

16:        Compute state \( t_{y_N} = G(t_u, y_N, t_{y_{N-1}}) \)

17:       Update state \( y_N = t_{y_N} \)

18:    

19:    Process N+2:

20:        Compute adjoint \( t_{\bar{y}_1} = f_y(t_u, t_{y_1}) + G_y(t_u, t_{y_1}, \ldots, t_{y_N})^\top \bar{y}_1 \)

21:       Update adjoint \( \bar{y}_1 = t_{\bar{y}_1} \)

22:        \ldots

23:    Process 2N+1:

24:        Compute adjoint \( t_{\bar{y}_N} = f_y(t_u, t_{y_N}) + G_y(t_u, t_{y_1}, \ldots, t_{y_N})^\top \bar{y}_N \)

25:       Update adjoint \( \bar{y}_N = t_{\bar{y}_N} \)

26: **end while**
Observation

Load balance I for the extended primal iteration, e.g.

Load balance II similar for extended adjoint iteration
Load balance III between primal and adjoint updates still possible
Load balance IV between (primal,adjoint) and design also possible

GOAL Allocating the resources to all 2N+2 processes needs to be harmonized like an orchestra to achieve maximal efficiency.
Summary

Blurred Piggyback/Oneshot using asynchronous and inconsistent updates
General Piggyback/Oneshot method for design optimization
Extension to Time dependent problems can be improved by
(Optimal) Load Balancing for use in HPC, convergence, efficiency
Outlook - TODO It’s still a loooooong way to go....

Thank you for your attention!
Shared memory in MPI 3.0 could be a great challenge for AD in the future. Any suggestions? (Except of saying: Don’t use it!)