Distributed solution of stochastic optimal control problems on GPUs

Ajay K. Sampathirao[a], Pantelis Sopasakis[a], Alberto Bemporad[a] and Panagiotis Patrinos[b]

***Abstract*— Stochastic optimal control problems arise in many applications and are, in principle, large-scale involving up to millions of decision variables. Their applicability in control applications is often limited by the availability of algorithms that can solve them efficiently and within the sampling time of the controlled system.**

In this paper we propose a dual accelerated proximal gradient algorithm which is amenable to parallelization and demonstrate that its GPU implementation affords high speed-up values (with respect to a CPU implementation) and greatly outperforms well-established commercial optimizers such as Gurobi.

***Index Terms*— Stochastic optimal control, accelerated proximal gradient, graphics processing unit (GPU).**

I. Introduction

A. Background

Stochastic optimal control problems typically give rise to large-scale optimization problems involving up to tens of millions of constrained decision variables. Such problems arise in stochastic model predictive control of Markovian switching systems [1], stochastic control of networked systems [2] and of large-scale uncertain systems [3], portfolio optimization under uncertainty [4], inventory management [5], management of supply chain systems [6] and in many other applications of stochastic optimal control.

Despite their popularity, control engineering practice has taken little initiative towards adopting the theoretical results of stochastic optimal control theory, and this is mainly due to the prohibitive computational footprint that accompanies it. The increasing need for computational power gives the floor to graphics processing units (GPUs) and field programmable gate arrays (FPGAs) which are gaining momentum in control applications [7]–[9]. Since one's ability to apply stochastic control methodologies is conditioned by the system's sampling rate, the need for algorithms and hardware that can solve such problems fast is becoming imperative.

Recently, Constantinides authored a tutorial paper in which he outlines the potential advantages of the use of FPGAs and GPUs for (deterministic) model predictive control (MPC) applications [10]. Rogers and Slegers demonstrated the potential of *in situ* GPU-aided controllers for the guidance of a parafoil where Monte-Carlo simulations are performed in real-time to counteract the wind uncertainty [9]. Although, there have been efforts to parallelize algorithms for the solution of MPC-related optimization problems (see [11]–[13]), there is no approach that exploits the structure of stochastic optimal control problems to achieve high computational throughput. For example, the approach of Di Cairano *et al.* [12] treats the MPC optimization problem as a general quadratic programming problem and, as a result, cannot scale up with the problem size. A GPU-based framework to solve large scale two-stage stochastic optimization problems with applications to uncertain energy dispatch systems is presented in [13].

In this paper we propose a scalable parallelizable algorithm for solving multi-stage stochastic optimal control problems. We use an accelerated proximal gradient method applied to the Fenchel dual optimization problem and we exploit the problem's structure to further accelerate computations. The dual gradient is calculated as the solution of an unconstrained minimization problem which we solve by dynamic programming. This problem can be properly decomposed and solved in a parallelized way.

This boils down to an algorithm that requires only matrix-vector products, it is highly parallelizable and can be readily implemented on a GPU or FPGA architecture. The algorithm is well-suited the control for linear dynamical systems with additive and multiplicative uncertain components assuming that these are driven by a stochastic process that can be modeled as an evolution along a scenario tree. The proposed algorithm is division-free and therefore, as O'Donoghue *et al.* accentuate in [14], it is suitable for embedded applications using fixed-point arithmetic. Moreover, we allow the cost function of the optimization problem to have a nonsmooth part which can be used to encode hard or soft constraints, or terms involving $\|\cdot\|_1$.

B. Notation and Mathematical Preliminaries

Let $\mathbb{R}$, $\mathbb{N}$, $\mathbb{R}^n$, $\mathbb{R}^{m\times n}$, $\mathbb{S}^n_+$, $\mathbb{S}^n_{++}$ denote the sets of real numbers, non-negative integers, column real vectors of length n, real matrices of dimensions m-by-n, symmetric positive semidefinite and positive definite n-by-n matrices respectively. Let $\bar{\mathbb{R}} = \mathbb{R}\cup\{\pm\infty\}$ denote the set of extended-real numbers. The transpose of a matrix A is denoted by A'. The set of of nonnegative numbers $\{k_1, k_1+1, \ldots, k_2\}, k_2 \geq k_1$ is denoted by $\mathbb{N}_{[k_1,k_2]}$.

Given a norm $\|\cdot\|$ on $\mathbb{R}^n$ we define the conjugate norm $\|\cdot\|_*$ defined as $\|y\|_* = \sup_{\|x\|\leq 1}\langle x, y\rangle$, where $\langle x, y\rangle$ is the standard inner product on $\mathbb{R}^n$. For a matrix $A\in\mathbb{R}^{m\times n}$ we define its norm as $\|A\| = \sup_x\{\|Ax\|_*; \|x\|\leq 1\}$. We

[a] IMT Institute for Advanced Studies Lucca, Piazza S. Fransesco 19, 55100 Lucca, Italy. Emails: {a.sampathirao, p.sopasakis, a.bemporad}@imtlucca.it.

[b] KU Leuven, Department of Electrical Engineering (ESAT), STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics, Kasteelpark Arenberg 10, 3001 Leuven, Belgium. Email: panos.patrinos@esat.kuleuven.be

This work was financially supported by the EU FP7 research project EFFINET "Efficient Integrated Real-time monitoring and Control of Drinking Water Networks," grant agreement no. 318556.

denote by $\|\cdot\|_1$ and $\|\cdot\|_2$ the 1-norm and Euclidean norm on $\mathbb{R}^n$.

The *indicator function* of a set $C \subseteq \mathbb{R}^n$ is the extended-real valued function $\delta(\cdot|C) : \mathbb{R}^n \to \bar{\mathbb{R}}$ and for $x \in C$ it is $\delta(x|C) = 0$ and $\delta(x|C) = +\infty$ otherwise. A function $f : \mathbb{R}^n \to \bar{\mathbb{R}}$ is called *lower semi-continuous* or *closed* if for every $x \in \mathbb{R}^n$, $f(x) = \liminf_{z\to x} f(z)$. A $f : \mathbb{R}^n \to \bar{\mathbb{R}}$ is called *proper* if there is a $x \in \mathbb{R}^n$ so that $f(x) < \infty$ and $f(x) > -\infty$ for all $x \in \mathbb{R}^n$. For a closed convex function $f : \mathbb{R}^n \to \bar{\mathbb{R}}$, we define its *conjugate* as $f^*(x^*) = \sup_x\{\langle x, x^*\rangle - f(x)\}$. A mapping $F : \mathbb{R}^n \to \mathbb{R}^m$ is called β-*Lipschitz continuous*, with $\beta \geq 0$, if $\|F(x_1) - F(x_2)\|_* \leq \beta\|x_1 - x_2\|$ for every $x_1, x_2 \in \mathbb{R}^n$. We call f σ-*strongly convex* if $f(x) - \frac{\sigma}{2}\|x\|_2^2$ is a convex function. Unless otherwise stated $\|\cdot\|$ stands for $\|\cdot\|_2$.

Every nonempty closed convex set $C \subseteq \mathbb{R}^n$ defines the convex function $\Pi(x|C) = \operatorname{argmin}_{c\in C} \|x - c\|_2$, which is called the *(Euclidean) projection* of x onto C. The Euclidean distance of a $x \in \mathbb{R}^n$ from C is defined as $d(x|C) = \min_{c\in C} \|x - c\|_2$.

II. Problem Statement

In this section we present the general stochastic optimal control problem and the algorithm to solve it.

A. Stochastic optimal control

Consider the following discrete-time stochastic linear system with additive disturbance

$$x_{k+1} = A_{\xi_k}x_k + B_{\xi_k}u_k + w_{\xi_k}, \tag{1}$$

where $x_k \in \mathbb{R}^{n_x}$ is the system state, $u_k \in \mathbb{R}^{n_u}$ is the input, and w_{ξ_k} is an additive disturbance term. Let Ω_k be the sample space of the random variable ξ_k. In what follows we shall consider that each Ω_i, $i \in \mathbb{N}$ is a nonempty finite set. We define the product space $\Omega = \prod_{i\in\mathbb{N}} \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \Omega_i$ and the probability space $(\Omega, \mathfrak{F}, \{\mathfrak{F}_k\}_{k\in\mathbb{N}}, \mathrm{P})$, where $\mathfrak{F}$ is the Borel σ-algebra on Ω and $\{\mathfrak{F}_k\}_{k\in\mathbb{N}}$ is a filtration on Ω where $\mathfrak{F}_k$ is the Borel σ-algebra on $\prod_{i=0}^{k} \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \Omega_i$. Finally, P is the product probability measure on Ω.

For system (1) we formulate the following *stochastic optimal control problem* of horizon $N \in \mathbb{N}$ with decision variable $\pi = \{u_k\}_{k=0}^{N-1}$

$$V^\star(p) = \min_{\pi} \mathbb{E}\left[V_f(x_N, \xi_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k, \xi_k)\right], \tag{2}$$

where $\mathbb{E}$ is the conditional expectation with respect to the above product measure P, while the states follow the dynamics in (1) with given initial condition $x_0{=}p$. The decision variables u_k in (2) are functions $u_k = \psi_k(p, \boldsymbol{\xi}_{k-1})$ with $\boldsymbol{\xi}_k = (\xi_0, \xi_1, \ldots, \xi_k)$. This means that decisions at time k are taken using only prior knowledge, *i.e.*, in a causal fashion. As we are about to explain, problem (2) encompasses a very wide class of optimization problems by allowing the stage cost ℓ and the terminal cost V_f to be extended-real valued functions. In what follows the stage cost can be readily taken to be a function of both k and $\boldsymbol{\xi}_k$, *i.e.*, it can be $\ell(x_k, u_k, \boldsymbol{\xi}_k, k)$, but for the sake of simplicity of presentation we will consider the simpler, yet quite general, case of Equation (2).

In (2) the stage cost function ℓ is written as $\ell(x, u, \xi) = \phi(x, u, \xi) + \bar{\phi}(F_\xi x + G_\xi u, \xi)$ where ϕ is real-valued, smooth in (x, u), and strongly convex over the manifold that defines the system dynamics, while $\bar{\phi}$ is an extended-real valued function, lower semi-continuous, proper, convex and possibly nonsmooth. Likewise, V_f can be decomposed as $V_f(x, \xi) = \phi_N(x, \xi) + \bar{\phi}_N(F_{N,\xi}x, \xi)$.

Function $\bar{\phi}$ can be chosen to be any nonsmooth function as dictated by the problem we need to solve. We can use $\bar{\phi}$ to encode arbitrary hard constraints on states and inputs of the form $F_{\xi_k}x_k + G_{\xi_k}u_k \in Y_{\xi_k}$ by choosing

$$\bar{\phi}(\cdot, \xi_k) = \delta(\cdot|Y_{\xi_k}), \tag{3}$$

where Y_{ξ_k} are nonempty convex closed sets for which projections $\Pi(\cdot|Y_{\xi_k})$ can be easily computed. Soft constraints can be encoded by choosing

$$\bar{\phi}(\cdot, \xi_k) = \eta_{\xi_k} d(\cdot|Y_{\xi_k}), \tag{4}$$

where $\eta_{\xi_k} > 0$ is a scaling factor, while one may also choose

$$\bar{\phi}(\cdot, \xi_k) = \|\cdot\|_1 \tag{5}$$

to force the optimizer to be sparse.

The smooth part of the stage cost ℓ is a quadratic function of the form $\phi(x_k, u_k, \xi_k) = x_k'Q_{\xi_k}x_k + u_k'R_{\xi_k}u_k$, where $R_{\xi_k} \in \mathbb{S}_{++}^{n_u}$ and $Q_{\xi_k}\mathbb{S}_{+}^{n_x}$. The smooth part of the terminal cost function V_f is a quadratic function $\phi_N(x_N, \xi_N) = x_N'P_{\xi_N}x_N$, with $P_{\xi_N} \in \mathbb{S}_{++}^{n_x}$. The function $\bar{\phi}_N$ can be selected in the same way as we have explained for $\bar{\phi}$, *e.g.*, terminal constraints of the form $F_{N,\xi}x_N \in X_f$ can be encoded using $\bar{\phi}_N(\cdot, \xi) = \delta(\cdot|X_f)$, where X_f is assumed to be such that $\Pi(\cdot|X_f)$ can be easily evaluated computationally.

B. Proximal gradient algorithms

In this section we briefly present how a dual proximal gradient algorithm can be used to solve the optimization problem presented in the previous section. The *proximal* operator $\operatorname{prox}_{\gamma f} : \mathbb{R}^n \to \mathbb{R}^n$ of a closed, convex, proper extended-real valued function $f : \mathbb{R}^n \to \bar{\mathbb{R}}$ is defined as

$$\operatorname{prox}_{\gamma f}(v) = \operatorname*{argmin}_{x\in\mathbb{R}^n} \left\{f(x) + \frac{1}{2\gamma}\|x - v\|_2^2\right\}, \tag{6}$$

for $\gamma{>}0$, and has interesting properties outlined in [15].

The optimization problem presented in Section II-A (as we will discuss in detail in Section III-A) can be compactly written in the form

$$P^\star = \min_{z=Hx} f(x) + g(z), \tag{7}$$

where $f : \mathbb{R}^n \to \bar{\mathbb{R}}$ is strongly convex and $g : \mathbb{R}^m \to \bar{\mathbb{R}}$ is closed, proper and convex. The Fenchel dual of (7) is written as [16, Corol. 31.2.1]:

$$D^\star = \min_y f^*(-H'y) + g^*(y), \tag{8}$$

and since f is assumed to be σ-strongly convex, f^* has Lipschitz-continuous gradient with constant $1/\sigma$ because of [17, Prop. 12.60]. Under the prescribed assumptions, strong duality holds, thus $P^\star = D^\star$.

The proximal gradient algorithm applied to the dual problem is then defined as the recursion on dual variables y^ν [18]

$$y^{\nu+1} = \text{prox}_{\lambda g^*}(y^\nu + \lambda H \nabla f^*(-H'y^\nu)), \tag{9}$$

starting from a dual-feasible vector y^0, *e.g.*, $y^0 = 0$. In light of the conjugate subgradient theorem [16, Thm. 23.5], the gradient in (9) can be written as

$$x^\nu \triangleq \nabla f^*(-H'y^\nu) = \underset{z}{\text{argmin}}\{\langle z, H'y^\nu\rangle + f(z)\}, \tag{10}$$

where the last problem is an unconstrained minimization problem which can be solved very efficiently as we will explain in the next section. The proximal operator in (9) can be evaluated as follows

$$z^\nu = \text{prox}_{\lambda^{-1}g}(\lambda^{-1}y^\nu + Hx^\nu), \tag{11}$$
$$y^{\nu+1} = y^\nu + \lambda(Hx^\nu - z^\nu). \tag{12}$$

The above is easily derived using the *Moreau decomposition* property $v = \text{prox}_{\gamma f}(v) + \gamma\,\text{prox}_{\gamma^{-1}f^*}(\gamma^{-1}v)$.

The proximal operator in (11) can be easily evaluated for a great variety of functions such as the ones discussed in the previous section [19]. For example for $g(x) = \delta(x|C)$ it is $\text{prox}_{\lambda g}(v) = \Pi(v|C)$. The proximal operator of $d(\cdot|C)$ can also be easily computed provided that C is simple enough so that both $d(\cdot|C)$ and $\Pi(\cdot|C)$ can be computed easily [19].

The proximal gradient method, given by (10)–(12), produces a sequence y^ν which, for properly small λ, converges to a dual optimal solution $y^\star$, while the corresponding primal sequences x^ν and z^ν converge to the (unique) primal optimal solution $(x^\star, z^\star)$.

Accelerated Proximal Gradient (APG) Algorithm: Nesterov proposed an accelerated version of the above algorithm to achieve a convergence rate of $\mathcal{O}(1/\nu^2)$ [20], according to which the dual gradient in (10) is calculated at an extrapolated dual vector of the form $v^\nu = y^\nu + \beta_\nu(y^\nu - y^{\nu-1})$. Nesterov's algorithm is summarized as follows:

$$v^\nu = y^\nu + \theta_\nu(\theta_{\nu-1}^{-1} - 1)(y^\nu - y^{\nu-1}), \tag{13a}$$
$$x^\nu = \text{argmin}_z\{\langle z, H'v^\nu\rangle + f(z)\}, \tag{13b}$$
$$z^\nu = \text{prox}_{\lambda^{-1}g}(\lambda^{-1}v^\nu + Hx^\nu), \tag{13c}$$
$$y^{\nu+1} = v^\nu + \lambda(Hx^\nu - z^\nu), \tag{13d}$$
$$\theta_{\nu+1} = 1/2(\sqrt{\theta_\nu^4 + 4\theta_\nu^2} - \theta_\nu^2), \tag{13e}$$

starting with $y^0 = y^{-1}$, $\theta_0 = \theta_{-1} = 1$. Note that the values of the sequence $\theta_\nu(\theta_{\nu-1}^{-1} - 1)$ can be precomputed in a floating point machine so that the above algorithm is division-free. The fast convergence results for the dual iterates do not translate to equally fast convergence for the primal iterate [21], but an ergodic iterate defined through the recursion $\bar{z}^\nu = (1-\theta_\nu)\bar{z}^{\nu-1} + \theta_\nu z^\nu$, $\bar{z}^{-1} = 0$, *i.e.*, a weighed average of the primal iterates, converges at rate $\mathcal{O}(1/\nu^2)$ [22].

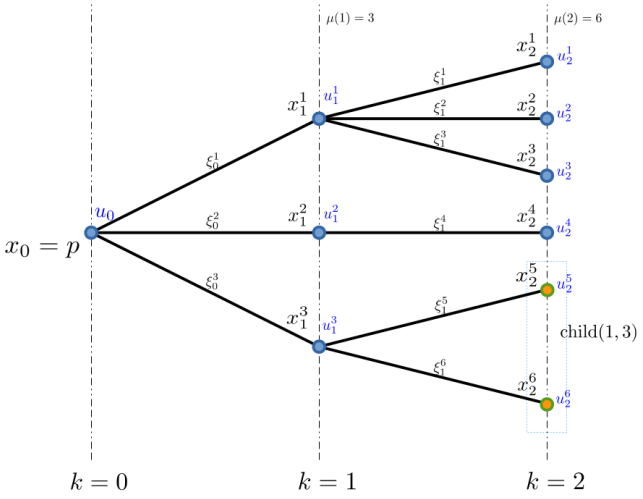

Fig. 1: Example of a scenario tree structure.

Choice of λ: APG converges to a primal-dual solution given that $\lambda \in (0, 1/L]$, where L is a Lipschitz constant of the function $\hat{f}(y) = \nabla f^*(-H'y)$, which can be easily evaluated as $\frac{\|H\|^2}{\sigma}$ as in [21]. When f is of the form $f(x) = x'Qx + \delta(x|E)$, where E is an affine space and f is strictly convex, then f^* is quadratic and its Lipschitz constant can be computed directly and, in particular, we can compute a tight Lipschitz constant.

Termination criteria: We need to terminate the algorithm so that the produced primal solution (x, z) is ϵ_V–suboptimal, *i.e.*, $f(x)+g(z)-P^\star \le \epsilon_V$ and ϵ_g–infeasible in the sense that $\|x - Hz\| \le \epsilon_g$. Such a solution is called (ϵ_g, ϵ_V)-optimal. Computationally tractable termination criteria to achieve this avoiding computationally expensive operations at every iteration are provided by Patrinos and Bemporad [22].

Preconditioning: As all first order methods, the proposed algorithm is sensitive to scaling. Preconditioning can considerably improve the performance of the algorithm. A simple preconditioning that works well in practice is to compute a diagonal approximation $\tilde{H}$ of the dual Hessian and perform a change of coordinates in the dual variable y with scaling matrix $\tilde{H}^{-\frac{1}{2}}$ [23, Sec. 2.3.1]. More elaborate preconditioning schemata have been proposed such as [24].

III. Parallelizable APG for Stochastic Optimal Control Problems

In this section we explain how the structure of a stochastic optimal control problem can be exploited in the context of APG to obtain a parallelizable implementation which will be facilitated by structuring the uncertainty as a scenario tree.

A. Scenario trees and scenario-based optimization

A scenario tree is exactly the structure dictated by the filtration of our probability space. This is illustrated in Fig. 1. In practice, such scenario trees can be generated, *inter alia*, from real data by the algorithm proposed by Heitsch and Römisch [25].

The node at time 0 is called *root* and nodes at stage N are called the *leaves* of the tree. Scenarios are sequences of admissible events spanning from the root node. The number of nodes at time k is denoted by $\mu(k)$, while the total number of nodes of the tree is μ. Non-leaf nodes have a set of *children*; for $i \in \mathbb{N}_{[1,\mu(k)]}$ we denote the children nodes of the i-th node at stage k by $\text{child}(k,i) \subseteq \mathbb{N}_{[1,\mu(k+1)]}$. The probability that we visit node i at stage k starting from the root node is denoted by p_k^i and clearly for all $k \in \mathbb{N}_{[0,N]}$ it is $\sum_{i=1}^{\mu(k)} p_k^i = 1$ and for $k \in \mathbb{N}_{[0,N-1]}$ and $i \in \mathbb{N}_{[1,\mu(k)]}$ $\sum_{j\in\text{child}(k,i)} p_{k+1}^j = p_k^i$.

The system dynamics along the tree, according to (1), is described by the recursion $x_{k+1}^j = A_k^j x_k^i + B_k^j u_k^i + w_k^j$, with $j \in \text{child}(k,i)$.

Taking into account the tree structure of the stochastic process ξ_k, the stochastic optimal control problem (2) can be restated in terms of the decision variable $\mathbf{x} = \{\{x_k^i\}_{k,i}, \{u_k^i\}_{k,i}\}$ as

$$V^\star(p) = \min_{\mathbf{x}\in\mathcal{X}(p)} \sum_{k=0}^{N-1}\sum_{i=0}^{\mu(k)} p_k^i \ell(x_k^i, u_k^i, i) + \sum_{i=0}^{\mu(N)} p_N^i V_f^i(x_N^i, i)$$

where $\mathcal{X}(p) = \{\mathbf{x} | x_{k+1}^j = A_k^j x_k^i + B_k^j u_k^i + w_k^j, \forall k \in \mathbb{N}_{[0,N-1]}, i \in \mathbb{N}_{[1,\mu(k)]}, j \in \text{child}(i,k)\}$ describes the system dynamics. The splitting we introduced in (7) now becomes

$$\begin{aligned} f(\mathbf{x}) &= \sum_{i=0}^{\mu(k)} p_k^i \phi(x_k^i, u_k^i, i) \\ &\quad + \sum_{i=0}^{\mu(N)} p_N^i \phi_N(x_N^i, i) + \delta(\mathbf{x}|\mathcal{X}(p)), \\ g(H\mathbf{x}) &= \sum_{i=0}^{\mu(k)} p_k^i \bar{\phi}(F_k^i x_k^i + G_k^i u_k^i, i) + \sum_{i=0}^{\mu(N)} p_N^i \bar{\phi}_N(F_N^i x_N^i, i), \end{aligned}$$

where $F_k^i \in \mathbb{R}^{n_{c,k}^i \times n_x}$, $G_k^i \in \mathbb{R}^{n_{c,k}^i \times n_u}$ and $F_N^i \in \mathbb{R}^{n_f^i \times n_x}$. With this choice of f and g the stochastic optimal control problem is equivalent to (7).

There is an alternative formulation where the scenario tree is decomposed as a collection of independent scenarios on which one imposes the so called *non-anticipativity constraints* that enforce the tree structure we described above [26]. That formulation, however, would lead to a significantly higher number of decision variables without facilitating the solution of the problem and was, therefore, not further pursued.

B. Dual gradient computation

The efficient computation of the dual gradient in (13b) is of crucial importance for the performance of the algorithm. The solution of the unconstrained optimization problem (13b) is done by dynamic programming and the solution is separated in two steps: the factor step (Algorithm 1) and the solution step (Algorithm 2). Having performed the factor step, the solution can be obtained at a very low computational cost. In case we have only one scenario as in deterministic optimal control these algorithms are the factor and solve steps described in [22]. Notice that both algorithms are parallelizable across all nodes at every stage of the scenario tree (*nodal decomposition*).

Algorithm 1 Factor step

for $k = N-1, \dots, 0$ **do**
  for $i \in \mu(k)$ **do** {in parallel}
    $\bar{P}_{k+1}^i \leftarrow \sum_{j\in\text{child}(k,i)} B_k^{j\prime} P_{k+1}^j B_k^j$
    $\bar{R}_k^i \leftarrow 2(p_k^i R_k^i + \bar{P}_{k+1}^i)$, $\Phi_k^i = -(\bar{R}_k^i)^{-1} G_k^{i\prime}$
    $K_k^i \leftarrow -2(\bar{R}_k^i)^{-1} \sum_{j\in\text{child}(k,i)} B_k^{j\prime} P_{k+1}^j A_k^j$
    $\sigma_k^i \leftarrow -2(\bar{R}_k^i)^{-1} \sum_{j\in\text{child}(k,i)} B_k^{j\prime} P_{k+1}^j w_k^j$
    $\bar{A}_k^j \leftarrow A_k^j + B_k^j K_k^i$, $D_k^j = F_k^j + G_k^j K_k^i$, $\forall j \in \text{child}(k,i)$
    $c_k^i \leftarrow 2\sum_{j\in\text{child}(k,i)} \bar{A}_k^{j\prime} P_{k+1}^j w_k^j$
  if $k = N-1$ **then**
    $\Theta_{N-1}^j \leftarrow -(\bar{R}_{N-1}^j)^{-1} B_{N-1}^{j\prime} F_N^{j\prime}, \forall j \in \text{child}(N-1,i)$
    $\Lambda_{N-1}^j \leftarrow F_N^j \bar{A}_{N-1}^{j\prime}, \forall j \in \text{child}(N-1,i)$
  else
    $\Theta_k^j \leftarrow -(\bar{R}_k^j)^{-1} B_k^{j\prime}, \forall j \in \text{child}(k,i)$
    $\Lambda_k^j \leftarrow \bar{A}_k^{j\prime}, \forall j \in \text{child}(k,i)$
  end if
  $P_k^i \leftarrow p_k^i (Q_k^i + K_k^{i\prime} R K_k^i) + \sum_{j\in\text{child}(k,i)} \bar{A}_k^{j\prime} P_{k+1}^j \bar{A}_k^j$
  end for
end for

The factor step is performed once before the execution of the APG algorithm, and produces the matrices Φ_k^i, Θ_k^i, Λ_k^i, D_k^i, K_k^i and vectors σ_k^i and c_k^i which are then used in Algorithm 2 to calculate the dual gradient at a given vector y. Given that the factor step can be executed on GPU and the involved computations can be parallelized to a good extent, its contribution to the overall computation time is negligible. The sequences $\{x_k^i\}_{k,i}$ and $\{u_k^i\}_{k,i}$ that are produced by Algorithm 2 define the primal iterate x^ν in (13b).

Notice that in Algorithm 1 we do not need to compute the inverse of $\bar{R}_k^i$; this matrix is symmetric and positive definite, so various methods can be used to solve the corresponding linear systems (*e.g.*, Cholesky factorization).

Algorithm 2 Solve step

$q_N^i \leftarrow y_N^i, \forall i \in \mathbb{N}_{[1,\mu(N)]}$, `%Backward substitution`
for $k = N-1, \dots, 0$ **do**
  for $i \in \mu(k)$ **do** {in parallel}
    $u_k^i \leftarrow \Phi_k^i y_k^i + \sum_{j\in\text{child}(k,i)} \Theta_k^j q_{k+1}^j + \sigma_k^i$
    $q_k^i \leftarrow D_k^{i\prime} y_k^i + \sum_{j\in\text{child}(k,i)} \Lambda_k^{j\prime} q_{k+1}^j + c_k^i$
  end for
end for
$x_0^1 = p$, `%Forward substitution`
for $k = 0, \dots, N-1$ **do**
  for $i \in \mu(k)$ **do** {in parallel}
    $u_k^i \leftarrow K_k^i x_k^i + u_k^i$
    for $j \in \text{child}(k,i)$ **do** {in parallel}
      $x_{k+1}^j \leftarrow A_k^j x_k^i + B_k^j u_k^i + w_k^j$
    end for
  end for
end for

Assuming that the row-dimension of all F_k^i is constant and equal to n_c, the backward substitution step of Algorithm 2 involves roughly $\mathcal{O}(N\mu(n_x(n_u+n_c)+n_u(n_x+n_c)))$ flops and the forward step counts $\mathcal{O}(N\mu(n_x^2+2n_x n_u))$ flops –

both depend linearly on the prediction horizon. For a q-ary tree, that is a scenario tree with constant branching factor q, and assuming perfect parallelization, Algorithm 2 is equivalent to a flop count of $\mathcal{O}(Nq(n_x^2+n_xn_c+n_un_c+4n_xn_u))$.

IV. Simulation results

To evaluate the proposed algorithm we formulate the stochastic optimal control problem which corresponds to the stochastic model predictive control problem for a linear discrete-time system with additive and parametric uncertainty as in (1). We consider a system of m aligned interconnected masses by $m-1$ linear spring-dampers of stiffness constant $\kappa=1$ and damping ratio $\beta=0.1$. The manipulated variables are the forces we may exercise on each spring along their principal axes and the state variables are the positions and speeds of the masses. We assume that the system dynamics is obtained by discretizing the continuous-time dynamics with sampling time $T_s=0.5$ and is written as in (1) with $n_x=2m$, and $n_u=m-1$. On the system state and input variables we impose the constraints $-5\leq x_k^i\leq 5$ and $-1\leq u_k^i\leq 1$ for all $k\in\mathbb{N}_{[0,1]}$ and $i\in\mathbb{N}_{[1,\mu(k)]}$. The stage cost was chosen to be $\ell(x,u,\xi)=x'Qx+u'Ru$ with $Q=I_{n_x}$ and $R=I_{n_u}$.

APG was implemented in CUDA-C [27] as presented in the previous section and matrix-vector multiplications were performed using cuBLAS. We compared the GPU-based implementation of APG with the interior point solver of Gurobi which runs on a dual-core environment. The active set algorithm of Gurobi, as well as qpOASES [28] and QPC [29] give computation times that are not very competent, and will therefore be omitted.

Computations on CPU were performed on a $4\times 2.60\,\text{GHz}$ Intel i5 machine with $8\,\text{GB}$ RAM running 64-bit Ubuntu 14.04 and GPU-based computations were carried out on a NVIDIA Tesla C2075 using the CUDA-6.0 API.

The dependence of the computation time on the size of the scenario tree is shown in Fig. 2; trees considered in this experiment had a fixed horizon $N=14$ and in their first stages were binary, i.e., had branching factor 2 and eventually evolved without branching until the end of the horizon. Notice that for a case of 8192 leaf nodes, Gurobi takes $32.6\,\text{s}$ on average (max. $46.8\,\text{s}$), whereas APG with $\epsilon_g=\epsilon_V=0.005$ requires just $1.3\,\text{s}$ (max. $5.92\,\text{s}$). This problem counts $6.39\cdot 10^5$ primal variables, and $1.75\cdot 10^6$ dual variables.

In Fig. 3 we show how the computation times scale with the increase of the horizon of the problem. The problem that corresponds to $N=60$ counts $0.92\cdot 10^6$ primal and $2.0\cdot 10^6$ dual variables and notice that APG with $\epsilon_g=\epsilon_V=0.005$ can solve it 17.6 times faster than Gurobi on average (6.43 times faster for the maximum time).

Compared to a MATLAB implementation, a very high speedup is achieved on GPU for the same algorithm which can be up to $\times 85$ and scales with the problem size as shown in Fig. 4. On average, the GPU implementation of APG for a scenario tree of 8192 leaf nodes is as high as $\times 83$.

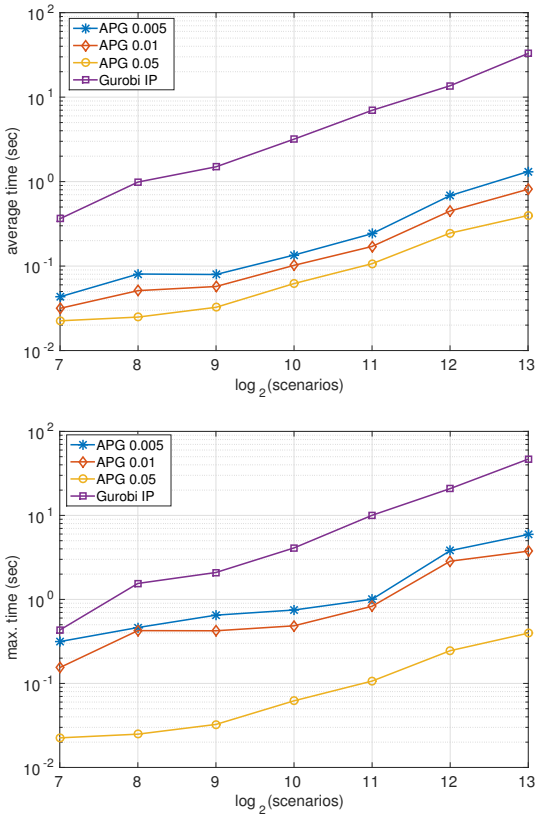

Fig. 2: Dependence of the computation time on the number of scenarios for a system of 10 masses (20 states, 9 inputs, bound constraints) with a fixed prediction horizon $N=14$. Average and maximum computation times reported here are for a random sampling of 100 initial states $x_0=p$.

V. Conclusions and Future work

In this paper we proposed a dual accelerated proximal gradient algorithm tailored for the solution of stochastic optimal control problems. The computation of the dual gradient at every iteration of the algorithm can be parallelized to offer a significant benefit in terms of speed-up. In particular computations are executed in parallel across all nodes at every stage of the scenario tree. As a result, for the special case of a tree with branching only at the root node (known as a *scenario fan*) stochastic MPC can be solved at the computational cost of deterministic MPC provided that the GPU has adequate computational capacity to accommodate the problem size. A CUDA-C implementation of the algorithm that runs on a GPU was found to outperform most state-of-the-art solvers that run on a multi-core CPU.

Additional speedup can be obtained by using advanced parallelization techniques for the computation of matrix-vector products such as [30]. In CUDA, fine tuning can play a crucial role for the overall performance, therefore the results presented here are only indicative and can be improved.

References

[1] P. Patrinos, P. Sopasakis, H. Sarimveis, and A. Bemporad, "Stochastic model predictive control for constrained discrete-time Markovian

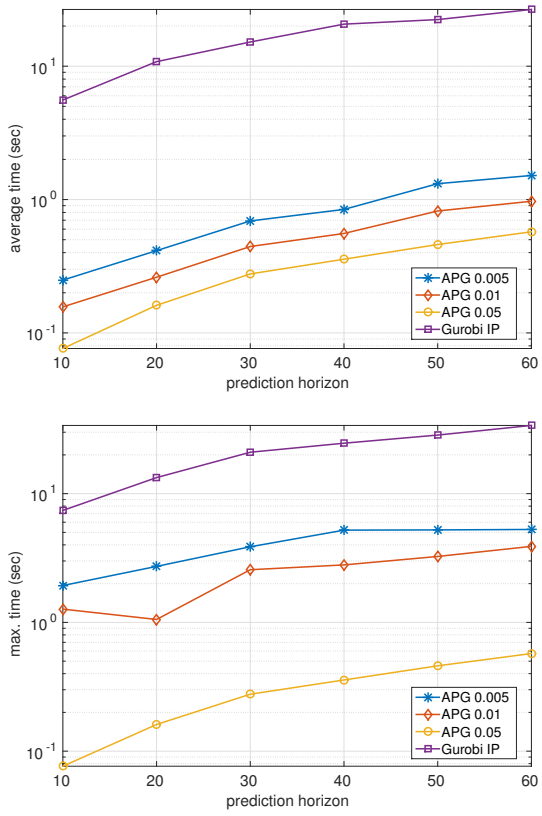

Fig. 3: Dependence of the computation time on the prediction horizon for a system of 10 masses and 500 scenarios. Average and maximum computation times reported here are for a random sampling of 100 initial states $x_0 = p$.

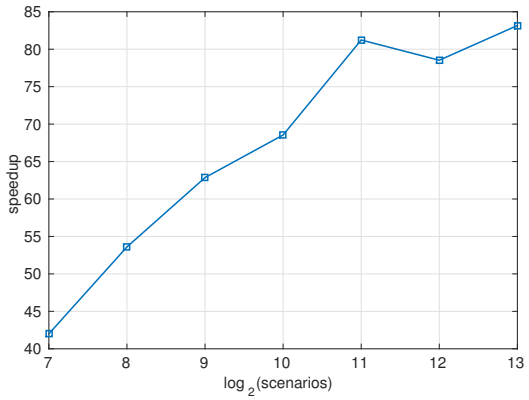

Fig. 4: Speedup with respect to a CPU implementation of APG vs. the size of the scenario tree.

switching systems," *Automatica*, vol. 50, no. 10, pp. 2504 – 2514, 2014.

[2] P. Patrinos, P. Sopasakis, and H. Sarimveis, "Stochastic model predictive control for constrained networked control systems with random time delay," in *18th IFAC World Congress*, pp. 12626–12631, 2011.

[3] A. K. Sampathirao, J. M. Grosso, P. Sopasakis, C. Ocampo-Martinez, A. Bemporad, and V. Puig, "Water demand forecasting for the optimal operation of large-scale drinking water networks: The Barcelona case study," in *19th IFAC World Congress*, pp. 10457–10462, 2014.

[4] P. Rocha and D. Kuhn, "Multistage stochastic portfolio optimisation in deregulated electricity markets using linear decision rules," *European Journal of Operational Research*, vol. 216, no. 2, pp. 397 – 408, 2012.

[5] P. Zipkin, *Foundations of Inventory Management*. New York: McGraw–Hill, 2000.

[6] D. Song, *Optimal Control and Optimization of Stochastic Supply Chain Systems*. Advances in Industrial Control, Springer, 2013.

[7] J. Liu, H. Peyrl, A. Burg, and G. Constantinides, "FPGA implementation of an interior point method for high-speed model predictive control," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, pp. 1–8, Sept. 2014.

[8] J. Mota, J. Xavier, P. Aguiar, and P. M, "Distributed ADMM for model predictive control and congestion control," in *IEEE 51st Conference on Decision and Control (CDC)*, pp. 5110–5115, Dec. 2012.

[9] J. Rogers and N. Slegers, "Robust parafoil terminal guidance using massively parallel processing," *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 5, pp. 1336–1345, 2013.

[10] G. Constantinides, "Tutorial paper: Parallel architectures for model predictive control," in *European Control Conference*, (Budapest, Hungary), 2009.

[11] N. Gade-Nielsen, B. Dammann, and J. Jørgensen, *Interior point methods on GPU with application to model predictive control*. PhD thesis, Technical University of Denmark, Lyngby, Denmark, 2014.

[12] S. Di Cairano, M. Brand, and S. A. Bortoff, "Projection-free parallel quadratic programming for linear model predictive control," *International Journal of Control*, vol. 86, no. 8, pp. 1367–1385, 2013.

[13] C. Petra, O. Schenk, and M. Anitescu, "Real-time stochastic optimization of complex energy systems on high-performance computers," *Computing in Science Engineering*, vol. 16, no. 5, pp. 32–42, 2014.

[14] B. O'Donoghue, G. Stathopoulos, and S. Boyd, "A splitting method for optimal control," *IEEE Trans. on Control Syst. Tech.*, vol. 21, no. 6, pp. 2432–2442, 2013.

[15] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, vol. 1, no. 3, pp. 123–231, 2013.

[16] R. Rockafellar, *Convex analysis*. Princeton university press, 1972.

[17] R. Rockafellar and R. Wets, *Variational analysis*, vol. 317 of *Grundlehren der Mathematischen Wissenschaften*. Berlin: Springer-Verlag, 1998.

[18] P. Combettes and J. Pesquet, *Proximal splitting methods in signal processing*. Fixed-Point Algorithms for Inverse Problems in Science and Engineering, 2011.

[19] P. Combettes and J. Pesquet, "Proximal splitting methods in signal processing," 2010. arXiv:0912.3522v4.

[20] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$," *Sov. Math. Doklady*, vol. 27, no. 2, p. 372–376, 1983.

[21] A. Beck and M. Teboulle, "A fast dual proximal gradient algorithm for convex minimization and applications," *Operations Research Letters*, vol. 42, pp. 1–6, 2014.

[22] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," *IEEE Trans. Aut. Contr.*, vol. 59, no. 1, pp. 18–33, 2014.

[23] D. Bertsekas, *Nonlinear Programming*. Belmont, MA, USA: Athena Scientific, 2nd ed., 1999.

[24] P. Giselsson and S. Boyd, "Diagonal scaling in douglas-rachford splitting and ADMM," in *53rd IEEE Conference on Decision and Control*, (Los Angeles, CA, USA), pp. 5033–5039, 2014.

[25] H. Heitsch and W. Römisch, "Scenario reduction algorithms in stochastic programming," *Computational optimization and applications*, vol. 24, no. 2, pp. 87–206, 2003.

[26] A. Shapiro, D. Dentcheva, and A. Ruszczynski, *Lectures on Stochastic Programming: Modeling and Theory*. Philadelphia: Society for Industrial and Applied Mathematics, 2009.

[27] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *Queue - GPU Computing*, vol. 6, pp. 40–53, Mar. 2008.

[28] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Math. Progr. Comp.*, vol. 6, no. 4, pp. 327–363, 2014.

[29] SPM: Signal Processing Microelectronics, "QPC: Quadratic programming in C version 2.0," 2010. Software program.

[30] J. Li, R. S., and S. Sahni, "Strassen's matrix multiplication on GPUs," in *17th International Conference on Parallel and Distributed Systems*, 2011.