Design and Implementation of Approximate Operator Splitting Algorithms On FPGA

Anis Hamadouche, Yun Wu, Andrew M. Wallace, and João F. C. Mota
The School of Engineering & Physical Sciences
Heriot-Watt University, Edinburgh EH14 4AS

July 24, 2023

Introduction

Any algorithm requires a computational platform to run. Some platforms, however, have limited size, power, or energy, especially if they are battery-operated. Examples include cell phones, unmanned vehicles like flying drones, or tiny sensors. Yet, they need to process information to make inferences about a signal or to compute a trajectory, which is done by running algorithms that often require a considerable amount of computation. Our goal is to investigate ways to reduce the total amount of power/energy/size of an algorithm by considering the full computational stack: from the task itself, to its algorithmic implementation, basic linear algebra operations, software, and hardware.

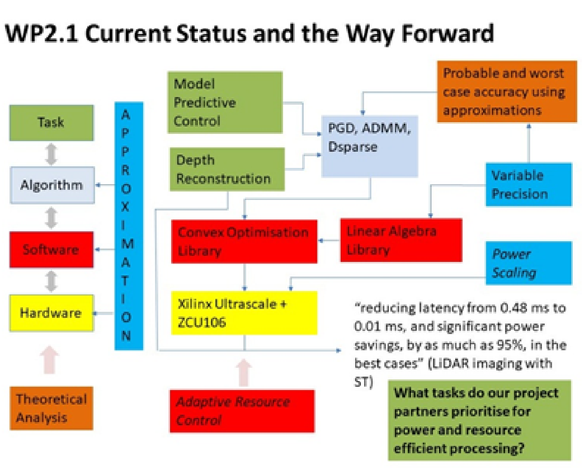

Figure 1: WP2.1 Overview.

See Fig. 1 for a visual illustration. Indeed, in each of these layers, there exist techniques to reduce power/energy/size, for instance, by reducing voltage or frequency, selecting a lower precision for representing numbers, or using approximate algorithms in linear algebra operations. However, such techniques are almost always considered in isolation without taking into account the impact on the downstream task. In this WP, we took a holistic view on this set of approximations and developed strategies to select the most effective approximations without undermining the task at hand. We thus made several experimental and theoretical contributions to the field.

Approximate Model Predictive Control[5]

The figures below show both the orientation of a NASA Calipso Satellite and the control signal in a real-time FPGA-in-the-loop (ZCU106) simulator.

Seven states are considered here: Roll, Pitch, Yaw, ω_1, ω_2, ω_3, ω_W, where Roll, Pitch, Yaw describe the rotating angles of the body frame relative to the orbit frame, and ω_1, ω_2, ω_3 are the corresponding angular velocities. ω_W is the angular velocity along the spin axis. The thrusters are controlled by three input voltages, τ_1, τ_2, τ_3, and the reaction wheel is controlled by input voltage τ_W accordingly.

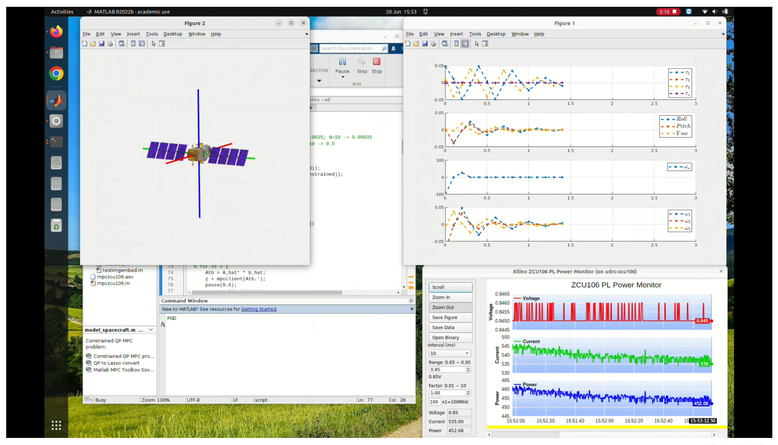

Figure 2: FPGA-in-the-loop simulation of MPC with $\boldsymbol{T} = \mathbf{10}$; $\boldsymbol{W} = \mathbf{64}$).

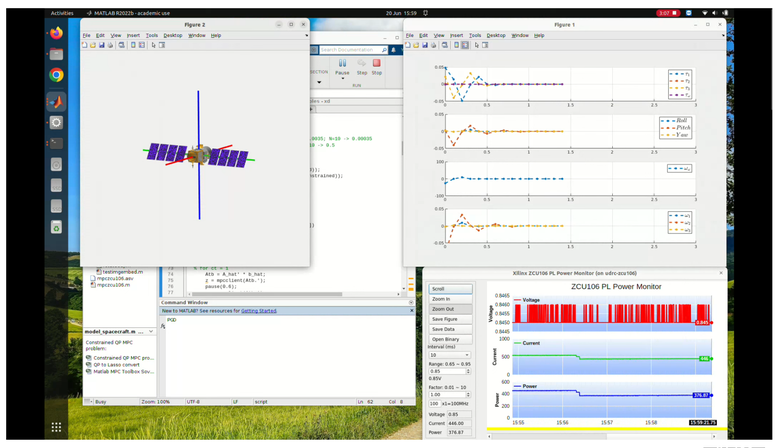

Figure 3: FPGA-in-the-loop simulation of MPC with $\boldsymbol{T} = \mathbf{10}$; $\boldsymbol{W} = \mathbf{28}$).

The thrusters are controlled by three input voltages, τ_1, τ_2, τ_3, and the reaction wheel is controlled by input voltage τ_W accordingly

The following table summarises FPGA resource utilisation, power, and latency of hardware-accelerated MPC using different arbitrary precisions and control horizons.

Table 1: Power report from post-implementation

	T=1				T=5				T=10			
Precision	FP-32	FP-12	FXP-24	UP-12	FP-32	FP-16	FXP-28	UP-14	FP-32	FP-14	FXP-32	UP-14
LUT ($\times 10^3$)	3.31	2.99	1.21	19.4	3.17	2.16	1.24	10.9	4.01	2.42	1.46	10.6
DSP48E1	30	0	11	12	20	6	10	4	20	6	16	4
BRAM	0	0	0	0	2	2	4	0	5	5	5	8
Clock (MHz)	482	465	443	393	434	401	403	382	370	384	379	382
Power (mW)	273	199	68	248	219	220	110	152	250	254	113	148
Bandwidth	100%	14.06%	56.25%	14.06%	100%	25%	76.56%	19.14%	100%	19.14%	100%	19.14%

The results show up to 60% logic cost reduction, 80% memory bandwidth saving, and 70% power reduction.

A Probabilistic Convergence Verification Framework For Hardware-based Solvers[2, 1]

Given computational errors with bounds known a priori, the proposed framework establishes probabilistic guarantees on the convergence of the algorithm up to a suboptimal noise ball. The radius of the noise ball (or approximation width) is parametrized by a probability parameter. Such a noise ball can be interpreted in a frequentist sense as the proportion of tests that fail during verification. In a Bayesian interpretation, it is equivalent to the smallest probability under which some test rule is expected to be successful in each verification session.

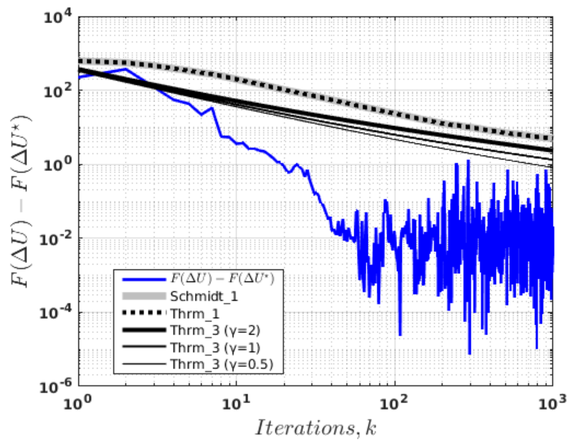

Figure 4: Approximate PGD convergence upper bounds (with gradient error bound = $\mathbf{2.2} \times \mathbf{10^1}$; proximal error bound = $\mathbf{10^1}$)

.

Design and Analysis of A Hardware-Friendly Algorithm (DFGPGD)[3]

The general form of Dual-Feedback Generalized Proximal Gradient Descent (DFGPGD) algorithm can be written as

$$\boldsymbol{u}^{k+1} = \boldsymbol{A}\boldsymbol{x}^k + \boldsymbol{B}\boldsymbol{z}^k - \boldsymbol{c} + \boldsymbol{v}^k, \tag{1a}$$

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \frac{1}{\lambda_x}\nabla \boldsymbol{g}(\boldsymbol{x}^k) - \frac{1}{\lambda_x \lambda}\boldsymbol{A}^\top \boldsymbol{L}\boldsymbol{u}^{k+1}, \tag{1b}$$

$$\boldsymbol{z}^{k+1} = \arg\min_{\boldsymbol{z}} \quad \boldsymbol{h}(\boldsymbol{z}) + \frac{1}{2}\|\boldsymbol{z} - \Lambda_{2_k}^{-1}\gamma_{2_k}\|^2_{\Lambda_{2_k}} \tag{1c}$$

$$\boldsymbol{v}^{k+1} = \boldsymbol{v}^k + (\boldsymbol{A}\boldsymbol{x}^{k+1} + \boldsymbol{B}\boldsymbol{z}^{k+1} - \boldsymbol{c}), \tag{1d}$$

In this experiment, we used **5** iterations in both ADMM and DFGPGD solvers for comparison, this is referred to as trip count in Vivado HLS timing reports. The iteration latency is reduced from **1401** cycles in ADMM to only **1052** cycles using DFGPGD, which is **23.39**% reduction in overall latency (i.e., from **7329** cycles to **5615** cycles as shown in Table 2 below).

Table 2: Timing report

	Latency (cycles)	CP post-synth. (ns)	CP post-impl. (ns)
ADMM	**7329**	**4.305**	**6.372**
DFGPGD	**5615**	**3.195**	**4.097**
Diff.	**-23.39%**	**-25.78%**	**-35.70%**

Table 3: Resource utilization report from post-implementation

	CLB	DSP48E	FF	LUT
ADMM	339	16	1577	1822
DFGPGD	204	10	879	1120
Diff.	**-39.82%**	**-37.50%**	**-44.26%**	**-38.53%**

Table 4: Power report from post-implementation

	Total Power (W)	Dynamic Power (W)
ADMM	0.636	0.044
DFGPGD	0.611	0.019
Diff.	**-3.93%**	**-56.82%**

Synthesisable Approximate Linear Algebra Library (SXLAL)[4]

We developed a linear algebra library that supports various arithmetic types and precisions. Linear algebra operations are performed with various precisions. In conjunction with the linear algebra computations, the library also includes general DSP algorithms, e.g., the fast Fourier transform (FFT), discrete cosine transform (DCT), and other transforms.

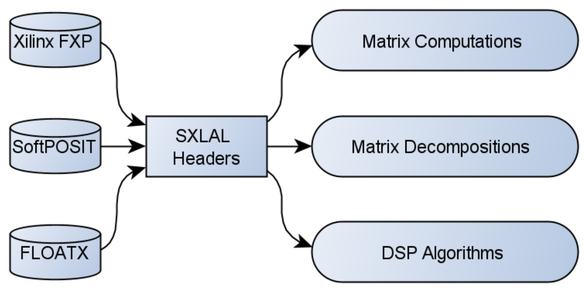

Figure 5: SXLAL headers make use of datatypes from external arithmetic libraries and is used for linear algebra operations and DSP algorithms.

References

- A. Hamadouche, A. M. Wallace, and J. F. Mota.
  Improved convergence bounds for operator splitting algorithms with rare extreme errors.
  arXiv preprint arXiv:2306.16964, 2023.
- A. Hamadouche, Y. Wu, A. M. Wallace, and J. F. Mota.
  Sharper bounds for proximal gradient algorithms with errors.
  arXiv preprint arXiv:2203.02204, 2022.
- A. Hamadouche, Y. Wu, A. M. Wallace, and J. F. Mota.
  A low-power hardware-friendly optimisation algorithm with absolute numerical stability and convergence guarantees.
  arXiv preprint arXiv:2306.16935, 2023.
- Y. Wu.
  Synthesizable approXimate Linear Algebra Library (SXLAL).
  https://github.com/wincle626/SXLAL, 2022.
- Yun Wu, Anis Hamadouche, Joao Mota, Andrew M Wallace.
  Real-time Approximate MPC Demo.
  https://github.com/wincle626/ApproximateLassoMPCDemo, 2023.