BitBench: A Benchmark for Bitstream Computing

Languages, Compilers, and Toolchains for Embedded Systems '19

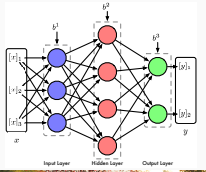
Kyle Daruwalla, Heng Zhuo, Carly Schulz, and Mikko Lipasti

23 June 2019

University of Wisconsin - Madison, Dept. of Elec. and Comp. Eng.

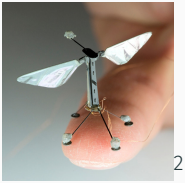
Motivation

CV/ML algorithms enable powerful new applications



[1]

Fabrication techniques enabling pico-aerial vehicles (PAVs)

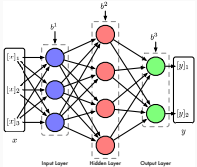


[2]

[1] Dllu 2017
[2] Ma 2015

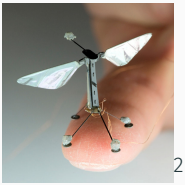
Motivation

CV/ML algorithms enable powerful new applications



1

Fabrication techniques enabling pico-aerial vehicles (PAVs)



2

How do we enable advanced CV/ML algorithms on ultra-low power platforms?

Overview

Roadmap for talk:

1. Highlight the need for bitstream computing
2. Discuss typical applications of bitstream computing
3. Present a series of benchmarks that cover typical applications (BitBench)
4. Show some initial results on the benchmarks
5. Discuss potential future directions

Motivation and Background

Limitations of Current Approaches

PAVs have ultra-low resources constraints
(< 35 mW compute power)[3]

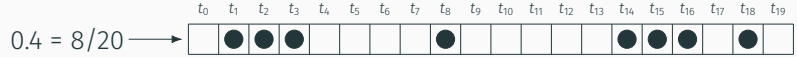
Traditional computing paradigms cannot meet these constraints!

- Prior system tethered to desktop computer
- Current on-device chip can only support basic stationary flight[4]

[3] Duhamel et al. 2011.
[4] Zhang et al. 2017.

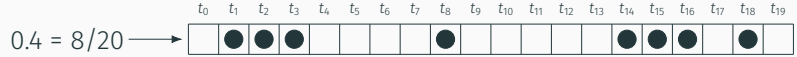
What is Bitstream Computing?

Stochastic Bitstreams:

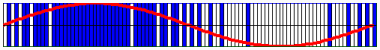


What is Bitstream Computing?

Stochastic Bitstreams:



Deterministic Bitstreams:



Density of “1” $\Rightarrow$ Higher amplitude

- Sequence is *deterministic*
- Oversampled audio data
- Leads to efficient filters

Applications of Bitstream Computing

Looking at RoboBee[5]

A flying robot should be able to:

1. Mimic biological bees
2. Perform aerial surveillance
3. Collect weather data
4. Assist search and rescue
5. Communicate with other robotic bees

[5] Floreano and Wood 2015.

Looking at RoboBee[5]

A flying robot should be able to:

1. Mimic biological bees (object tracking, optical flow, inverse kinematics)
2. Perform aerial surveillance (optical flow, inverse kinematics)
3. Collect weather data (optical flow, inverse kinematics)
4. Assist search and rescue (object tracking, homography decomposition)
5. Communicate with other robotic bees (audio filtering)

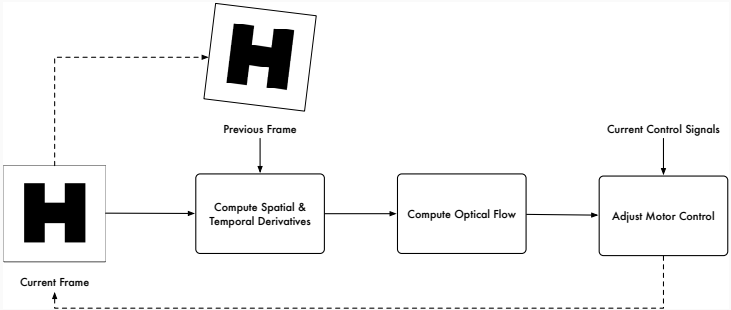
[5] Floreano and Wood 2015.

Example Applications

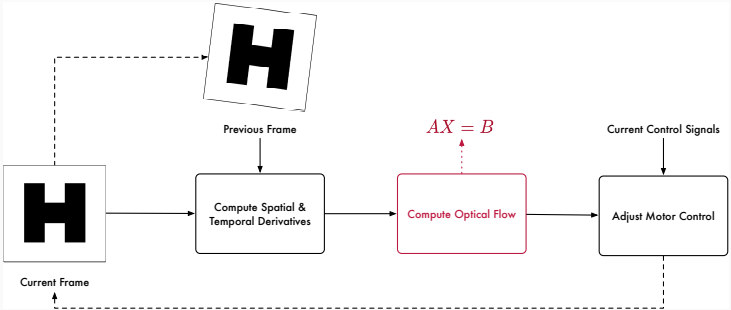
Introduce kernels with three examples:

1. Drone stabilization
2. Navigation and path planning
3. Channel shaping

An Example: Drone Stabilization



An Example: Drone Stabilization



Algorithm 1 Linear Solver (Pseudoinverse)[6]

Require: Input matrices $A \in \mathbb{R}^{m \times p}$, $B \in \mathbb{R}^{m \times n}$, Step-size $\alpha > 0$

1: **for** $k = 1 \dots T$ **do**
2: $\quad X_{k+1} \leftarrow (I - \alpha A^\top A) X_k + \alpha A^\top B$
3: **end for**
4: **return** X_T, final solution

[6] Shukla et al. 2018.

Linear Solver

Algorithm 2 Linear Solver (Pseudoinverse)[6]

Require: Input matrices $A \in \mathbb{R}^{m \times p}$, $B \in \mathbb{R}^{m \times n}$, Step-size $\alpha > 0$

1: **for** $k = 1 \ldots T$ **do**
2: $\quad X_{k+1} \leftarrow (I - \alpha A^\top A) X_k + \alpha A^\top B$
3: **end for**
4: **return** X_T, final solution

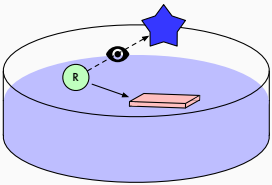
Used to implement:

- Optical flow
- Object tracking
- Inverse kinematics

Algorithm	Size of A	Size of B
Optical Flow	4×2	4×1
Object Tracking	3×3	3×3
Inverse Kinematics	2×2	2×1

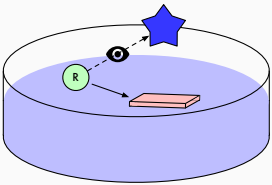
[6] Shukla et al. 2018.

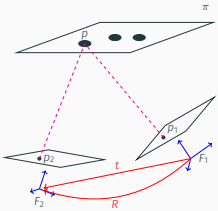
Consider a robot (denoted “R”) that needs to navigate an unknown environment



[7] Malis and Vargas 2007.

Consider a robot (denoted “R”) that needs to navigate an unknown environment

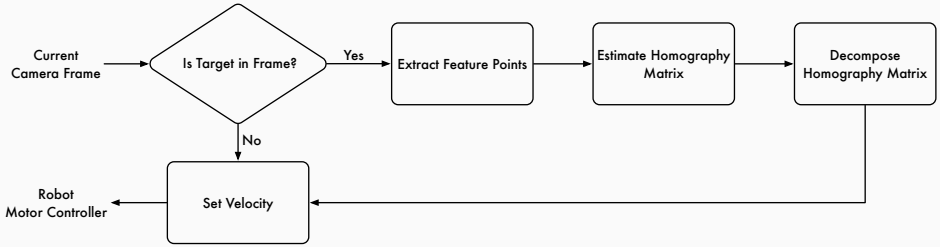




Navigation by visual cues — homography estimation and decomposition[7]

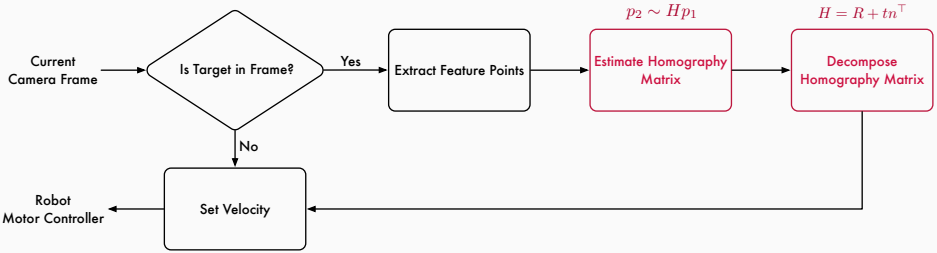
[7] Malis and Vargas 2007.

Navigation Pipeline



[8] Dubrofsky 2009.

Navigation Pipeline



Requires linear solver and singular value decomposition[8]

[8] Dubrofsky 2009.

Algorithm 3 Iterative SVD[9]

Require: Input matrix $A \in \mathbb{R}^{m \times n}$ and initial guess $v_0 \in \mathbb{R}^n$

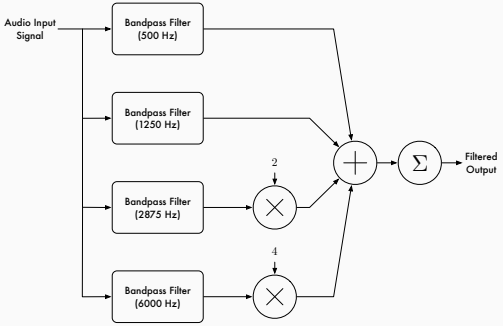
1: **for** $k = 1, 2, \ldots$ (until convergence) **do**
2: $\quad w_k = A v_{k-1}$
3: $\quad \alpha_k = \|w_k\|_2 = \sqrt{w_k^\top w_k}$
4: $\quad u_k = w_k / \alpha_k$
5: $\quad z_k = A^\top u_k$
6: $\quad \sigma_k = \|z_k\|_2 = \sqrt{z_k^\top z_k}$
7: $\quad v_k = z_k / \sigma_k$
8: **end for**
9: **return** First left/right singular vectors, u_k & v_k, and first singular value, σ_k

[9] Bentbib and Kanber 2015.

Channel Shaping

Audio information is sent across several frequency channels

Need to shape input signal to focus on channels of interest

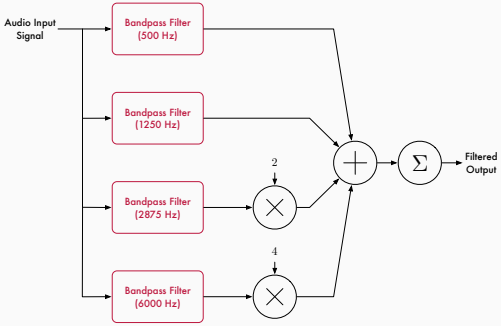


Channel Shaping

Audio information is sent across several frequency channels

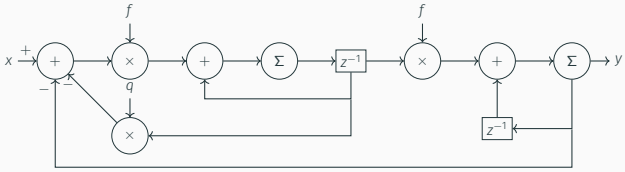
Need to shape input signal to focus on channels of interest

How do we create efficient filters?



Traditional Filters

Consider the digital state-variable filter (SVF)[10]



Filter coefficients given by:

$$f = 2 \sin\left(\frac{\pi F_c}{F_s}\right)$$

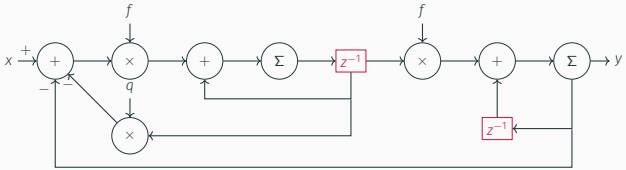
$$q = \frac{1}{Q}$$

Parameter	Symbol
Center Frequency	F_c
Sample Frequency	F_s
Quality Factor	Q

[10] Redmond 2003.

Traditional Filters

Consider the digital state-variable filter (SVF)[10]



Filter coefficients given by:

$$f = 2 \sin\left(\frac{\pi F_c}{F_s}\right)$$

$$q = \frac{1}{Q}$$

Choose F_s by adjusting delay blocks to get better coefficients!

[10] Redmond 2003.

Bitstream Filtering

First choose f and q to be binary-friendly (e.g. $f = 0.125$ and $q = 1.875$)

Set effective sample frequency according to[11]:

$$F_s = \frac{\pi F_c}{\arcsin(f/2)}$$

$$d = \left\lceil \frac{F_{\text{PDM}}}{F_s} \right\rceil$$

Parameter	Symbol
Center Frequency	F_c
Sample Frequency	F_s
PDM Sample Rate	F_{PDM}
Delay	d

[11] Lipasti and Schulz 2017.

Bitstream Filtering

First choose f and q to be binary-friendly (e.g. $f = 0.125$ and $q = 1.875$)

Set effective sample frequency according to[11]:

$$F_s = \frac{\pi F_c}{\arcsin(f/2)}$$

$$d = \left\lceil \frac{F_{\text{PDM}}}{F_s} \right\rceil$$

Parameter	Symbol
Center Frequency	F_c
Sample Frequency	F_s
PDM Sample Rate	F_{PDM}
Delay	d

Using binary-friendly coefficients results in 4-bit FXP units instead of 16-bit units

[11] Lipasti and Schulz 2017.

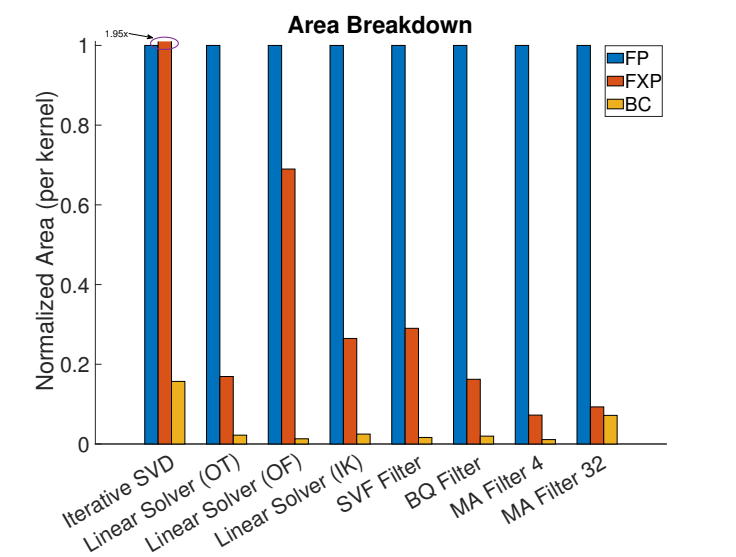
BitBench Kernels

Summary of Kernels

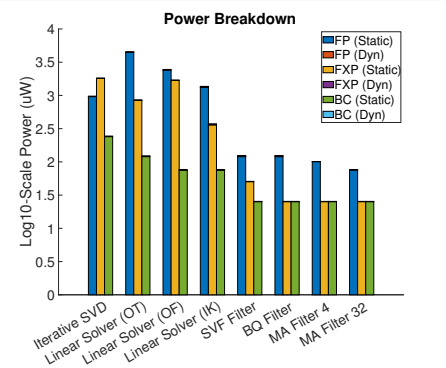
We present the following kernels:

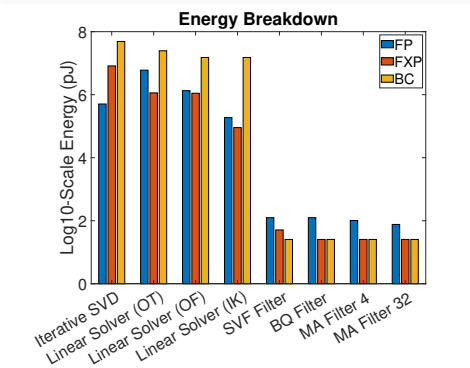
- Linear Solver (Optical Flow)
- Linear Solver (Object Tracking)
- Linear Solver (Inverse Kinematics)
- Iterative SVD
- State-Variable Filter
- Bi-Quad Filter
- Moving Average Filter (Width of 4)
- Moving Average Filter (Width of 32)

Area Results

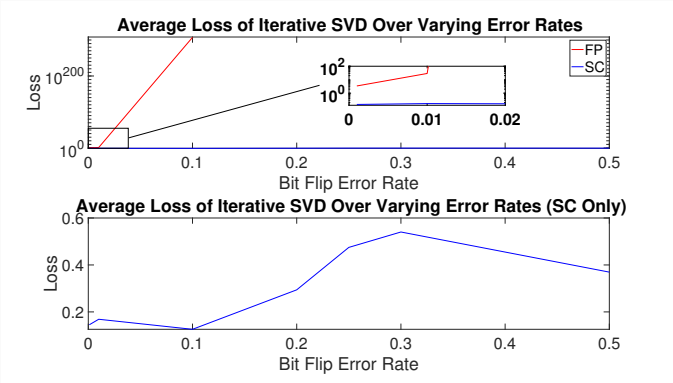


Power and Energy Results





Fault Tolerance



Conclusions

Ultra-low power/resource constraint applications require new computing paradigm

Bitstream computing is:

- Low resource
- Ultra-low power
- Under-researched (opportunity for novel work)

BitBench introduces 8 kernels centered on:

- Linear solvers
- Singular value decomposition
- Filtering

Conclusions

Ultra-low power/resource constraint applications require new computing paradigm

Bitstream computing is:

- Low resource

Check out the benchmarks at: `https://github.com/UW-PHARM/BitBench`
Built using our domain-specific language for bitstream computing (`https://github.com/UW-PHARM/BitSAD`)

- Linear solvers
- Singular value decomposition
- Filtering

Questions?

References i

- Bentbib, A. H. and A. Kanber (2015). “Block power method for SVD decomposition”. In: *Analele Stiintifice ale Universitatii Ovidius Constanta, Seria Matematica* 23.2, pp. 45–58. ISSN: 18440835. DOI: 10.1515/auom-2015-0024.
- Dllu (2017). URL: https://commons.wikimedia.org/wiki/File:Waymo_Chrysler_Pacifica_in_Los_Altos,_2017.jpg.
- Dubrofsky, Elan (2009). “Homography Estimation”. PhD thesis. Carleton University.
- Duhamel, Pierre Emile et al. (2011). “Hardware in the loop for optical flow sensing in a robotic bee”. In: *IEEE International Conference on Intelligent Robots and Systems*, pp. 1099–1106. ISSN: 2153-0858. DOI: 10.1109/IROS.2011.6048759.

References ii

- Floreano, Dario and Robert J. Wood (2015). “Science, technology and the future of small autonomous drones”. In: *Nature* 521.7553, pp. 460–466. ISSN: 14764687. DOI: 10.1038/nature14542.
- Lipasti, Mikko and Carly Schulz (2017). *End-to-End Stochastic Computing*. Austin, TX, USA. URL: http://pharm.ece.wisc.edu/papers/wp3_2017.pdf.
- Ma, Kevin Y. (2015). *RoboBee*. URL: http://www.aboutkevinma.com/index.html#publications (visited on 04/01/2018).

References iii

- Malis, Ezio and Manuel Vargas (2007). “Deeper understanding of the homography decomposition for vision-based control”. In: *Sophia* 6303.6303, p. 90. ISSN: 0036-8075. DOI: 10.1126/science.318.5857.1691b. URL: http://hal.archives-ouvertes.fr/inria-00174036/.
- Redmond, Nigel (2003). *The digital state variable filter*. URL: http://www.earlevel.com/main/2003/03/02/the-digital-state-variable-filter/.
- Shukla, Rohit et al. (2018). “Computing Generalized Matrix Inverse on Spiking Neural Substrate”. In: *Frontiers in Neuroscience* 12, p. 115. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00115. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00115.

References iv

Zhang, Xuan et al. (2017). “A Fully Integrated Battery-Powered System-on-Chip in 40-nm CMOS for Closed-Loop Control of”. In: *IEEE Journal of Solid-State Circuits* 52.9, pp. 2374–2387. DOI: 10.1109/JSSC.2017.2705170.