Processing System Design for a Linear Quadratic Controller

To optimize the Real-Time Correction of High Wind-blown Turbulence

Mike K. Kim

Contributors: S. Mark Ammons, Brian Hackel, Lisa A. Poyneer

October 8, 2019

LLNL-PRES-792467

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Low-Latency Adaptive Mirror System (LLAMAS)

- Low latency, real-time, closed-loop, woofer-tweeter Adaptive Optics Control (AOC) system
- Multi-disciplinary team (Optics, AO, Fluid Dynamics, Control Systems, Software)
- Based on work developed for the Gemini Planet Imager (GPI) with a feedback control update rate of 1 kHz
- Feedback control update rate of greater than 16 kHz
- Entirely developed with Commercial Off-the-Shelf (COTS) components.

Processing System Design for Implementing a Linear Quadratic Gaussian (LGQ) Controller

- Software Implementation for the LQG Controller consists of matrix multiplications with variable element sizes. Deterministic timing is more important than average latency.

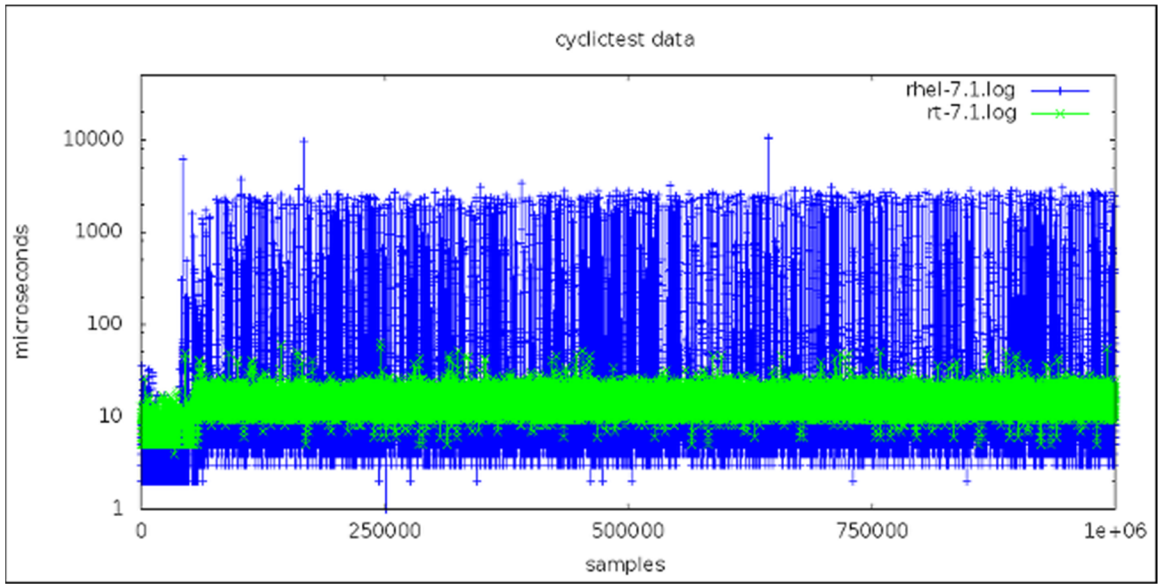
- Hardware selection (Server)

	CPU	Clock (GHz)	Cores	Cache (MB)	RAM (GB)
LLAMAS	Intel Xeon Platinum 8158	3.0	3 x 12	24.75	128
GPI	Intel Xeon E7440	2.4	4 x 4	16	32

  - +Cores: Allows processing power to implement an LQG for the Higher Order turbulence
  - +Cache: helps with core affinity

- Operating System Selection
  - GPI: Wind River Real-Time Core
    - Linux kernel executes as low-priority on a real-time executive
    - Prevented use of open source math libraries for LQG implementation
      - 600 lines of C-Code with fixed matrix dimensions
  - LLAMAS: Red Hat Enterprise Linux for Real Time. Required architecture change

Red Hat Enterprise Linux for Real Time to Optimize Latency (and Determinism)



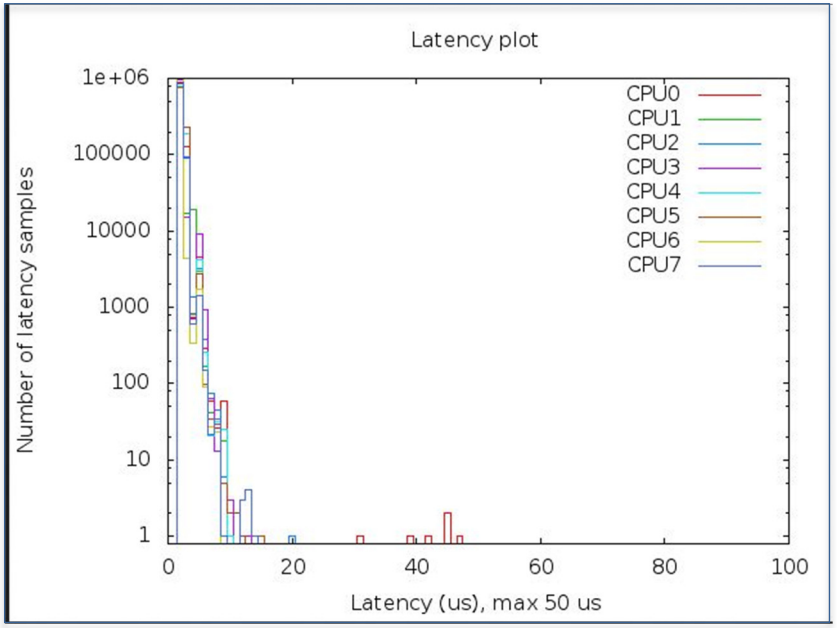
← low variance

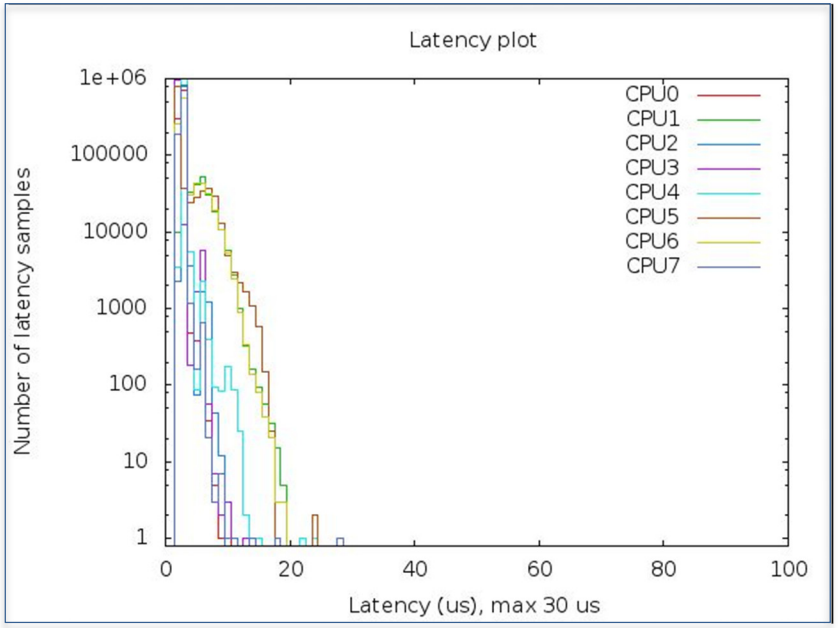
cyclictest: open source tool to measure system latency

Source: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/7/html-single/installation_guide/index

LLAMAS Cyclictest Results: Tuning for Determinism

Iterative Hardware (HP) and OS (Red Hat) Tuning





Before Tuning:
Lower average latency

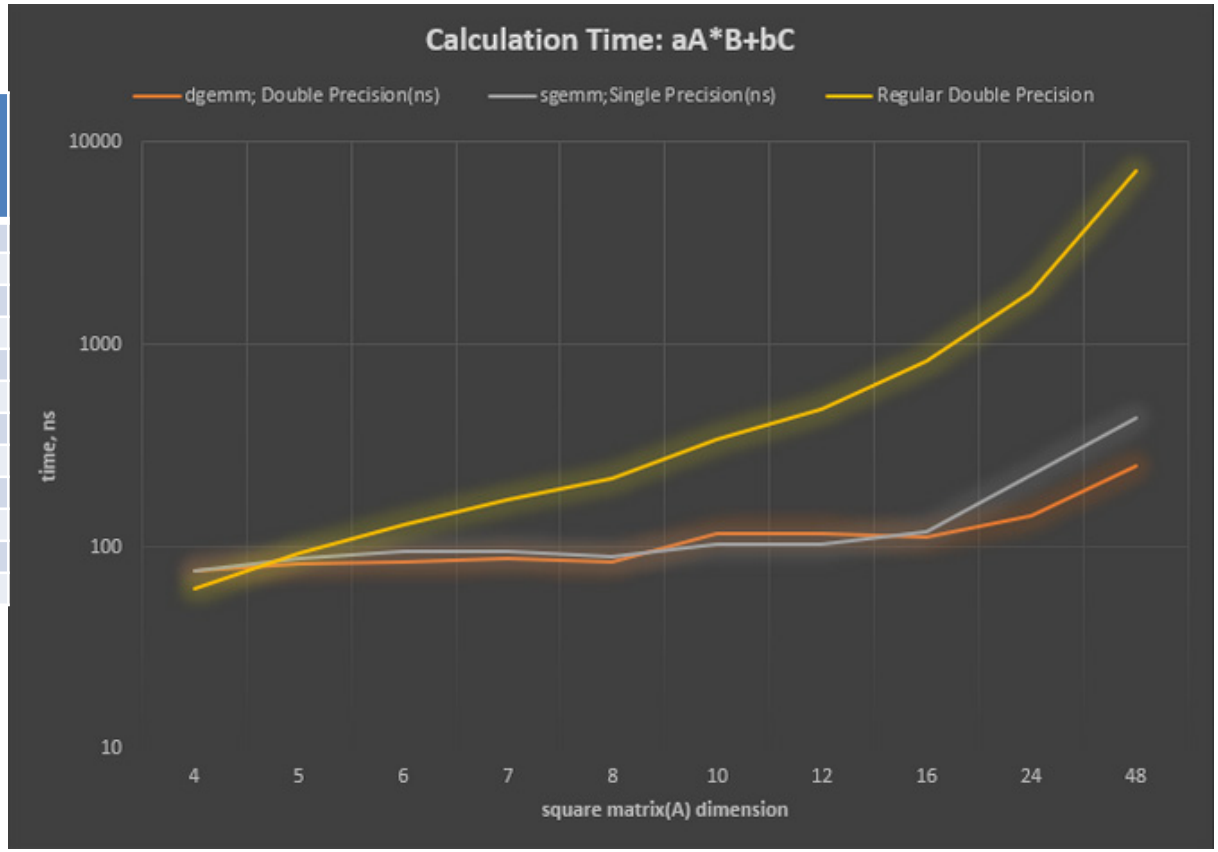
After Tuning:
More deterministic

Intel Math Kernel Library Timing (dgemm, sgemm, C-code)

LLAMAS System Results

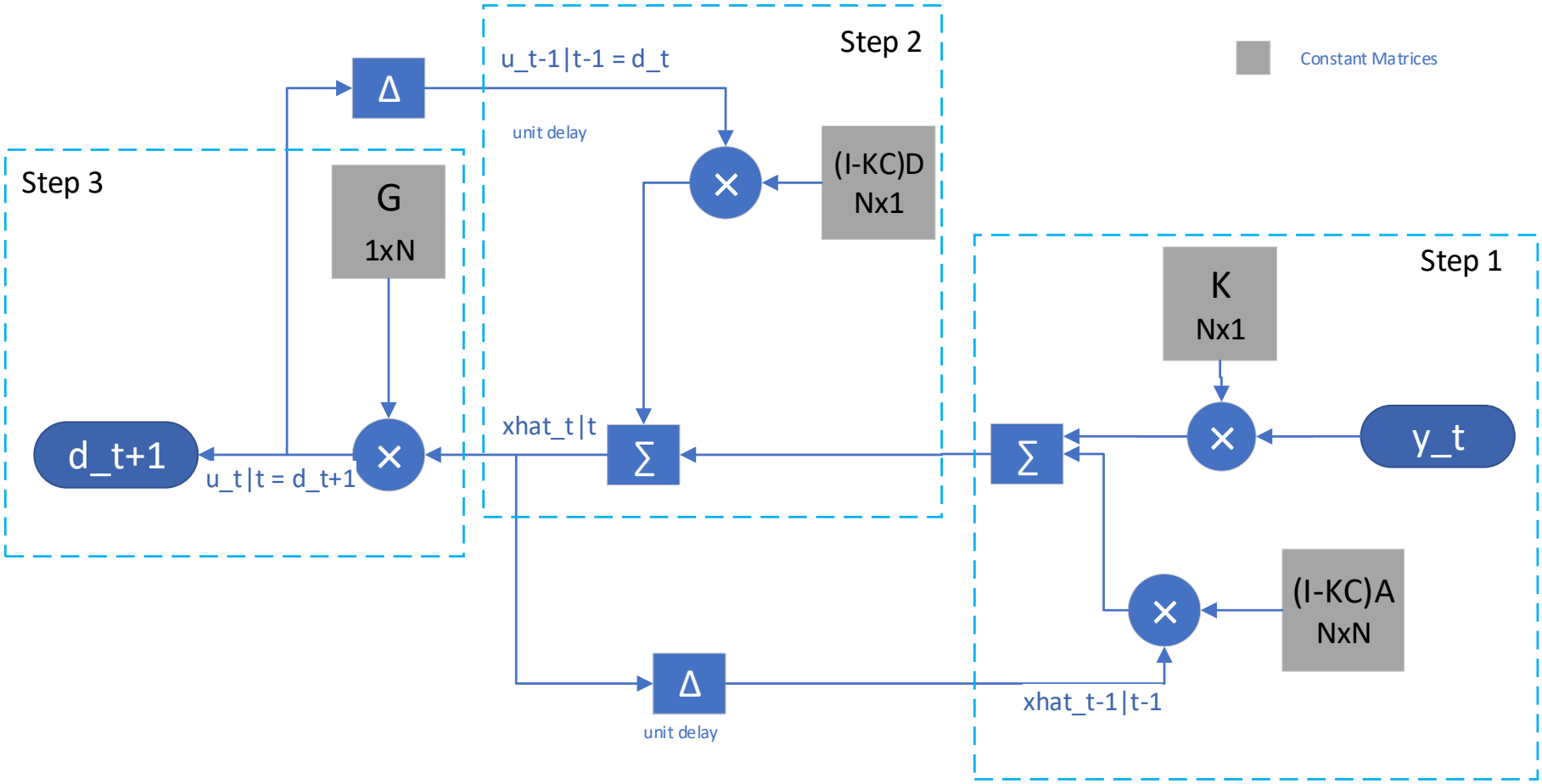
N	dgemm; double precision (ns)	sgemm single precision (ns)	C–code double precision (ns)
4	77	77	62
5	82	88	93
6	84	95	129
7	88	96	172
8	85	90	218
10	116	103	344
12	116	103	484
16	113	119	833
24	142	227	1831
48	250	437	7186
64	404	609	12763
128	1552	1920	51389

Scalar: a, b
N x N Matrix: A
N x 1 Matrix: B, C



- Intel Benchmark timing started at dimensions of 256
- MKL takes advantage of the Advanced Vector Extensions(AVX2) in the processor which utilizes the Fused Multiply-Add (FMA) hardware

LLAMAS LQG Flow Diagram



Lisa A. Poyneer

LQG Implementation Timing by Data size/type

Step 1: C = (I-KC)A * xhat_t-1 + y_t * K
cblas_dgemm(… ,….);

Step 2: xhat_t = C + d_t * (I-KC)D
#if MKL
cblas_daxpy(..)
#else if GENERIC
for (i = 0; i < (lqgN_S*lqgN_D); i++) {
M_C[m*lqgN_S*lqgN_D + i] += d_t[m] * M_IKCD[i];
}
#else // Optimized because D only has one non-zero element
M_C[m*lqgN_S*lqgN_D + 0] += d_t[m] * M_IKCD[0];
#endif

Step 3: d_t+1 = xhat_t * G
#if MKL
d_t[m] = **cblas_ddot(..)**
#else if GENERIC // regular multiplication, non optimized
d_t[m] = 0;
for (i = 0; i < (lqgN_S*lqgN_D); i++) {
d_t[m] += M_C[m*lqgN_S*lqgN_D + i] * M_G[i];
}
#else // Optimize because G only has one non-zero element
d_t[m] = M_C[m*lqgN_S*lqgN_D + 3] * M_G[3];
#endif

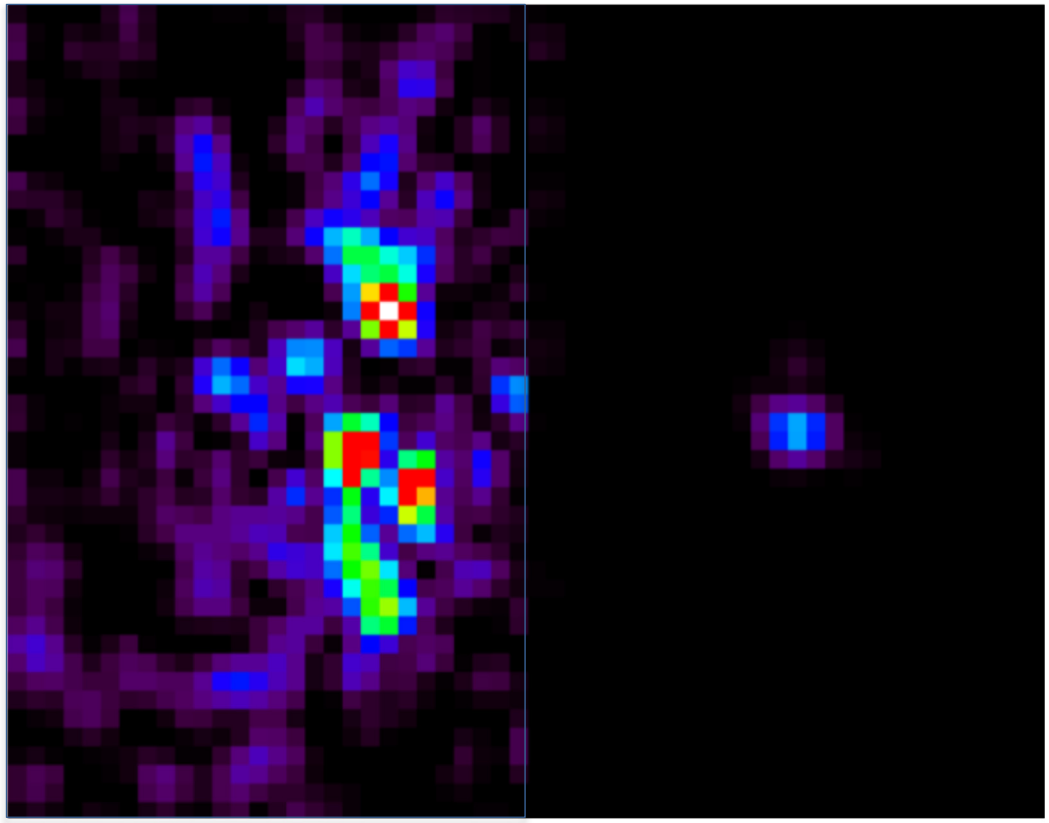
Step 4: Setup for the next frame
Save state for the next frame xhat_t → xhat_t-1; d_t+1 → d_t
Initialize the Kalman Gain vector

Data Size/Type	Modes	Step 1	Step 2	Step 3	Total Time (µs)	Time per mode (ns)
Double/ Real	576	cblas_ dgemm	cblas_ daxpy	cblas_ ddot	84	146
Double/ Real	576	cblas_ dgemm	optimized	cblas_ ddot	76	132
Double/ Real	576	cblas_ dgemm	C-code	C-code	84	146
Double/ Real	576	cblas_ dgemm	optimized	optimized	62	108
Double/ Real	3	cblas_ dgemm	C-code	C-code	0.45	150
Double/ Real	3	cblas_ dgemm	cblas_ daxpy	cblas_ ddot	0.42	140
Double/ Real	3	cblas_ dgemm	optimized	optimized	0.43	143
Double/ Complex	576	cblas_ zgemm	cblas_ zaxpy	cblas_ zdotu	103	179
Double/ Complex	576	cblas_ zgemm	optimized	optimized	88	153
Float/ Complex	576	cblas_ cgemm	cblas_ caxpy	cblas_ cdot	90	156
Float/ Complex	576	cblas_ cgemm	cblas_ zaxpy	cblas_ zdotu	76	132

MKL vs. C-code Timing Comparison

Any size matrices! & Reduced Code Size!

Open Loop vs. Closed Loop Correction



Open Loop
(Amplified scale)

Closed Loop

Mark Ammons

Latest Progress and Next Steps

23 microsec HO LQG computation time -> 10kHz frame rate

Next Steps (many opportunities to improve)

- Implement MKL multi-threading
- Optimize the HO LQG algorithm
- Optimize parallel processing
- Investigate GPU or FPGA matrix multiplication
- Revisit Red Hat Tuning suggestions [7]
- Use profiling tools to identify further optimizations
- Upgrade system to handle increased turbulence

Questions?