ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPL02

PROCESSING SYSTEM DESIGN FOR IMPLEMENTING A LINEAR QUADRATIC GAUSSIAN (LQG) CONTROLLER TO OPTIMIZE THE REAL-TIME CORRECTION OF HIGH WIND-BLOWN TURBULENCE*

M. K. Kim†, S. M. Ammons, B. Hackel, L. A. Poyneer
Lawrence Livermore National Laboratory, Livermore, USA

Abstract

LLNL has developed a low latency, real-time, closed-loop, woofer-tweeter Adaptive Optics Control (AOC) system with a feedback control update rate of greater than 16 kHz. The Low-Latency Adaptive Mirror System (LLAMAS) is based on controller software previously developed for the successful Gemini Planet Imager (GPI) instrument which had an update rate of 1 kHz. By tuning the COTS operating system, tuning and upgrading the processing hardware, and adapting existing software, we have the computing power to implement a Linear-Quadratic-Gaussian (LQG) Controller in real time. The implementation of the LQG leverages hardware optimizations developed for low latency computing and the video game industry, such as fused multiply add accelerators and optimized Fast Fourier Transforms. We used the Intel Math Kernel Library (MKL) to implement the high-order LQG controller with a batch mode execution of 576 6x6 matrix multiplies. We will share our progress, lessons learned and our plans to further optimize performance by tuning high order LQG parameters.

INTRODUCTION

The development of an Adaptive Optics (AO) system to correct for high wind-blown turbulence requires a team of experts in the fields of optics, fluid dynamics, controls systems, and software. This paper focuses on the processing and software optimizations.

Increasingly, the more significant terms in adaptive optics wavefront error budgets spanning a myriad of applications are temporal wavefront errors. These are associated with an adaptive optics system' inability to keep up with ever evolving turbulence due to a deficiency in sensor update rate ("frame rate") and/or the latency in the system. The latency is the amount of time required for the wavefront sensing measurement to be completed and readout, the reconstruction computation time, and the electrical and mechanical latency in updating the deformable mirror position

The maximum frame rate is set by multiple factors, including the camera readout time, reconstruction latency, and beacon brightness, and is thus tied to the total end-to-end latency. With standard leaky integrator controllers utilizing modal gains, there is limited performance benefit to increasing the frame rate beyond a level at which the latency is 2-3 frame times. The principle strategy to reduce bandwidth and delay errors is to reduce end-to-end latency.

This manuscript is concerned with the wavefront reconstruction process in adaptive optics systems, particularly for the Low Latency Adaptive Mirror System (LLAMAS) at Lawrence Livermore National Laboratory [1]. This testbed was designed to run with minimal computational latency so as to maximize performance at high frame rates (> 16 kHz). The system utilizes Linear Quadrature Gaussian (LQG) control to maximize bandwidth at a given frame rate [2,3].

PROCESSING IMPROVEMENTS

The team focused on improving the processor-intensive portion of the AO system. Since the processing system was selected for GPI during its Preliminary Design Phase, the technology has advanced considerably. The latest servers and operating systems were researched with latency performance and determinism in mind.

Hardware Selection

The team selected the HPE ProLiant DL580 Gen 10 8SFF CTO Server. The LLAMAS server specifications and server specifications for the GPI project are listed for comparison in Table 1.

Table 1: LLAMAS vs. GPI Server Specification Comparison

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

With only a 25% improvement in clock speed, one may conclude that the performance of the software would not improve by magnitudes. This metric is no longer the primary factor when comparing performance. The increase in cores provides a performance advantage for a software architecture that uses multiple execution threads to maximize the parallel processing. Even though the LLAMAS server has less sockets (3 vs. 4), the number of cores per socket is also a processing advantage if execution multiple threads

* Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory Contract DE-AC52-07NA27344 with document release number LLNL-PROC-792238.
† kim94@llnl.gov

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPL02

on the same socket. With 36 cores available, we were comfortable with the processing power necessary to achieve the LLAMAS project performance goals.

Hardware Tuning for Low-Latency The server was tuned referencing the Hewlett Packard Enterprise, "Configuring and tuning HPE Proliant Servers for low-latency applications [4]

Table 2: Hardware Tuning Parameters

BIOS Parameter	Value
Workload Profile	Low Latency
Thermal Configuration	Optimal Cooling
Minimum Processor Idle Power Package C-State	No Package State
Minimum Processor Idle Power Core C-State	No C-States
Memory Refresh Rate	1x Refresh
Memory Patrol Scrubbing	Disabled
Intel VT-d	Disabled
Intel Turbo Boost Technology	Disabled
Intel Hyperthreading Options	Disabled
HPE Power Regulator	HPE Static High Performance Mode
HPE Power Profile	Maximum Performance
Energy/Performance Bias	Maximum Performance
Dynamic Power Capping Functionality	Disabled
Collaborative Power Control	Disabled

Operating System Selection

The GPI software was designed using the Wind River Real-Time Core [5]. Since this operating system was no longer available, the team had to select a different OS. In order to maintain a similar software architecture, the team searched for a similar OS solution that had the Linux kernel executing as low-priority on a real-time executive. This would have allowed us to easily port the prior existing software. After a thorough trade study, we decided to re-architect the software and use the latest Red Hat Enterprise Linux for Real Time. This OS is designed for low-latency determinism which provides low variance response times.

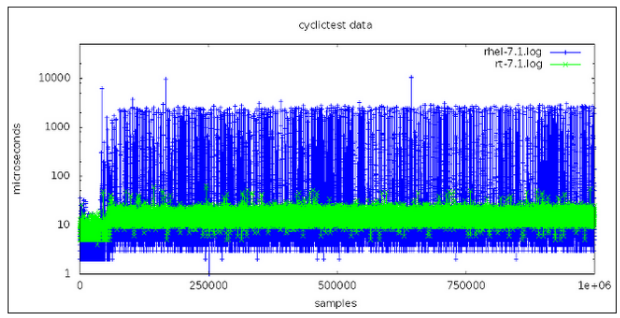

Figure 1: Benefit of Using Realtime over Standard Kernel System Tuning [6].

Figure 1 compares a million samples of machines using the Red Hat Enterprise Linux 7 and Red Hat Enterprise Linux for Real Time kernels respectively. The blue points in this graph represent the system response time (in microseconds) of machines running a tuned Red Hat Enterprise Linux 7 kernel. The green points in the graph represent the system response time of machines running a tuned realtime kernel. The figure indicates that the response time of the realtime kernel is very consistent, in contrast to the standard kernel, which has greater variability with points scattered across the graph [6].

OS Tuning for Real-Time Red Hat provide tools and documentation to tune the OS for latency and determinism. We followed the Red Hat Enterprise Linux for Real Time Tuning Guide [7].

We used the "tuned-adm" command to experiment with the various predefined profiles. We set the profile to "realtime". It appeared the various profiles did not make a significant impact to the performance of the LLAMAS software. We plan on revisiting the investigation of the OS tuning. We also experimented with the following:

- Setting the smp_affinity mask to isolate interrupt requests to specific cores
- Used "isolcpus" to isolate CPUs from the kernel scheduler

Tuning Results (Cyclictest)

To measure the effects of tuning our system, we used Cyclictest which is included in the RedHat rt-tests package. The following Cyclictest latency plots [8] in Figure 2 demonstrate the improvement in the system after tuning (Figure 3) the CPU and operating system. Note that we tuned the processing system to optimize determinism as opposed to average low latency. Processors are tuned by default for low latency for typical server applications. For closed loop control systems implementations, any latency greater than the correction rate results in data loss. This data loss results in reducing the performance of the closed loop system.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPL02

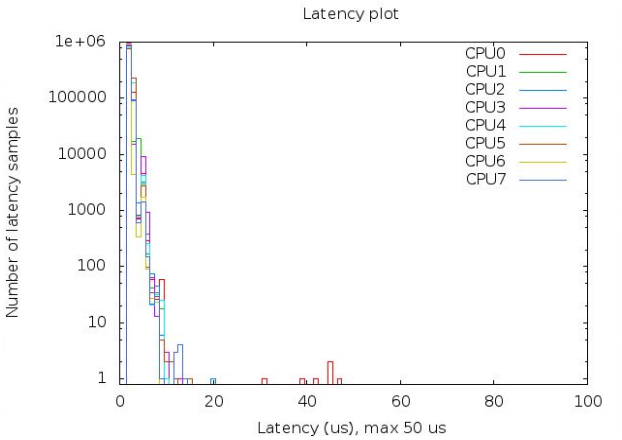

Figure 2: Cyclictest Histogram, before tuning.

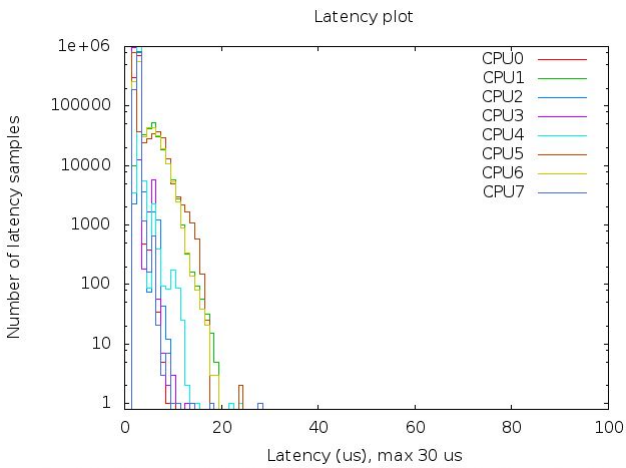

Figure 3: Cyclictest Histogram, after tuning for determinism.

LLAMAS SOFTWARE ARCHITECTURE

Software Architecture Modifications

Due to the selection of Red Hat RT Linux, the software architecture needed to be modified from the WindRiver Linux architecture used on the prior project. With the increase in cores, we were able to experiment with assigning execution threads to various cores and sockets. With minimal experimentation, we were able to verify the thread groupings on specific sockets. As expected, grouping execution threads on the same socket improved performance. We also set the processor or core affinity of the LLAMAS execution threads to take advantage of the saved cache and thus improving performance. The repetitive nature of the LLAMAS AO processing architecture fully takes advantage of cache memory.

PREDICTIVE ADAPTIVE OPTICS

Increasing system frames rates beyond the 1 kHz that was achieved with astronomical AO systems could significantly reduce bandwidth wavefront error (WFE) and improve contrast. Predictive control algorithms provide a second way to "keep up" with fast-moving turbulence. Predictive methods use an internal model of the AO system and turbulence dynamics and explicitly predict ahead given the known system delays. This requires accurate knowledge of the AO system itself, as well as robust "identification" of the aberration of interest. Our baseline approach to predictive control uses a Linear-Quadratic-Gaussian (LQG) controller. This approach has already been experimentally validated in GPI's AO system, where it provided excellent correction of large focus vibrations at specific temporal frequencies.

LINEAR-QUADRATIC-GAUSSIAN CONTROL (LQG)

Tip, tilt and focus can be corrected with a LQG controller to selectively suppress vibrations, even at high temporal frequencies. Technical details on the LQG algorithm can found in Poyneer's research article, "Performance of the Gemini Planet Imager's adaptive optics system" [9]. With the increased processing power of the LLAMAS system, we were able increase the frame rate which resulted in increased performance.

MATRIX MULTIPLICATION

The software implementation of an LQG controller consists of several matrix multiplications which can be one of the most processor intensive operations in the AO system. Matrix multiplication can be implemented on the CPU with math libraries, or a Graphical Processing Unit (GPU), or a Field-Programmable Gate Array (FPGA). For cost and time constraints, we chose to pursue an implementation with a Linux standard math library.

Intel Math Kernel Library (MKL)

After researching various Linux math libraries to implement to support matrix math, we settled on the Intel MKL library which leveraged the hardware selected for the project.

The Intel MKL takes advantages of the Advanced Vector Extensions 2(AVX2) features in the Intel Xeon processor. The AVX2 utilizes the fused multiply-add (FMA) hardware feature that greatly increases the performance of the multiply-add operations that are extensively used in matrix multiplication [10].

The published performance benchmarks of some of the MKL functions start with matrices of dimension 256 x 256. It was anticipated the matrix dimensions would be less than 128. A test application was written to provide the timing of the dgemm and sgemm functions for various size matrices. Refer to the results in Table 3 which are graphed in Fig. 4.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPL02

Table 3: Test Application MKL Timing 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

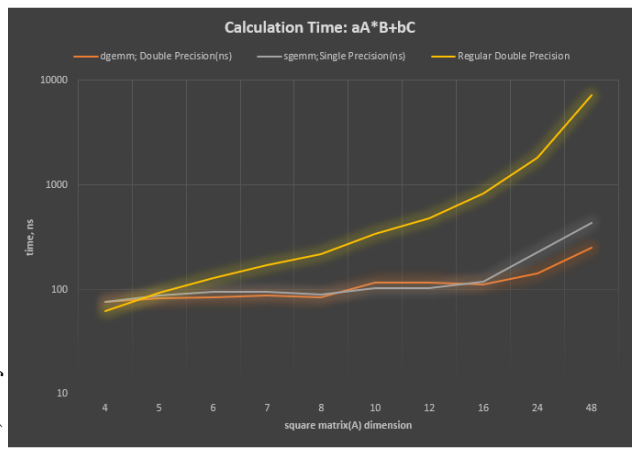

Figure 4: MKL Timing.

LQG DESIGN

Low-Order LQG

The fundamental LQG algorithm implemented in LLAMAS is based on the GPI approach. However, extra processing power enabled LLAMAS to use a more general, matrix-based implementation that provides flexibility to change filter order and aberration models. In contrast, the former implementation was hard-coded to work with a single aberration model. This more general framework is based on standard LQG equations, such as those given in Looze, "Minimum variance control structure for adaptive optics systems" [11].

The LQG block diagram or flow diagram was provided in Fig. 5.

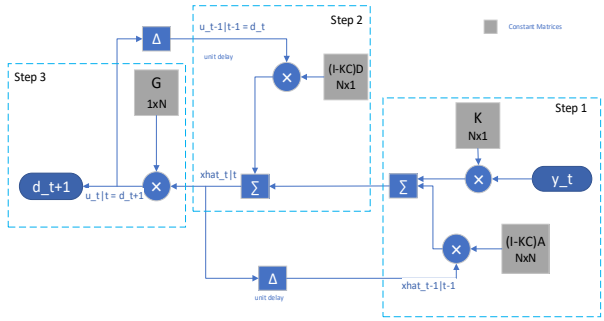

Figure 5: At each time step, the tip or tilt residual (y_t, right side) is sent into the LQG filter. It is weighted by multiplication by the Kalman gain vector K, added to the one-step prediction of the prior state. The impact of prior commands are added, then the gain vector G determines the best mirror commands. Both G and K are calculated offline through numeric solutions to the discrete Algebraic Riccati Equations. LQG Flow Diagram courtesy of Lisa Poyneer. Double real data was used for implementing the low-order LQG with three modes (x, y, & focus).

LQG Software Implementation

The software implementation of the LQG Flow Diagram involves 3 primary steps of calculation using MKL functions. The former implementation of the LQG algorithm was about 600 lines of C-code optimized for fixed dimension matrices. Matrix math libraries were not available for the Wind River Real-Time OS. Implementation using the MKL was implemented in only about 50 lines of C-code while allowing for any size of matrix. This architecture allowed the team to easily optimize the size and processing time for optimal performance.

The following minimal pseudo code, Table 4, for the 3 steps is listed to help the reader understand the implementation.

The following four constant matrices are required for the LQG:

(I-KC)A	lqgN_S x lqgN_D matrix (I-KC)A
M_G	1 x lqgN_S vector G
M_IKCD	lqgN_D x 1 vector (I-KC)D
K	lqgN_D x 1 vector K

The values in these matrices are determined by the intensity of the turbulence. The matrices are initialized from files and can be changed while the LLAMAS software is executing. Multiple sets of matrices can be defined for various levels of turbulence. This allows for optimal performance of the LQG filter.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPL02

Table 4: LQG Pseudo-Code

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

LQG Algorithm Timing

A test application was written to determine the processing time for various models. This test application was used to select data size, data type, and matrix size. Steps 2 & 3 of the test application was modified to use the MKL function, C-code, or optimized C-code. The optimized C-code allowed for measuring the timing benefit of reducing the complexity of the LQG model. Below are the timing results for one LQG model with matrix dimensions of 6x6. Note: double precision is 8 bytes, float is 4 bytes.

The data in Table 5 helped the team determine the data type and data size while still maintaining our processing time budget.

Table 5: Test Application Timing, Matrix Dimension 6x6

Data Size/Type	Modes	Step 1	Step 2	Step 3	Total Time (us)	Time per mode (ns)
Double/ Real	576	cblas_ dgemm	cblas_ daxpy	cblas_ ddot	84	146
Double/ Real	576	cblas_ dgemm	opti-mized	cblas_ ddot	76	132
Double/ Real	576	cblas_ dgemm	C-code	C-code	84	146
Double/ Real	576	cblas_ dgemm	opti-mized	opti-mized	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	opti-mized	opti-mized	0.43	143
Double/ Com-plex	576	cblas_ zgemm	cblas_ zaxpy	cblas_ zdotu	103	179
Double/ Com-plex	576	cblas_ zgemm	opti-mized	opti-mized	88	153
Float/ Com-plex	576	cblas_ cgemm	cblas_ caxpy	cblas_ cdot	90	156
Float/ Com-plex	576	cblas_ cgemm	cblas_ zaxpy	cblas_ zdotu	76	132

Low-Order LQG Results

A visual comparison of the closed loop vs. open loop system can be found in Fig. 6.

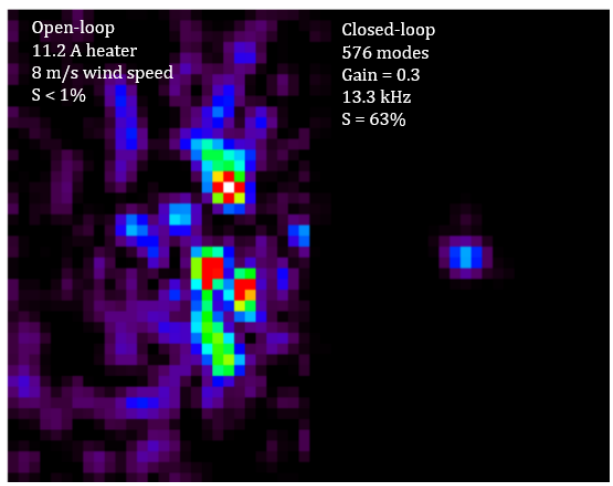

Figure 6: Instantaneous Point Spread Functions for Open-loop vs. Closed-loop LLAMAS operation using Low-Order LQG. Ideal performance of the integrated system produces narrow Point Spread Functions, as seen in the right plot.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUCPL02

HIGH-ORDER LQG CORRECTION

HO LQG Software Implementation

The HO LQG algorithm is identical to the LO LQG algorithm except the data type is different and the MKL functions are the complex versions. See Table 2. The data is complex float for the HO LQG versus double real values for the LO LQG. The HO LQG consists of 576 modes.

For high-order LQG control, we have elected to parallelize the 576 6x6 matrix multiplies over 8 cores in the LLAMAS computer. This results in a total LQG computation time of 23 microseconds, which compares favorably to (1) the current wavefront reconstruction time of 52 microseconds for the Integrator Controller and (2) the desired frame time of 67 microseconds for a goal frame rate of 16 kHz. We have been able to close all HO LQG loops stably at frame rates up to 10 kHz. In order to reach 16 kHz with HO LQG, it will be necessary to parallelize the algorithm further.

NEXT STEPS

1. Implement MKL multi-threading
2. Continue optimizing the HO LQG algorithm
3. Increase parallel processing with threads
4. Investigate GPU or FPGA matrix multiplication
5. Revisit Red Hat Tuning suggestions [7]
6. Use profiling tools to identify further optimization

REFERENCES

[1] Mark Ammons et al, "LLAMAS: Low-Latency Adaptive Optics at LLNL", Proc. SPIE 10703, Adaptive Optics Systems VI, 107031N, Jul 2018.
doi: 10.1117/12.2314281

[2] L. A. Poyneer and J.-P. Véran, "Predictive wavefront control for adaptive optics with arbitrary control loop delays," J. Opt. Soc. Am. A 25, pp. 1486–1496, 2008.

[3] L. A. Poyneer, B. A. Macintosh, and J.-P. Véran, "Fourier transform wavefront control with adaptive prediction of the atmosphere," *J. Opt. Soc. Am. A*, no. 24, pp. 2645–2660, 2007.

[4] Hewlett Packard Enterprise, "Configuring and tuning HPE ProLiant Servers for low-latency applications", Technical White Paper, PN: 581608-009, Edition 10, Oct. 2017.

[5] https://www.windriver.com/products/product-overviews/PO_RTCore_for_LX_May2009.pdf

[6] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/7/html-single/installation_guide/index#chap-Why_Use_RT_to_Optimize_Latency

[7] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_for_real_time/7/html/tuning_guide/index

[8] http://www.osadl.org/Create-a-latency-plot-from-cyclictest-hi.bash-script-for-latency-plot.0.html

[9] L. A. Poyneer, "Performance of the Gemini Planet Imager's adaptive optics system", *Applied Optics*, vol. 55, no. 2, pp. 324-325, Jan. 2016.

[10] https://software.intel.com/en-us/articles/performance-comparison-of-openblas-and-intel-math-kernel-library-in-r

[11] Douglas P. Looze, "Minimum variance control structure for adaptive optics systems", *J. Opt. Soc. Am. A*, vol. 23, no. 3, pp. 605-606, Jan. 2016.